Completion time minimization for distributed feature extraction in a visual sensor network testbed

JORDI SERRA TORRENS

KTH Electrical Engineering

Master's Degree Project
Stockholm, Sweden November 2014

XR-EE-LCN 2014:010

Abstract

Real-time detection and extraction of visual features in wireless sensor networks is a challenging task due to its computational complexity and the limited processing power of the nodes. A promising approach is to distribute the workload to other nodes of the network by delegating the processing of different regions of the image to different nodes. In this work a solution to optimally schedule the loads assigned to each node is implemented on a real visual sensor network testbed. To minimize the time required to process an image, the size of the subareas assigned to the cooperators are calculated by solving a linear programming problem taking into account the transmission and processing speed of the nodes and the spatial distribution of the visual features. In order to minimize the global workload, an optimal detection threshold is predicted such that only the most significant features are extracted. The solution is implemented on a visual sensor network testbed consisting of BeagleBone Black computers capable of communicating over IEEE 802.11. The capabilities of the testbed are also extended by adapting a reliable transmission protocol based on UDP capable of multicast transmission. The performance of the implemented algorithms is evaluated on the testbed.

Contents

1 Introduction

Computer vision has many applications such as object tracking, object recognition and classification, automatic surveillance, robot navigation and many more. With the recent advances on image sensors and the emergence of low-cost cameras, visual sensor networks have started to gain attention. Visual sensor networks consist of low-powered nodes that incorporate low-cost cameras. The sensors are typically autonomous, powered by a battery or energy harvesting, and include a wireless communication module. They are able to establish network topologies such as mesh networks and collaborate to route packets to their destination. The nodes can run unattended for long periods of time in areas where physical access is difficult. They are capable of capturing images and forwarding them to other nodes or to a central location for analysis and typically present little processing power. This has encouraged a lot of research in the area and has opened a wide range of applications. For instance, large number of nodes can be deployed in remote locations to perform surveillance of large areas. Then, alerts can be generated automatically on certain events, greatly reducing the amount of human resources that would otherwise be needed to monitor large areas. If the information from multiple cameras is combined, moving objects can be automatically tracked along their path. If an object is seen from multiple angles, a 3D reconstruction of the scene can be done.

Due to the nature of visual information, visual sensor networks present higher bandwidth and processing requirements than other types of sensor networks. Because of this, combined with the limited computational power of the nodes, the processing has to be done either in a central location with large processing power or inside the sensor network by the nodes following a collaborative scheme. The distributed processing approach can decrease the processing delay and lowers the bandwidth requirements, as the image does not need to be transmitted to a central location, which could be located multiple network hops away. In this thesis, the approach is to distribute the processing tasks by splitting the images in multiple regions and assigning their processing to different nodes. However, the task of distributing the workload in an optimal way is challenging. The size of each one of the regions, the scheduling of the nodes and other parameters need to be determined in real time.

The main focus of this thesis is to develop and implement a load balancing strategy on a visual sensor testbed. The objective is to achieve real-time analysis of captured video by optimally distributing the workload among multiple nodes of the sensor network.

1.1 Methodology

The first stage of this work was to study multiple areas relevant to our problem and the testbed. It includes the basics of visual sensor networks, visual feature extraction and the specific detectors implemented on our testbed (SURF and BRISK). Regarding the distribution of processing loads, topics such as divisible load theory, linear programming, predictors and regression models were studied. Other topics were the ASN.1 syntax and PER encoding, multi-threading and ad-hoc wireless networks. The recent literature on distributed visual feature extraction was reviewed, which includes the motivation for the solution imple-

mented in this thesis. A linear programming solver software was chosen and the solution implemented on it. Following that the testbed that serves as a basis for this work was studied. In order to support multicast communication between the nodes, which is required by the offloading mechanism, the original TCP-based communication module of the testbed had to be replaced by adapting a reliable communication module developed for a previous version of the testbed. At this point the main work of this thesis was ready to be implemented on the testbed. Following that came the evaluation of its performance on the testbed.

1.2 Report structure

The rest of the report is organized as follows. Section 2 presents a background on multiple topics underlying this thesis. It describes concepts of visual sensor networks, computer vision, distributed computing and linear programming. Section 3 states the workload balancing problem and describes its solution. Section 4 includes a description of the base testbed on which the work of this thesis is implemented. In Section 5 the implementation of the solution and the interaction with the rest of the testbed are detailed. In Section 6 the performance of proposed solution is evaluated on the testbed.

2 Background

2.1 Wireless Sensor Networks

A Wireless Sensor Network (WSN) is a network consisting of multiple autonomous spatially-distributed nodes, which gather information from their sensors and cooperatively relay this information to a central location. The nodes are typically battery operated, low-cost and low-powered devices consisting of a sensor, a microprocessor, a small amount of memory and a wireless communication interface. In some systems the nodes can be powered by solar panels.

Wireless Sensor Networks are typically deployed in remote locations where physical access is difficult, to monitor environmental parameters such as temperature, atmospheric pressure or humidity. The measured data can be used for scientific purposes, as well as, for example, generate alerts when a forest fire or a flood is detected. Another type of sensor could be motion sensors, which can be used to detect intrusions in the monitored area.

The nodes can form wireless communication networks of different complexity, ranging from simple star topologies to multi-hop mesh networks, which can route information from the remote nodes to a central location. The nodes have typically low computational power, which can be used to process the information within the network.

WSNs can be classified as homogeneous if all the nodes consist of the same hardware and characteristics, or heterogeneous if there are different types of nodes. An example of a heterogeneous WSN could be a network where some nodes include sensors and some other nodes don't include any sensor and are only used to relay information from the sensor nodes.

2.2 Visual Sensor Networks

Visual Sensor Networks (VSN) are a type of Wireless Sensor Network where the sensor nodes include cameras which are capable of acquiring visual information. VSNs can perform a wide range of computer vision tasks, such as object recognition, target tracking, 3D reconstruction and area surveillance. Examples of the application of these techniques could be traffic monitoring and remote area surveillance.

Because visual sensors produce larger amounts of information than other types of sensors, visual sensor networks typically present higher processing power and larger transmission bandwidth compared to other wireless sensor networks. As a result, nodes in VSN are more expensive and have higher energy consumption, which presents a series of challenges.

VSNs can be classified in two groups. A VSN can consist of cheap camera nodes that simply capture images and forward them to a more powerful central node, where image analysis is performed. This approach requires significant transmission bandwidth. A VSN can also consist of more expensive nodes capable of performing image analysis locally and communicating the results to a central node. This reduces the bandwidth requirements. Moreover, nodes in the VSN can cooperate in order to perform distributed image analysis, which further reduces the bandwidth requirements and processing delay. This constitutes a promising solution that could allow real-time visual analysis of video sequences.

In [1], an exhaustive review of the unique characteristics and challenges of VSNs is presented.

2.3 Visual feature extraction

Visual feature extraction is a computer vision technique consisting of detecting important regions of an image and extracting relevant information to describe them. This information can be used for multiple purposes, such as object recognition and matching, target tracking or 3D scene reconstruction.

The process can be divided in three different steps: keypoint detection, descriptor extraction and descriptor matching.

The first step, *keypoint detection*, consists in analyzing the pixel data of an image and selecting salient points based on changes of the brightness of their surrounding pixels.

In the second step, *descriptor extraction*, each one of the previously obtained keypoints is analyzed and its descriptor is obtained. A feature descriptor is a vector that summarizes the relevant information of a keypoint, which allows us to identify and compare keypoints.

Finally, *descriptor matching* is performed, which consists in comparing the obtained descriptors against the ones obtained from another image or against a database. By finding matching descriptors and their locations, one can track an object over a sequence of images. By matching the detected descriptors against a database, one can identify and classify the objects in view.

Desirable properties of local descriptors are invariance to rotation, scale, illumination, translation, as well as robustness to noise. Descriptors need to be distinctive for different keypoints and repeatable in order to detect multiple occurrences of keypoints or objects. Limited processing complexity can also be a desirable property. In some applications real-time performance is required.

There exist many different algorithms to detect keypoints and extract their feature descriptors such as *Scale-Invariant Feature Transform* (SIFT) [2], *Speeded Up Robust Features* (SURF) [3], *Features from Accelerated Segment Test* (FAST) [4], *Binary Robust Independent Elementary Features* (BRIEF) [5] or *Binary Robust Invariant Scalable Keypoints* (BRISK) [6]. In [7] and [8] different algorithms are evaluated both in terms of visual analysis accuracy and computational performance. The experimental results show that detectors based on binary descriptors, such as BRISK and BRIEF, present a significant speed-up respect to SURF and SIFT, while maintaining a comparable precision/recall. It is shown that binary descriptors constitute a very good technique for time-constrained applications.

The algorithm implemented on our testbed is BRISK. The SURF algorithm was used in a previous version of the testbed. In the following sections an overview of the two detectors is provided.

2.3.1 SURF

Speeded Up Robust Features (SURF) [3] is a scale and rotation-invariant detector and descriptor. The interest point detection is based on a blob detector, which uses a simple Hessian-matrix approximation to detect intensity changes in the image. This can be computed very efficiently using integral images.

To detect the interest points, for each point $\mathrm{x} = (x, y)$ in the image I, the Hessian matrix $\mathcal{H}(\mathrm{x}, \sigma)$ is calculated with different filter sizes.

$$\mathcal{H}(\mathrm{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathrm{x}, \sigma) & L_{xy}(\mathrm{x}, \sigma) \\ L_{yx}(\mathrm{x}, \sigma) & L_{yy}(\mathrm{x}, \sigma) \end{bmatrix},$$

where $L_{xx}(\mathrm{x}, \sigma)$ is the convolution of the Gaussian second-order derivative $\frac{\partial^2}{\partial x^2} g(\sigma)$ with image I at point x. $L_{xy}(\mathrm{x}, \sigma)$, $L_{yx}(\mathrm{x}, \sigma)$ and $L_{yy}(\mathrm{x}, \sigma)$ are defined similarly. The Gaussian second-order derivatives are approximated as box filters, which can only take the discrete values -1 and 1 and can be calculated very efficiently when using integral images. The box filter approximation of $L_{xx}(\mathrm{x}, \sigma)$ is denoted as $D_{xx}(\mathrm{x}, \sigma)$, and $D_{xy}(\mathrm{x}, \sigma)$, $D_{yx}(\mathrm{x}, \sigma)$ and $D_{yy}(\mathrm{x}, \sigma)$ can be defined similarly.

The response score is then calculated as the determinant of the approximated Hessian matrix:

$$\det(\mathcal{H}_{approx}) = D_{xx} D_{yy} - (w D_{xy})^2$$

A region is considered to be an interest point if its score is higher than a particular *detection threshold* value. The process is repeated for different σ values, which define different filter sizes according to an octave structure. To select the interest points and their scale, the responses are interpolated between neighboring octave layers.

To find the orientation of each interest point, the Haar wavelet responses within a circular neighborhood are calculated in the x and y direction and then weighted by a Gaussian centered at the interest point. Finally, for each interest point a descriptor is calculated. Descriptors describe the intensity distribution of the neighboring pixels around the interest point. To calculate them, a squared region centered around the interest point and oriented according to the previously calculated orientation is constructed. This region is divided into smaller 4x4 sub-regions. For each subregion, the Haar wavelet response is calculated at 5x5 regularly spaced sample points, weighted by a Gaussian centered at the interest point. We denote d_x as the Haar wavelet response in the horizontal direction, according to the orientation, and d_y for the vertical direction. Then, for each subregion we obtain a four-dimensional vector of floating point values $\mathrm{v} = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$. We compute this vector for each one of the 16 subregions, obtaining a 64 element long descriptor.

2.3.2 BRISK

Binary Robust Invariant Scalable Keypoints (BRISK) [6] is a method for fast feature detection, description and matching. It achieves image analysis performance comparable to SURF while presenting much lower computational complexity.

Unlike SURF, which uses floating point descriptors, BRISK uses binary descriptors. Binary descriptors are very fast to compute and have low memory requirements. This makes BRISK a very good candidate for time constrained applications such as real-time systems, or systems with energy constraints.

Interest point detection is done by the FAST 9-16 detector [4], which is a corner detector. This detector identifies an interest point if at least 9 consecutive pixels in a 16-pixel circle are sufficiently brighter of darker than the central pixel. This process is applied for each one of the octaves and intra-octaves. To find

the scale of a feature, its score at the layer of the interest point is compared to the score at the layers above and below. A quadratic function is fitted in the scale axis and its maximum is found, thus obtaining the final scale and score.

Scale invariance is achieved by selecting the scale where the score of a keypoint is highest. The scale-space consists of n octaves, denoted c_i, and n intra-octaves, denoted d_i, for $i \in \{0, 1, \ldots n-1\}$. Typically $n = 4$. The original image is denoted c_0, and the next octaves are formed by repeatedly halfsampling the original image. The intra-octaves are located between successive octaves c_i and c_{i+1}. The first intra-octave is formed by downsampling the original image by a factor of 1.5 and the following ones are formed by halfsampling the previous one.

The BRISK descriptor is a binary string resulting from simple brightness comparisons around the feature following a sampling pattern, with orientation normalization to provide rotation invariance. The resulting bit string has a length of 512 bits, with each bit corresponding to the result of a brightness comparison. Descriptor matching can be performed very efficiently as a simple Hamming distance, which is a measure of the similarity between two keypoints.

In [9] an optimized version of BRISK tailored to low-power ARM architectures is proposed, allowing energy savings close to 30% respect to the original implementation.

2.4 Distributed feature extraction in VSNs

If the nodes of a VSN have significant processing capabilities, the task of visual feature extraction can be distributed among them. This constitutes a low cost solution for performing visual analysis with lower processing delay and it allows to balance the power consumption of the nodes. As the processing is performed inside the VSN, the bandwidth requirements are lower because now only the descriptors will be transmitted to the server, instead of the pixel data.

The camera node captures an image and distributes the workload among other nodes of the VSN, referred as *processing nodes* or *cooperators*. When the computations are completed, the processing nodes transfer the descriptors to the server. The distributed computation is completed once all the nodes have finished their part. Typically, the optimal distribution is that where all the processing nodes finish their share of the task at the same time [10]. Therefore, in order to minimize the completion time, the distribution of the task must be done in an optimal way, which is a challenging task.

In the following sections different ways to delegate the detection and extraction of interest points are described.

2.4.1 Delegation of interest point detection

The detection of interest points can be delegated to the processing nodes in three different ways, as proposed in [11]:

- Area-split:

  Each node detects the interest points of an area of the image. In order to detect the interest points near the border of a region, some pixels of the neighboring region need to be known. This defines an overlapping area that needs to be transmitted to both nodes. If we consider unicast links,

the overlapping area will have to be transmitted more than once. If multicast transmission is possible, it can be used to transmit the overlapping areas, while the non-overlapping areas are transmitted by unicast.

The width of the overlap is defined by the size of the largest interest area, which depends on the largest expected scale. In SURF the overlap width is $\sqrt{2} \cdot 10$ times the largest expected scale [11]. In BRISK, the FAST detector looks at a circle with a radius of 3 pixels plus the central pixel. The original image is downsampled 24 times to obtain the largest scale, therefore the width of the overlapping region is 168 pixels.

- Scale-split:

  Each node detects all the interest points at one of the octave layers. The complete image needs to be transmitted to all the nodes.

- Hybrid-split:

  The distribution is done both in terms of area and scale.

The workload for each node will depend on the spatial distribution of the interest points, in the case of area-split; on the distribution across the octaves, in the case of scale-split; or on both, in the case of hybrid-split. In [11], the statistical characteristics of the distribution of the interest points are studied. It is shown that the distribution of the location and the scale of the interest points vary significantly between images and in order to allocate the processing in a balanced way, a-priori information on the image characteristics is required.

2.4.2 Delegation of processing steps

As proposed in [11], the delegation of the workload from the camera node to the processing nodes can be classified by the degree of involvement of the camera node.

- No Detection / No Extraction (ND/NE)

  The camera node does not perform any processing. The entire image is sent to the processing nodes, where detection and extraction are performed. The task can be divided by area-split, scale-split or hybrid-split.

- Partial Detection / Partial Extraction (PD/PE)

  The camera node detects and extracts some of the interest points. The rest are detected and extracted by the processing nodes. The distribution can be done by area, scale or both.

- Complete Detection / No Extraction (CD/NE)

  All the interest points are detected by the camera node but their descriptors are extracted by the processing nodes. Orientation can also be calculated by the camera node, which can reduce the number of pixels that need to be transmitted to the cooperators. Because the camera node knows the location and the scale of the interest points, only the pixel data relevant to their extraction needs to be transmitted. The load distribution among the nodes can be easily done.

- Complete Detection / Partial Extraction (CD/PE)

  All the interest points are detected at the camera node and some of their descriptors are extracted. The rest of the descriptors are extracted by the processing nodes.

2.4.3 Recent work on distributed feature extraction in VSNs

The challenges of distributed processing of image content in VSNs have been addressed from multiple sides in the recent literature. The main issues we are faced with are the constraints on transmission bandwidth, processing power and power consumption.

In order to minimize the amount of bits required to send an image to another node, [12] proposes a JPEG quantization table optimized for feature extraction. Their results show an improvement over the default JPEG table in terms of image analysis performance. In [13] the authors analyze different video compression techniques for VSNs from the energy efficiency viewpoint. They study both inter- and intra-frame encoding schemes. Their results show that while inter-frame encoding achieves higher compression rates, it does so at the expense of increased energy consumption, which is not suitable for low-powered nodes. In order to minimize the size of the descriptors that need to be transmitted, [14] proposes a lossless entropy coding scheme. The authors also propose a strategy to select only the most discriminative pairwise comparisons for building the binary descriptors and evaluate their discriminative performance as a function of the number of bits needed for each descriptor. [15] proposes a rate-accuracy model to maximize the network lifetime subject to a target accuracy, based on the number of selected features, the number of bits used for their quantization and the selection of only a subset of the features. In [16] the authors consider the compression of the descriptors for video sequences by means of inter-frame and intra-frame coding. Their results also show that processing the visual information locally outperforms transmitting it to a central node for processing. In [17] the aim is to minimize the amount of information that needs to be exchanged between the nodes for matching objects seen by different cameras at different times. A hierarchical distribution of the feature knowledge is proposed, and queries are routed accordingly to the nodes that have the full information stored locally. In [18] the authors aim to extend the network lifetime by deploying a Gaussian distributed relay network in addition to an already deployed VSN.

In [11] the statistical characteristics of SURF and BRISK interest points are studied, with the aim of evaluating the possibility of distributing their extraction in a distributed system. The results show that in order to properly balance the workload among the nodes, a-priori information of the image characteristics needs to be obtained. Following these results, in [19] a prediction scheme is proposed for obtaining a detection threshold that results in the extraction of a target number of interest points in a VSN. In addition to this, in [20] the authors present a linear programming model for obtaining the optimal workload distribution in an area-split scheme.

2.5 Divisible Load Theory

In distributed systems, we can exploit data parallelism to process large computational loads by partitioning the data and assigning each part to a different processor. Divisible load theory aims to obtain the optimal scheduling that results in minimal completion time [10], [21].

Loads can be classified depending on whether they can be divided in smaller loads or not. This is referred as *divisibility property*. Loads are *indivisible* when they can't be further divided in smaller tasks and therefore they have to be processed by a single processor. Loads are *modularly divisible* when they can be divided in smaller modules based on some characteristic. Loads are *arbitrarily divisible* when they can be divided in any number of parts requiring the same type of processing.

There may exist precedence relations between load fractions. In the case where no such dependencies exist, loads are said to be *independent*.

We are interested in independent arbitrarily divisible loads, which is the case for visual feature extraction. Images can be split in a number of regions and each one of them is analyzed by a different processor.

The processing of a task is completed when all of its fractions have been processed. To minimize the completion time, the goal is to obtain the optimal size of each partition of the load and the scheduling order. Completion time is minimized when all the processors finish their share of the job at the same instant. Intuitively, we can see that if one of the processors completes its task before the others, another task distribution can be found where this node will take some of the work assigned to the other processors, resulting in a faster completion time.

When we consider a network computing environment, instead of a parallel processing environment with shared memory, we have to take into account the communication delays. The solution also depends on the network topology and whether the nodes are equipped with a *front-end* or not. When equipped with a front-end, nodes can process their part of the load while simultaneously transmitting data to another node. Without a front-end, a node will first transmit the corresponding data to the other nodes before starting to process its own load. The transmission of the loads is typically done in a sequential way, where a processor only receives its share of the load after the previous processor has received its share. In some systems, the originator node simply transmits the data to the processors but does not perform any processing. In many applications, the transmission time of the results from the processors to the originator node is not considered, as it can be considered negligible.

In [22] the authors propose a *uniform multi-round* scheduling algorithm, which aims to decrease the latency when transmitting the data to the processors by transmitting smaller chunks of work in multiple rounds, so that reception and processing can overlap in the processing nodes.

In [23] the authors propose an stochastic analysis for time-varying systems where the processing and bandwidth capabilities vary due to external load in the system. [24] addresses divisible loads in real-time systems. In [25] a linear programming based approach for optimizing the finish time for real-time loads is evaluated.

2.6 Linear programming

A Linear Programming (LP) problem consists of maximizing or minimizing a linear function subject to a number of linear constraints. The function we want to maximize or minimize is called objective function. The constraints may be equalities or inequalities.

LP problems can be expressed in canonical form:

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ \text{and} & \mathbf{x} \geq \mathbf{0} \end{array}$$

where $\mathbf{x}$ is the variable vector (unknowns), $\mathbf{c}$ and $\mathbf{b}$ are vectors of known coefficients, and A is a matrix of known coefficients. Typically, there exist *nonnegativity constraints* that define a lower bound of zero for all variables $\mathbf{x}$.

When some or all of the variables $\mathbf{x}$ are restricted to be integer, we have a Integer Linear Programming (ILP) problem. When all of the variables are integer, it is a *pure* integer programming problem. When some, but not all, of the variables are integer, it is a *mixed* integer programming problem.

One of the most popular algorithms for solving linear programming problems is the *revised simplex method*, a more efficient implementation of the original *simplex method*. For integer programming problems, a popular choice is the *branch-and-bound* algorithm [26].

2.6.1 Special Ordered Sets

In the context of linear programming, a Special Ordered Set (SOS) of degree N is an ordered set of variables where at most N variables can be non-zero. The non-zero variables must be contiguous.

In a Special Ordered Set of type 1 (SOS1), only one of the variables can take a non-zero value, while the rest are zero. This can be used to model a choice from a set of mutually exclusive alternatives.

In a Special Ordered Set of type 2 (SOS2), only two variables can take non-zero value, and those variables must be contiguous. This is typically used to model piecewise-linear functions, or non-linear functions as piecewise-linear functions, which can then be solved by linear programming.

Models containing SOS variables are solved using the branch-and-bound method. When facing SOS variables, the branch-and-bound search can be performed faster, as the knowledge that a variable belongs to a set enables the solver to branch on sets or subsets (depending on the SOS order) rather than on individual variables [27].

2.6.2 Approximating non-linear functions as piecewise-linear functions

Non-linear functions can be approximated by piecewise-linear functions by joining points on the original curve with linear segments. By using more points on the curve, the approximation can be improved. Piecewise-linear functions can be modeled by means of SOS2 variables in ILP models.

If we want to approximate the dotted function in Figure 1 with four linear segments, we need five points of the function: (R_1, F_1), (R_2, F_2), (R_3, F_3),

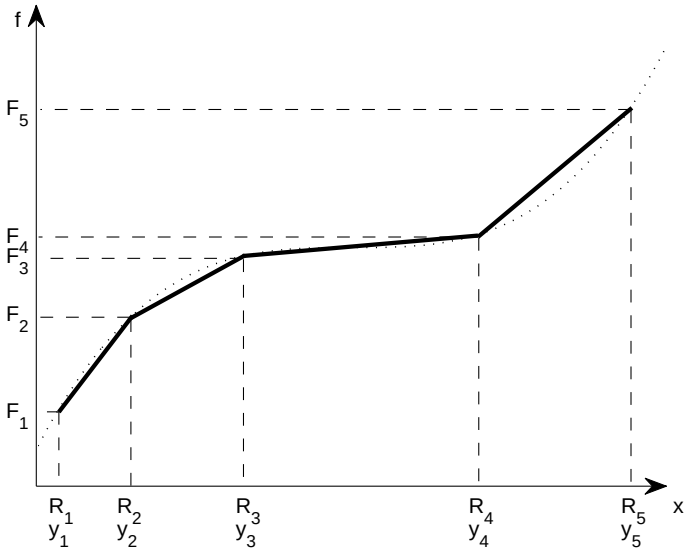

Figure 1: Piecewise-linear approximation of a non-linear function.

(R_4, F_4) and (R_5, F_5). Each point i is associated with a weight variable y_i belonging to a SOS2 set.

Then, we can model its piecewise-linear approximation, in this case with $\mathrm{K} = 5$, as:

$$\mathrm{x} = \sum_{i=1}^{\mathrm{K}} \mathrm{R}_i \cdot \mathrm{y}_i \qquad \text{(Reference row)}$$

$$\sum_{i=1}^{\mathrm{K}} \mathrm{y}_i = 1, \quad \mathrm{y}_i \geq 0 \qquad \text{(Convexity row)}$$

$$\mathrm{f} = \sum_{i=1}^{\mathrm{K}} \mathrm{F}_i \cdot \mathrm{y}_i \qquad \text{(Function row)}$$

Plus the SOS2 condition that states that only two variables y_i can take a non-zero value and they must be contiguous.

2.7 ASN.1

Abstract Syntax Notation One (ASN.1) is a standard for defining data structures and their encoding. It allows to transmit data structures over telecommunication protocols independently of their machine-specific representation. ASN.1 defines multiple encoding and decoding rules that can be used to transmit the defined data types [28], [29].

In our testbed, ASN.1 is used to define the data structures for the inter-node communications. In the following, an overview of the data types and different encoding techniques defined in ASN.1 is presented.

2.7.1 Data types

ASN.1 data types can be classified as basic types or constructed types:

- Basic types

  It includes types such as `INTEGER` (signed integer values), `REAL` (real numbers), `BOOLEAN` (two-state values), `BIT STRING` (binary data of indefinite length) and `OCTET STRING` (binary data of indefinite length multiple of eight).

- Constructed types

  Composition of basic types or other constructed types. An example of a constructed type is the following:

```
Measurement ::= SEQUENCE{
    time        Time,
    date        Date,
    temperature REAL
}

Time ::= SEQUENCE{
    second  INTEGER,
    minute  INTEGER,
    hour    INTEGER
}

Date ::= SEQUENCE{
    day    INTEGER,
    month  INTEGER,
    year   INTEGER
}
```

2.7.2 Basic Encoding Rules (BER)

Data elements are encoded as a type identifier, a length description and a value. This method is referred as *type-length-value* (TLV) encoding [30]. Sometimes, an end-of-content octet is required.

- Identifier octets

  They indicate the type of the encoded value, whether it is a primitive type or a constructed type.

- Length octets

  For definite length types, this field indicates the length of the value that follows. For indefinite length types, the length octet indicates that the following type is of indefinite length and is terminated by an end-of-content octet.

- Content octets

  They contain the data value.

- End-of-content octet

  It indicates the end of an indefinite length value.

In BER certain data types have multiple valid encodings. For instance, the boolean value *true* can be encoded as any non-zero value within an octet.

2.7.3 Distinguished Encoding Rules (DER)

DER is a subset of BER that provides exactly one way to encode a data structure. BER and DER are interoperable, meaning that a BER decoder can decode a DER stream.

2.7.4 Packed Encoding Rules (PER)

PER aims to achieve a more compact encoding than BER. Unlike BER, it does not use a *type-length-value* encoding. To further reduce the number of bits needed to encode a value, lower and upper bounds on the numerical values can be specified. The decoder needs to know the complete abstract syntax of the structure.

There exist two variants of PER: *unaligned* and *aligned*. In the unaligned variant (UPER) the data is encoded using the minimum number of bits, with no regard for byte alignment of the fields. This may require more processing time to decode. In the aligned variant, data structures are aligned on a byte level, introducing padding bits when necessary [31].

Similarly to DER, Canonical-PER provides a unique way to encode a data structure.

The syntax to constrain the value range of a data field is to specify the lower bound *lb* and/or upper bound *ub* as `(lb..ub)` after the data type. As an example, Figure 2 shows a comparison between UPER, with constrained and unconstrained values, and BER. It shows how specifying the value range of a field effectively reduces its encoded length. PER also has the benefit over BER of not needing to include data type tags.

2.8 System software

2.8.1 lp_solve

lp_solve is a Mixed Integer Linear Programming (MILP) solver distributed under the GNU Lesser General Public Licence (GNU LGPL). The solver is based on the revised simplex algorithm and the branch-and-bound method. There are a great variety of ways to use lp_solve, ranging from an IDE, a native C API and interfaces for other languages such as MATLAB, Java, Python and others. In this project, we interact with lp_solve using its C API, which can be compiled on the BeagleBone arm-hf architecture. The current version is 5.5.2.0 and can be found at `http://lpsolve.sourceforge.net/5.5/`.

Supported features include the MILP solver, Special Ordered Set (SOS) variables, integer variables and semi-continuous variables.

Encoder	Type	Value	Encoded Bytes
UPER	INTEGER(0..255)	25	1
UPER	INTEGER	25	2
BER	INTEGER	25	3
UPER	INTEGER(0..65535)	25000	2
UPER	INTEGER	25000	3
BER	INTEGER	25000	4
UPER	MySeqA	25 | 25000	3
UPER	MySeqB	25 | 25000	5
BER	MySeqB	25 | 25000	9

```
MySeqA ::= SEQUENCE {
    first  INTEGER(0..255),
    second INTEGER(0..65535)
}

MySeqB ::= SEQUENCE {
    first  INTEGER,
    second INTEGER
}
```

Figure 2: Encoding length comparison.

2.8.2 OpenCV

Open Source Computer Vision (OpenCV) is a programming library that includes functions to perform visual analysis tasks, specially aimed at real-time applications. It is distributed under the BSD licence. OpenCV is written in C/C++ and has interfaces for C/C++, Python and Java, and support for Linux, Windows, Mac OS, iOS and Android. It includes an implementation of SURF, BRISK and other algorithms.

2.8.3 ASN.1 compiler

An ASN.1 compiler reads an ASN.1 definition and generates C/C++ code that contains a native representation of the data types and provides the functions to encode and decode the data. The ASN.1 compiler we use in this project can be found at `http://lionet.info/asn1c/`.

2.9 Testbed hardware

The testbed consists of BeagleBone Black computers equipped with an IEEE 802.11 WiFi communication module or an IEEE 802.15.4 TelosB module.

2.9.1 BeagleBone Black

The BeagleBone Black is a small low-power open-source hardware computer belonging to the BeagleBoard family. The board has a microSD card reader, from which operating systems can be booted. It is capable of running Linux distributions such as Ubuntu, which we use in this testbed. In our configuration,

one can access the board using SSH through the ethernet port, a USB to ethernet adapter installed on the board or IEEE 802.11 if the adapter is connected. The board can be powered through a USB cable attached to a computer or through a 5V DC power supply.

The nodes in our VSN are BeagleBone Black boards, and wireless communications is done either by a IEEE 802.15.4 or a IEEE 802.11 stick attached to the USB port.

The dimensions of the board are 86.40 mm x 53.3 mm and its cost is around $45USD

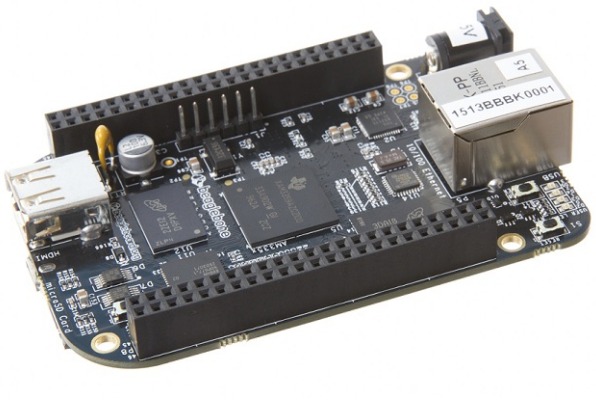

Figure 3: BeagleBone Black

Hardware specifications:

- Processor: AM335x 1GHz ARM Cortex-A8
- Memory: 512MB DDR3 RAM
- On-board flash storage: 2GB eMMC
- NEON floating-point accelerator
- 2x PRU 32-bit microcontrollers
- Power: 210-460 mA@5V

Connectivity:

- USB client for power and communications
- USB host
- Ethernet
- HDMI
- 2x 46 pin headers

2.9.2 IEEE 802.11 WiFi module

The WiFi module we use for communication between the nodes is the Netgear N150 (WNA1100). The module supports IEEE 802.11b/g/n at the 2.4 GHz frequency band and can be plugged in the USB port of the BeagleBone Black. The Linux driver available for this module supports IBSS mode (ad-hoc), which was as issue with other modules where it was not supported.

Figure 4: Netgear N150

2.9.3 IEEE 802.15.4 TelosB module

Crossbow's TelosB mote is used for 802.15.4 communications between the nodes. It can be connected to the BeagleBone's USB port.

Specifications:

- IEEE 802.15.4/ZigBee compliant RF transceiver
- Integrated onboard antenna
- Frequency band: 2.4 to 2.4835 GHz
- Data rate: 250 kbps
- RF power: -24 dBm to 0 dBm
- Receive sensibility: 90 dBm (min), -94 dBm (typ)
- Outdoor range: 75 m to 100 m
- Indoor range: 20 m to 30 m

Figure 5: TelosB

3 System design

The objective of this thesis is to implement a system that achieves to perform real-time feature extraction from video sequences captured by a camera node. The extraction of the visual features is distributed among multiple nodes of the VSN. This section describes a solution that allows to balance the workload among the nodes, minimizing the completion time of the task while maintaining good visual analysis performance, which requires the detection of a number of features close to a target value.

The camera node offloads the visual analysis task by assigning the processing of a different region of an image to each one of the cooperator nodes, referred as area-split. The camera node performs neither detection nor extraction (ND/NE). The processing load of each cooperator depends on the size of the sub-area and the number of features contained in it. The camera node is in charge of computing the optimal distribution of the workload, which includes determining the size of the sub-areas assigned to each node, the node scheduling order and predicting a detection threshold that yields a number of features close to the target value.

In Section 3.1 a description of the system operation is provided. Section 3.2 describes the optimization of the size of the sub-areas assigned to each node to minimize the completion time. In Section 3.3 the scheduling of the cooperators is discussed. In Section 3.4 the prediction of an optimal detection threshold is described. Finally, Section 3.5 describes the estimation of multiple parameters required to perform the optimization.

3.1 System description

A node can adopt three different roles: *sink*, *camera* or *cooperator*. The sink node is connected to a server and can forward requests from the server to the rest of the nodes. For example, the sink node can instruct the camera node to take a picture. The camera node captures images and is able to process them locally, send them to the cooperator nodes for distributed processing or forward them directly to the sink node. In any case, the camera node sends back to the sink the results of the requested operation. In Figure 6 the topology of the system is shown. The nodes communicate to each other through IEEE 802.11 in IBSS (ad-hoc) mode and are capable of unicast, multicast and broadcast transmissions.

After the camera has captured an image, the extraction of its features can be done in three different ways. In *Compress-Then-Analyze* (CTA) the image is compressed using JPEG and sent to the server, where the processing will be performed. In *Analyze-Then-Compress* (ATC) the camera node extracts the descriptors and sends them to the sink node. Finally, in *Distributed-Analyze-Then-Compress* (DATC) the camera node splits the image in multiple regions and assigns their processing to different cooperators. The cooperators report the results back to the camera node, and the camera node aggregates them and sends the descriptors to the sink node.

When splitting an image in multiple regions, an overlapping area along the border needs to be sent twice. This is due to the fact that the detectors require a square region around the pixels to detect an interest point. Figure 7 shows the area-split of an image in three parts and their overlapping areas. The

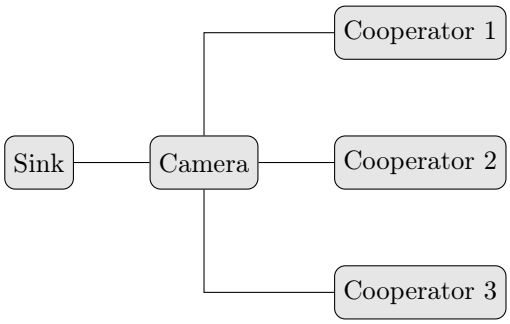

Figure 6: Topology of the links between a sink node, a camera node and three cooperators.

overlapping area can be transmitted separately by unicast to the two nodes, or by multicast.

The transmission of the sub-areas to the cooperators is done as follows. A node is idle during the time its preceding nodes receive their data, then receives the overlapping data between the previous node and itself, then receives its non-overlapping data, and finally receives the overlapping data between itself and the next node. Once a node has received all the data it needs, it can start processing. In Figure 8 the different transmission and processing phases are illustrated.

3.2 Optimal cut-point locations

This section describes the solution proposed in [20] for optimizing the size of the sub-area assigned to each node, given a particular node scheduling. The solution is then implemented in linear programming so it can be solved in real-time on the testbed.

Section 3.2.1 describes how the completion time is formulated and states the optimization problem. This formulation requires the approximation and prediction of the spatial distribution of the features, which is detailed in Section 3.2.3. In Section 3.2.4 the optimization problem is implemented in linear programming.

3.2.1 Problem formulation

We consider a VSN consisting of a camera node C, a set of N processing nodes P, and a sink node S. We consider the nodes numbered by the order in which they will receive the data and that the overlapping area spans only two nodes. We denote the average pixel transmission time from C to $P_n \in P$ as C_n. We define $C_n^M \triangleq \max(C_n, C_{n+1})$ as the pixel transmission time for multicast transmissions to nodes n and $n+1$. The multicast transmission rate to two nodes is limited by the throughput of the slowest one. Let D_j and E_j be $N \times N$ matrices and G_j a $N \times 1$ column vector. We define the normalized positions for the vertical cuts of image i as a column vector $x_i = (x_{i,1}, \ldots, x_{i,N})$, with $x_{i,N} = 1$.

The completion time for each node can be decomposed in different components. Matrices D_j and E_j and vector G_j can be formulated for each component:

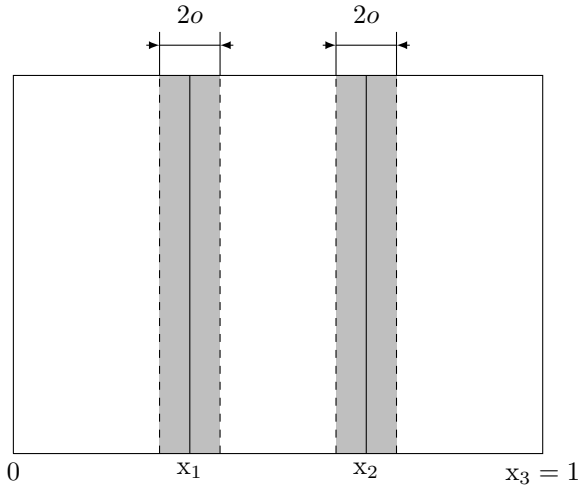

Figure 7: Area-split of an image in three parts, with normalized overlap o and cut-vector $\mathrm{x} = (\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3)$.

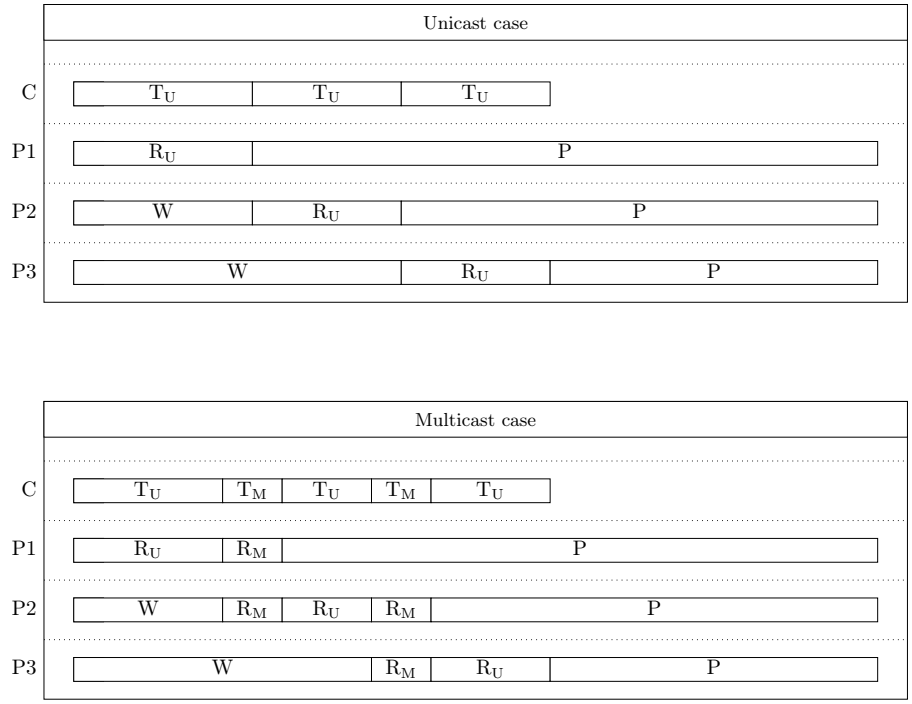

Figure 8: Transmission and processing with three cooperators. $\mathrm{T_U}$ indicates unicast transmission, $\mathrm{T_M}$ multicast transmission, $\mathrm{R_U}$ unicast reception, $\mathrm{R_M}$ multicast reception, W waiting to receive data and P processing of the image.

- Idle time:

$$T_{idle,i} = D_1 x_i + G_1$$

$$d_{1,m,n} = \begin{cases} hwC_n, & m = n+1 \\ hwC_n - hwC_{n+1}, & m > n+1 \\ 0, & otherwise \end{cases}$$

$$g_{1,n} = \begin{cases} 0, & n = 1 \\ -hwoC_1 + \sum_{j=2}^{n-1}(2hwoC_{j-1}^M - 2hwoC_j), & n > 1 \end{cases}$$

- Overlapping transmission time (multicast transmission):

$$T_{overlap,i} = G_2$$

$$g_{2,n} = \begin{cases} 2hwoC_1^M, & n = 1 \\ 2hwoC_{n-1}^M + 2hwoC_n^M, & 1 < n < N \\ 2hwoC_{N-1}^M, & n = N \end{cases}$$

- Non-overlapping transmission time (unicast transmission):

$$T_{transmit,i} = D_3 x_i + G_3$$

$$d_{3,m,n} = \begin{cases} hwC_n, & m = n \\ -hwC_{n+1}, & m = n+1 \\ 0, & otherwise \end{cases}$$

$$g_{3,n} = \begin{cases} -hwoC_n, & n \in \{1, N\} \\ -2hwoC_n, & otherwise \end{cases}$$

- Interest point detection:

  The distribution of the interest points in each region $F_i(v_i, x_i)$ is approximated by its values at Q quantiles as $\tilde{F}_i(v_i, x_i)$. The number of interest points assigned to each node results in

$$\tilde{f}_{i,n} = M^* \left(\tilde{F}_i(v_i, x_{i,n}) - \tilde{F}_i(v_i, x_{i,n-1}) \right),$$

  with M^* being the desired number of interest points to be detected.

$$T_{detect,i} = D_4 x_i + E_4 f_i(v_i, x_i)$$

$$d_{4,m,n} = \begin{cases} \frac{hw}{P_{d,px,n}}, & m = n \\ -\frac{hw}{P_{d,px,n+1}}, & m = n+1 \\ 0, & otherwise \end{cases}$$

$$e_{4,m,n} = \begin{cases} \frac{1}{P_{d,ip,n}}, & m = n \\ 0, & otherwise \end{cases}$$

  with $P_{d,px,n}$ being the rate at which interest points are analyzed as a linear function of the area, and $P_{d,ip,n}$ the rate at which the interest points are detected, as a function of the number of interest points being detected.

- Interest point extraction:

$$T_{extract,i} = E_5 f_i(\upsilon_i, x_i)$$
$$e_{5,m,n} = \begin{cases} \frac{1}{P_{e,n}}, & m = n \\ 0, & otherwise \end{cases}$$

with $P_{e,n}$ being the rate at which the descriptors for the interest points are calculated, as a linear function of the number of detected interest points.

Therefore, we can group the terms together:

$$D \triangleq D_1 + D_3 + D_4,$$
$$G \triangleq G_1 + G_2 + G_3,$$
$$E \triangleq E_4 + E_5.$$

We can calculate the approximated completion time for each node for image i as

$$\tilde{T}_i(\upsilon_i, x_i) = Dx_i + E\tilde{f}_i(\upsilon_i, x_i) + G$$

The cut-vector x_i that minimizes the completion time can be found by solving the following integer linear programming (ILP) problem:

$$\min t$$

s.t.

$$Dx_i + E\tilde{f}_i(\upsilon_i, x_i) + G \leq t\mathbf{1} \quad (1)$$
$$x_{i,n}w - x_{i,n+1}w \leq -1 \quad \forall n \quad (2)$$
$$x_{i,n}w \in \{1, \ldots, w\} \quad \forall n \quad (3)$$

The inequality in (1) is evaluated for each component. $\mathbf{1}$ is an $N \times 1$ vector of ones. Condition (2) enforces that the cut-point coordinates are increasing. Condition (3) ensures that the cuts are aligned to an integer pixel location.

We can solve a linear relaxation of the previous ILP by omitting condition (3). We also impose that overlap spans only two nodes, $2o \leq x_{i,n+1} - x_{i,n}$, so constraint (2), under the linear relaxation, can be transformed to $x_{i,n} - x_{i,n+1} \leq -2o$.

Therefore, the piecewise-linear optimization problem that needs to be solved is the following:

$$\min t \quad (4)$$

s.t.

$$Dx_i + E\tilde{f}_i(\upsilon_i, x_i) + G \leq t1 \quad (5)$$
$$x_{i,n} - x_{i,n+1} \leq -2o \quad (6)$$

3.2.2 Unicast-only formulation

This section proposes a modification to the formulation in the previous section. This modified formulation considers exclusively unicast links. In this case, each overlapping area will have to be transmitted separately to the corresponding nodes, increasing the transmission time. This will allow us to make a comparison with the multicast version. This formulation is also suitable for systems where multicast transmissions are not possible.

Vectors G_1, G_2 and G_3 are modified, while the rest remain unchanged:

- Idle time:

$$T_{idle,i} = D_1 x_i + G_1$$
$$d_{1,m,n} = \begin{cases} hwC_n, & m = n+1 \\ hwC_n - hwC_{n+1}, & m > n+1 \\ 0, & otherwise \end{cases}$$
$$g_{1,n} = \begin{cases} 0, & n = 1 \\ hwoC_1 + \sum_{j=2}^{n-1} 2hwoC_j, & n > 1 \end{cases}$$

- Multicast transmission time:

$$T_{overlap,i} = 0$$
$$g_{2,n} = 0$$

- Unicast transmission time:

$$T_{transmit,i} = D_3 x_i + G_3$$
$$d_{3,m,n} = \begin{cases} hwC_n, & m = n \\ -hwC_{n+1}, & m = n+1 \\ 0, & otherwise \end{cases}$$
$$g_{3,n} = \begin{cases} hwoC_n, & n \in \{1, N\} \\ 2hwoC_n, & otherwise \end{cases}$$

The times related to the processing task do not change respect to the previous formulation.

3.2.3 Interest point spatial distribution estimation

Because we consider the sub-areas as slices defined by vertical cuts of the original image, in order to know the number of interest points contained in each sub-area we want to find their spatial distribution along the horizontal direction. Thus $F_i(v_i, x)$ is defined as the distribution of the interest points' horizontal coordinates, from which we are able to calculate the number of interest points in each region $f_i(v_i, x)$ for the given cut-point locations x.

The distribution $F_i(v_i, x)$, however, can't be known prior to performing the feature extraction process, and can be arbitrary. Therefore, in order to include the distribution in our LP formulation for the current frame, it has to be predicted based on the distribution for the previous frames. The prediction of the

position of every single keypoint is infeasible. Therefore, we will approximate the distribution by its percentiles.

The approximation can be done by taking evenly spaced percentiles, named quantiles, or by choosing the optimal percentiles that minimize the approximation error. The second approach improves the prediction quality and reduces the number of percentiles needed to obtain the same performance, which makes solving the linear programming problem faster. However, obtaining the optimal percentiles is computationally expensive. This approach can be beneficial when dealing with large number of nodes, because the complexity of the LP problem increases with the number of nodes and the number of percentiles, however the complexity of finding the optimal percentiles does not depend of the number of nodes. Therefore, we can use a smaller number of optimal percentiles to achieve the same approximation error than using a larger number of uniformly spaced percentiles [20].

The distribution is approximated by linear interpolation between its values at Q percentiles. The prediction for the next frame is done by predicting its percentiles. In [20], different predictors are evaluated. The *last value* predictor assumes that the content of the next image is identical to the previous image. Autoregressive predictors of different orders are also studied. The last value predictor shows good results while being computationally very simple. The gain of higher order autoregressive predictors is small, and in some cases can even achieve worse results when faced with large changes of the image content. For this reason, in this implementation we will use the last value predictor.

3.2.4 Implementation in linear programming

The optimization problem described in Section 3.2.1, together with the percentile-based approximation of the distribution of the interest points has to be formulated as a linear programming problem so the optimal cut-point locations can be found by a LP solver.

The interest point distribution is implemented using SOS2 variables, as discussed in Section 2.6.2.

For a model containing N processing nodes and considering Q quantiles to approximate the keypoint distribution, the implementation is the following:

$$\text{Objective function:} \quad \min t \tag{7}$$

subject to

$$\text{N constraints} \quad \left\{ D\mathbf{x} + E\mathbf{ip} - t\mathbf{1} \leq -G \right. \tag{8}$$

$$\text{N-1 constraints} \quad \left\{ x_n - x_{n+1} \leq -2o \right. \tag{9}$$

$$\text{1 constraint} \quad \left\{ x_N = 1 \right. \tag{10}$$

$$\text{N-1 constraints} \quad \begin{cases} x_1 = q_1 d_{1,1} + q_2 d_{1,2} + \ldots + q_Q d_{1,Q} + 1 d_{1,Q+1} \\ \vdots \\ x_{N-1} = q_1 d_{N-1,1} + q_2 d_{N-1,2} + \ldots + q_Q d_{N-1,Q} + 1 d_{N-1,Q+1} \end{cases} \tag{11}$$

N-1 constraints
$$\begin{cases} d_{1,0} + d_{1,1} + \ldots + d_{1,Q} + d_{1,Q+1} = 1 \\ \vdots \\ d_{N-1,0} + d_{N-1,1} + \ldots + d_{N-1,Q} + d_{N-1,Q+1} = 1 \end{cases} \tag{12}$$

N-1 constraints
$$\begin{cases} f_1 = \frac{1}{Q} d_{1,1} + \frac{2}{Q} d_{1,2} + \ldots + \frac{Q}{Q} d_{1,Q} + 1 d_{1,Q+1} \\ \vdots \\ f_{N-1} = \frac{1}{Q} d_{N-1,1} + \frac{2}{Q} d_{N-1,2} + \ldots + \frac{Q}{Q} d_{N-1,Q} + 1 d_{N-1,Q+1} \end{cases} \tag{13}$$

N constraints
$$\begin{cases} ip_1 = f_1 \\ ip_2 = f_2 - f_1 \\ \vdots \\ ip_N = 1 - f_{N-1} \end{cases} \tag{14}$$

SOS2 sets:

$$\{d_{1,0}, \ldots, d_{1,Q+1}\}$$
$$\vdots$$
$$\{d_{N-1,0}, \ldots, d_{N-1,Q+1}\}$$

Constraints in (8) include matrix D and vectors E and G. $\mathbf{x}$ is a vector containing the N cut-point locations and **ip** is a vector containing the number of interest points in each of the N sub-regions. The variable to minimize t is included in this group of constraints.

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix},$$

with $x_N = 1$;

$$\mathbf{ip} = \begin{pmatrix} ip_1 \\ ip_2 \\ \vdots \\ ip_N \end{pmatrix}.$$

t is the objective function, representing the expected completion time, that is, the worst completion time over all the nodes. Our goal is to obtain the optimal cutvector x that achieves to minimize the completion time t.

Constraints in (9) impose the increasing condition for the cut-point locations, that is, the next cut-point pixel position must be greater than the previous one. In these constraints it is also ensured that the overlap will only span two nodes.

Constraint in (10) sets the last cut-vector element to 1, which is the position of the last pixel when normalized by the width of the image.

Constraints in (11), (12) and (13) are used to represent the piecewise linear approximation of the interest point distribution. We need to define a different SOS2 set for each node. Constraints in (11) constitute the *reference rows*, where

q_i are the values for the Q quantiles. Constraints in (12) constitute the *convexity rows*. Constraints in (13) constitute the function rows, such that f_i represent the number of interest points left of the normalized horizontal coordinate x_i.

Finally, in constraints in (14), ip_i represents the proportion (over 1) of the number of interest points located in region i.

Therefore, the model contains $6N - 3$ constraints and $3N + (N - 1)(Q + 2)$ variables, of which $(N - 1)(Q + 2)$ belong to $N - 1$ different SOS2 sets.

When the model is solved, the variables of interest are the cut-point locations $\mathbf{x} = (x_1, \ldots, x_N)$ and the expected completion time t.

The model assumes the proposed linear relaxation, therefore the resulting cut-point vector $\mathbf{x}$ has to be denormalized by multiplying by the width of the image and each element rounded to the closest integer pixel value.

3.3 Processing nodes scheduling

In Section 3.2 the optimal area cuts that minimize the completion time with a specific node ordering are found. The minimum achievable completion time, however, depends on the node scheduling. Using all the available nodes is not always optimal [10] and therefore one needs to find the optimal subset of nodes to be used and the optimal order among these nodes. In our system, where we have to transmit an overlapping area to two different nodes, we are faced with another decision. The overlapping area can be transmitted by unicast or by multicast. In [20] the authors discuss a simplified case where only two nodes are considered and the processing time is only a function of the area, and not of the number of interest points. It is shown that the existence of an overlapping area affects the optimal scheduling. When there is no overlap, it is shown that the completion time is minimized by scheduling the nodes by increasing order of bit transmission time, regardless of their processing capabilities. When there exists an overlapping area, it is shown that there exist some bit transmission times and processing rates for which the reverse scheduling, the usage of a single processor or the unicast transmission of the overlapping areas are optimal.

As the VSN considered in this thesis is homogeneous, in the sense that all the processing nodes have the same processing capabilities, we can consider a simplified case where the node scheduling can be done by decreasing link speed between the camera node and each cooperator.

A possible way to find the optimal number of cooperators that should be used is to solve the linear programming problem iteratively. One can start considering one single cooperator and solve the linear programming problem adding one more node at each step until the optimal completion time is found. This process is computationally expensive and it is not feasible to perform it for every frame.

3.4 Detection threshold

In order to achieve good image analysis results we need to compute the descriptors for a sufficient number of interest points. The objective is to obtain a target amount of interest points by selecting only the M most significant ones. This is referred as *Top-M* extraction in [15].

In a non-distributed scheme, where the entire image is analyzed by a single node, only the descriptors for the keypoints with the highest response would

be extracted. In a distributed analysis scheme, where each node of the VSN analyses a portion of the image (*area-split*), each processing node has to determine which of the detected interest points belong to the Top-M set of the entire image. In a processing intensive approach, each node would detect and extract M interest points and transmit them to the central node, where the non-Top-M interest points would be discarded. This results in all the Top-M interest points being detected, at the expense of unnecessary processing load. On the other hand, in the least processing intensive approach, if we consider N processing nodes, each node would extract the M/N descriptors with the highest response in its assigned region. In this case, the processing load is balanced across the nodes, but the complete Top-M set is not guaranteed to be found. This would be the case if the spatial distribution of the interest points is not uniform.

In [11] it is shown that for SURF and BRISK, the spatial distribution of the interest points location presents a high variability, therefore balancing the load among the processing nodes while maintaining good Top-M accuracy is not feasible without a-priori information.

The next sections describe the solution proposed in [19], which leverages the temporal correlation between successive frames of a video sequence to reconstruct the missing data and predict the optimal threshold for the next frame.

3.4.1 Threshold reconstruction

If the threshold used for frame i results in more than the desired number of interest points M^*, we can conclude that the threshold for that frame should have been higher, concretely, the score of M^*-th interest point when ordered by decreasing score.

If the threshold used for image i results in less than the desired number of interest points M^*, we should have used a lower threshold. However, in this case the optimal threshold is unknown. Our goal is to obtain an estimate of the optimal value based on the information of previous frames for which the number of obtained keypoints was greater than M^*. This allows us to estimate the slope of function $f_i(\hat{\vartheta}_i)$, defined as the number of detected interest points in image i when using threshold $\hat{\vartheta}_i$. We can define its inverse function $f_i^{-1}(m) = \max\{\vartheta | f_i(\vartheta) = m\}$, and the set of images before i for which $f_j(\hat{\vartheta}_j) \geq M^*$ as $\mathcal{I}_{i-}$.

The two proposed regression schemes in [19] are:

- **Forward Estimate:** We use the regression coefficients

$$\beta_{i-}^f = \frac{\frac{1}{|\mathcal{I}_{i-}|}\sum_{j\in\mathcal{I}_{i-}}(f_j(\hat{\vartheta}_j) - M^*)(\hat{\vartheta}_j - f_j^{-1}(M^*))}{\frac{1}{|\mathcal{I}_{i-}|}\sum_{j\in\mathcal{I}_{i-}}(f_j(\hat{\vartheta}_j) - M^*)^2}$$

  to estimate the slope of the function f_i^{-1}. Then, the estimated threshold is

$$\hat{\vartheta}_i^{f*} = \hat{\vartheta}_i - (f_i(\hat{\vartheta}_i) - M^*)\beta_{i-}^f$$

- **Backward Estimate:** We compute a regression coefficient for each difference $d = M^* - f_j(\vartheta)$, $d < M^*$:

$$\beta_{i-}^b(d) = \frac{1}{|\mathcal{I}_{i-}|}\sum_{j\in\mathcal{I}_{i-}}\frac{f_j^{-1}(M^*) - f_j^{-1}(M^* - d)}{d}$$

Then, the estimated threshold is

$$\hat{\vartheta}_i^{b*} = \hat{\vartheta}_i - (f_i(\hat{\vartheta}_i) - M^*)\beta_{i-}^b(d)$$

In the previous expressions for the regression coefficients, all the samples are weighted equally. In order to adapt quickly to changes in the statistics of the images we propose a weighted average of the regression coefficients, where the most recent samples have a higher weight than the older ones. The next coefficient can be computed recursively by updating the previous one.

For the forward estimate coefficients we define $\beta_n^f = \frac{P_n}{Q_n}$, with

$$P_n = \sum_{j \in \mathcal{I}_{n-}} (f_j(\hat{\vartheta}_j) - M^*)(\hat{\vartheta}_j - f_j^{-1}(M^*))$$

$$Q_n = \sum_{j \in \mathcal{I}_{n-}} (f_j(\hat{\vartheta}_j) - M^*)^2$$

Then, we obtain the next regression coefficient as

$$\beta_n^f = \frac{(1-\alpha_f)P_{n-1} + \alpha_f(f_j(\hat{\vartheta}_j) - M^*)(\hat{\vartheta}_j - f_j^{-1}(M^*))}{(1-\alpha_f)Q_{n-1} + \alpha_f(f_j(\hat{\vartheta}_j) - M^*)^2}$$

For the backward estimate coefficients:

$$\beta_n^b(d) = (1-\alpha_b)\beta_{n-1}^b(d) + \alpha_b \frac{f_j^{-1}(M^*) - f_j^{-1}(M^* - d)}{d}$$

Another method, referred as *scaling*, consists in scaling the threshold value by a value $0 < \alpha < 1$ whenever $f_i(\hat{\vartheta}_i) < M^*$. In this case, $\hat{\vartheta}_i^{s*} = \alpha \cdot \hat{\vartheta}_i$

3.4.2 Threshold prediction

Once we have estimated the optimal detection threshold for frame i, we want to predict the threshold for frame $i+1$. In [19], autoregressive predictors of different orders are studied, alongside the *last value* predictor, which predicts that the next threshold is the same as the optimal threshold that we should have used for the last frame.

Results show that the *last value* predictor achieves good performance while presenting very little computational complexity. At the same time, autoregressive predictors of higher order show little gains and sometimes present even worse performance.

3.5 Performance parameters estimation

The solution described in Section 3.2 needs knowledge of the transmission speed between the camera node and each node, as well as their processing speeds. Those parameters are not constant and need to be estimated in real time for every processing node.

3.5.1 Transmission speed estimation

The unicast link speed between the camera node and a processing node i, referred as C_i, can be measured by the camera node when transmitting data to the processing node if the transmission is done through a reliable protocol that acknowledges the reception of message once it is completely received. Therefore, the considered transmission time spans the transmission of the data, the reception of the acknowledgement message, as well as any possible packet retransmissions due to lost packets.

The observations are obtained by measuring the time between the start of the transmission and the reception of the acknowledgement message. In order to obtain an estimation robust to noisy variations we apply an exponential smoothing to the observations:

$$C_0 = x_0$$
$$C_t = \alpha x_{t-1} + (1 - \alpha) C_{t-1}$$

where α is the smoothing factor $0 < \alpha < 1$, C_t is the link speed estimation at instant t and x_t is the observation at instant t.

The estimated value will be used to calculate the transmission time for images, which are big messages. In the case of IEEE 802.11, the overhead added by the packet header is small in comparison to the payload data. For IEEE 802.15.4, which has a much smaller MTU, the overhead is significant. In any case, for our estimation we should not consider small messages, such as control messages, where there is very little payload data, as it would not provide a good estimation of the time needed to transfer an image.

3.5.2 Processing speed estimation

When performing feature extraction we can measure two different processing times. The first one is the time it takes to detect the keypoints. The other is the time it takes to extract their descriptors.

The time it takes to detect the interest points of a region has two components. One is a linear function of the size of the area in pixels, with rate $P_{d,px}$ pixels/second. The other is a linear function of the number of interest points in the region, detected with rate $P_{d,ip}$ interest points/second [20]. Thus, the detection time can be expressed the following way:

$$T_{detect} = N_{pix} \frac{1}{P_{d,px}} + N_{ip} \frac{1}{P_{d,ip}},$$

where N_{pix} and N_{ip} denote respectively, the number of pixels and interest points in the region.

Both $P_{d,px}$ and $P_{d,ip}$ have to be estimated from the observation of the time it took to perform the detection process, with the knowledge of how many pixels and interest points were processed. If we have multiple observations with different N_{pix} or N_{ip}, we can solve it as a least-squares parameter fitting problem. Given the variability of the image content it is reasonable to assume that N_{pix} and N_{ip} will present enough variation between different frames of a video sequence to make the least-squares fitting possible.

The resulting estimation should reflect the current load of the system by considering only the most recent samples. The use of multiple samples, apart

from allowing us to obtain the two parameters, also provides averaging. The most recent samples can be given a higher weight.

The least-squares problem is formulated as follows. $\mathbf{y}$ is a vector of the last n observations of the detection time. $\mathbf{X}$ is an n by p matrix containing, for each of the n observations, its number of pixels (N_{pix}) and number of interest points (N_{ip}). Vector $\mathbf{c}$ contains the p unknown parameters, with $p = 2$ in our case.

$$\mathbf{X} = \begin{bmatrix} N_{pix,1} & N_{ip,1} \\ N_{pix,2} & N_{ip,2} \\ \vdots & \vdots \\ N_{pix,n} & N_{ip,n} \end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix} \frac{1}{P_{d,px}} \\ \frac{1}{P_{d,ip}} \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} T_{detect,1} \\ T_{detect,2} \\ \vdots \\ T_{detect,n} \end{bmatrix}$$

The system can be expressed as

$$\mathbf{y} = \mathbf{X}\mathbf{c} + \epsilon$$

where vector ϵ is an error term.

Given $\mathbf{y}$ and $\mathbf{X}$, the objective is to estimate the value of the two parameters in vector $\mathbf{c}$ that result in the best fit. The objective function is defined as

$$S(\mathbf{c}) = \sum_{i=1}^{n} w_i (y_i - \sum_{j=1}^{p} X_{ij} c_j)^2,$$

where w_i are the elements of the weighting vector and can be used if we wish to assign a higher weight to the most recent observations. The best fit is achieved when $S(\mathbf{c})$ is minimized:

$$\hat{\mathbf{c}} = \underset{\mathbf{c}}{\operatorname{argmin}}\, S(\mathbf{c})$$

The parameter related to the extraction of descriptors, P_e, expressed as interest points/second, can be obtained simply by dividing the number of extracted descriptors by the time it took to extract them.

$$P_e = \frac{N_{ip}}{T_{extract}}$$

Exponential smoothing is applied to obtain a more stable estimation. The choice of the initial value for the exponential smoothing is important and can affect the quality of the estimation for many observations. Thus, to ensure that the measured initial value is close to the true value, the initial value is obtained by averaging the first samples, which constitutes a training period. After that, the exponential smoothing begins.

The cooperator nodes measure T_{detect} and $T_{extract}$ and report them to the camera node together with the results of the image analysis task. The camera node calculates the parameters and stores them.

4 Previous testbed

This section presents an overview of the base testbed, described in [32], where our load balancing algorithm will be implemented. The base testbed demonstrates a VSN where the nodes can act as cameras, cooperators or sinks. The roles of the nodes can change over time and their software is capable of acting as any type.

Different wireless communication modules are available. Communications between the sink node and the camera are done over IEEE 802.15.4 using TelosB modules. Communications between the camera node and the cooperators are done over IEEE 802.11 and TCP is used to provide reliability.

The testbed implements three modes of operation. In *Compress-Then-Analyze* (CTA) the camera compresses the acquired image and sends it to the sink node. In *Analyze-Then-Compress* (ATC) the camera extracts and encodes the visual features of an image and sends them to the sink node. In *Distributed-Analyze-Then-Compress* (DATC) the camera offloads the processing of the image by splitting it in multiple regions and sending each one of them to a different cooperator node for processing.

In order to minimize the completion time, the offloading algorithm described in Section 3 will be implemented on top of the current DATC implementation. The implementation of the new offloading algorithm on the testbed is detailed in Section 5.

4.1 Class structure

The central class of the system is the `NodeManager`, which coordinates the other components. The `NodeManager` creates the different classes needed according to the role assigned to the node. The `TaskManager` keeps a list of operations that the node has to execute. Each operation is represented as a `Task` object. A `Task` is an atomic operation that is performed by one of the subsystems of the `NodeManager`. The `TaskManager` reads the list of pending `Tasks` and executes them sequentially or according to a priority. When a task is completed, the `NodeManager` is notified. Complex operations are accomplished by combining multiple `Tasks`.

The `RadioSystem` contains the methods that handle the communication to other nodes. It keeps track of the existing connections, packetizes the messages and sends them. Transmissions from other nodes are also received by the `RadioSystem`. The received packets are assembled into complete messages, then decoded by `MessageParser` and delivered to the `NodeManager`, where the message will be interpreted and the corresponding tasks will be allocated. The `RadioSystem` can send and receive messages from different interfaces, such as IEEE 802.11 (WiFi) and IEEE 802.15.4 (TelosB).

The `MultimediaSystem` contains the methods related to image analysis, such as image adquisition, encoding of images, detection and extraction of keypoints and object recognition.

The `OffloadingManager` handles the offloading of processing loads to the cooperators when preforming Distributed-Analyze-Then-Compress. It keeps a list of cooperators, information about them and a pointer to their connection object. The `OffloadingManager` is in charge of deciding the size of the load

assigned to each cooperator, coordinating them and creating the corresponding messages.

4.2 Message exchanges

Figure 9 shows the message exchange between the sink node and the camera node when operating in CTA mode. The sink node sends a `StartCTAMessage` to the camera node. This message contains parameters regarding the image to be captured and how to encode it. Then the camera captures the image and sends it to the sink node in a `DataCTAMessage`.

Figure 10 shows the message exchange between the sink node and the camera node when operating in ATC mode. The sink sends a `StartATCMessage` to the camera indicating the parameters for the feature extraction task. Then the camera node captures the image and analyzes it according to the received parameters, and sends the results to the sink in a `DataATCMessage`.

Figure 11 shows the message exchange between the sink node, the camera node and two cooperators when operating in DATC mode. The sink node sends a `StartDATCMessage` to the camera node indicating the parameters for the feature extraction task, the number of cooperators to be used and the offloading algorithm that should be used. The camera node sends a `StartDATCMessage` to the cooperators indicating the feature extraction parameters. Then the camera captures the image, splits it according to the offloading algorithm and sends each part to its corresponding cooperator in a `DataCTAMessage`. The cooperators process their slice of the image and send the results to the camera node in a `DataATCMessage`. The camera node aggregates the results and sends them to the sink node in a `DataATCMessage`.

The complete ASN.1 definitions for the messages can be seen in Appendix A.

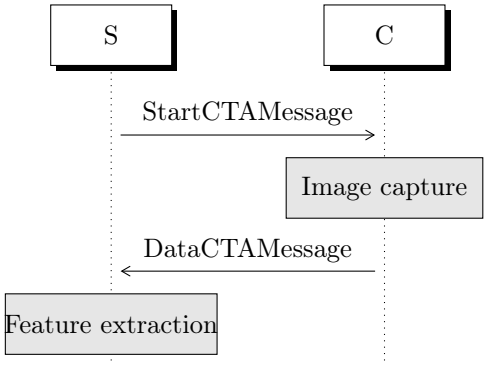

Figure 9: Message exchange in Compress Then Analyze (CTA).

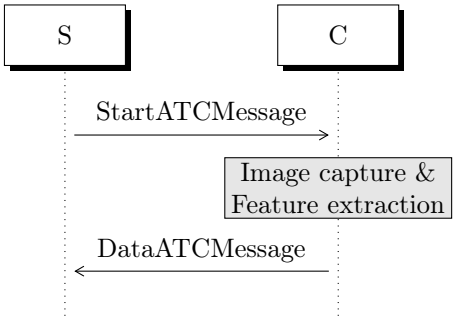

Figure 10: Message exchange in Analyze Then Compress (ATC).

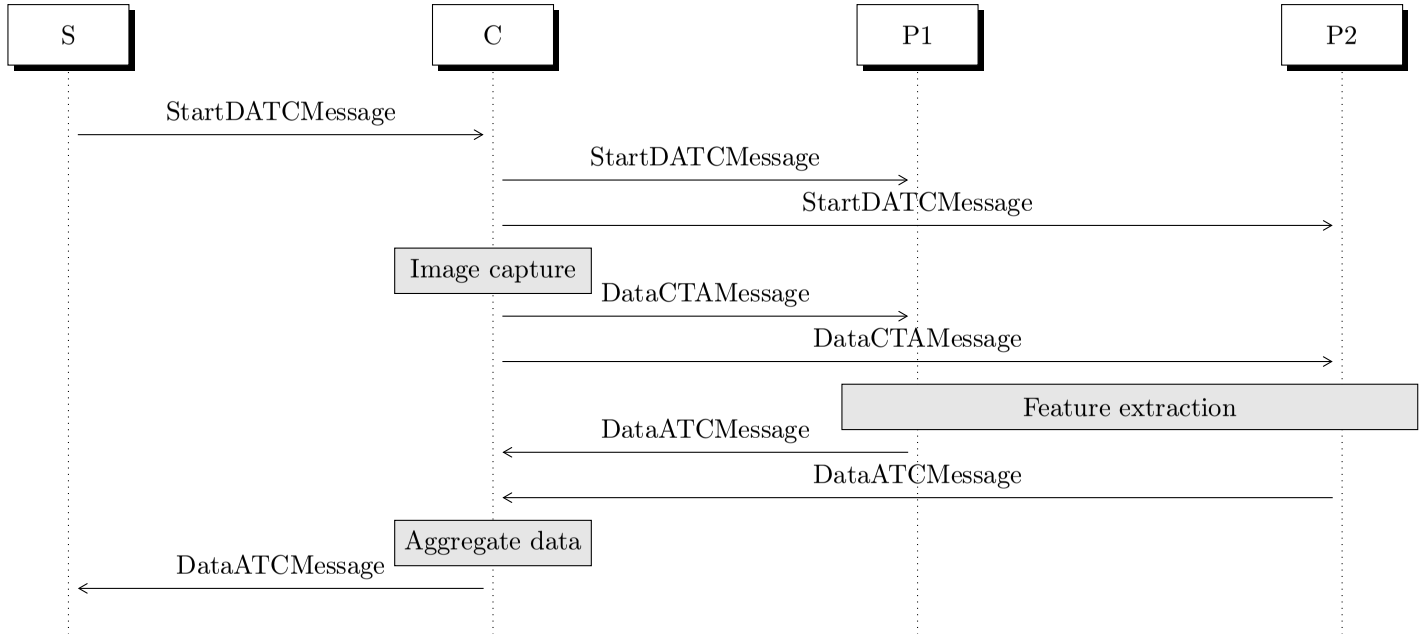

Figure 11: Message exchange in Distributed Analyze Then Compress (DATC) with two cooperators.

5 System Implementation

This section describes the implementation of the DATC offloading mechanism in terms of class structure and interaction with the rest of the testbed. In particular, it describes the classes that calculate the size of the slices assigned to each node, calculate the detection threshold to achieve a target number of keypoints, estimate the transmission speed and processing speed parameters and the storage of the required information about the cooperators.

The base testbed was described in Section 4. The implementation of the new offloading algorithm requires the modification of the `OffloadingManager` class and the development of new classes that will be used by `OffloadingManager`.

The `LoadBalancing` class is used to calculate the size of the sub-area assigned to each cooperator and to predict the detection threshold for the next frame. `LoadBalancingConfig` is used to set multiple configuration parameters for `LoadBalancing`.

`ProcessingSpeedEstimator` is used to estimate the processing speed parameters of each cooperator. `TxSpeedEstimator` estimates the transmission throughput of each cooperator. The estimated parameters will then be used by `LoadBalancing` to optimize the area distribution.

5.1 DATC offloading workflow

Figure 12 shows the workflow in the camera node when operating in DATC mode. When the camera node receives a `StartDATCMessage` from the sink node, it sends a `StartDATCMessage` to the cooperators to initialize them, takes a picture and converts it to grayscale using the `MultimediaSystem`. Then `NodeManager` creates the offloading task in `OffloadingManager` and sets the number of cooperators to use and the target number of features. `NodeManager` instructs `OffloadingManager` to sort the cooperators by decreasing transmission bandwidth and compute the load assigned to each cooperator. In turn, `OffloadingManager` instructs `LoadBalancing` to calculate the optimal cuts for splitting the image. `LoadBalancing` reads the cooperator information from a list of cooperators stored in `OffloadingManager` and solves the linear programming problem to obtain the optimal cuts. After the computation of the loads, `NodeManager` instructs the `OffloadingManager` to transmit the loads to the cooperators. When a cooperator finishes the processing of its load, it transmits the results to the camera node in a `DataATCMessage` and the keypoints are passed to `OffloadingManager`. Once the results from all the cooperators have been received, the keypoints are passed to `LoadBalancing` to update the approximation of their spatial distribution and predict the detection threshold for the next frame. `OffloadingManager` then notifies the `NodeManager` that the offloading task is completed and the process can start again for the next image.

To ensure that the image slices are transmitted sequentially to each cooperator, that is, the transmission to a cooperator does not start until the previous one has successfully received its slice, a new message called `ACKsliceMessage` is implemented. When a cooperator has received its slice, an `ACKsliceMessage` is sent to the camera node to trigger the transmission to the next cooperator. Figure 13 shows the sequential transmission of slices to the cooperators upon reception of an `ACKsliceMessage`.

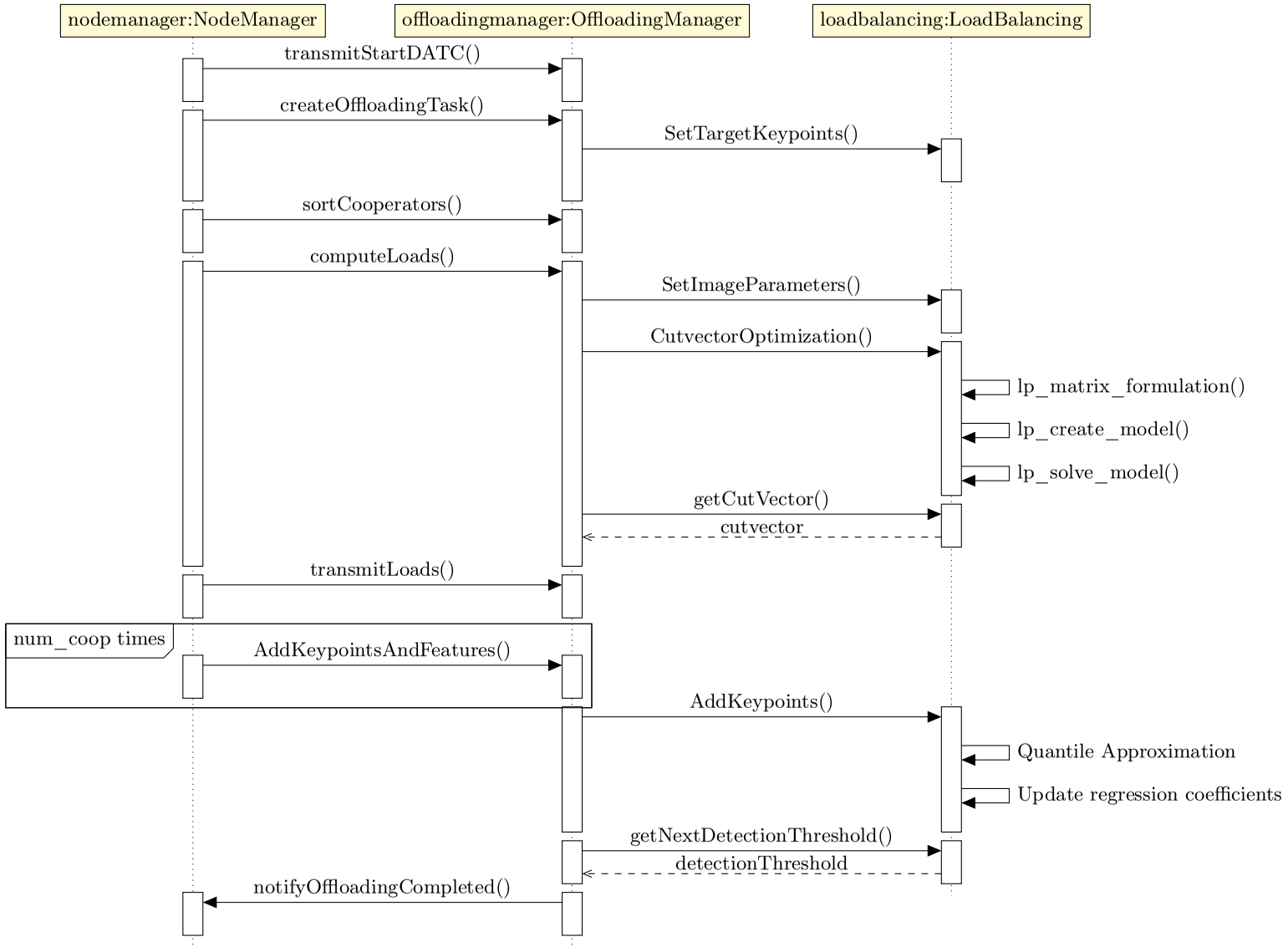

Figure 12: UML sequence diagram for the computation of the loads and detection threshold in DATC.

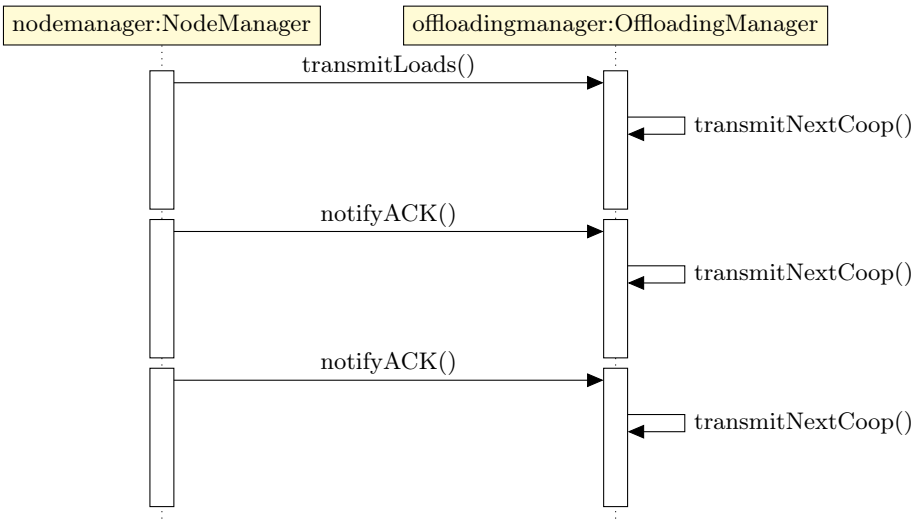

Figure 13: UML sequence diagram of the transmission of loads to the cooperators.

5.2 Class layout

The following section presents an overview of the classes related to the offloading mechanism.

5.2.1 OffloadingManager

Figure 14 shows an overview of the `OffloadingManager` class. `OffloadingManager` is used by the camera node to manage the offloading of the processing of image sub-areas to the cooperators. It keeps a list of the available cooperators with information about them and implements the methods for slicing the image and sending the slices to the cooperators. To calculate the size of each slice `OffloadingManager` has an instance of the `LoadBalancing` class. When a cooperator sends the results back to the camera node, `OffloadingManager` aggregates the received keypoints and features in a buffer. The information about the cooperators is then updated with the new data. When all cooperators have finished, the aggregated keypoints and features are delivered to `NodeManager`.

The information contained in `Cooperator` is detailed in Figure 15. It includes a pointer to its `Connection` object, the slice of the image assigned to that node and information regarding the bandwidth and processing parameters needed by `LoadBalancing`. It also contains instances of `ProcessingSpeedEstimator` and `TxSpeedEstimator` used to estimate the processing parameters and transmission throughput for that particular cooperator.

5.2.2 LoadBalancing

The `LoadBalancing` class implements the methods that compute the cut-points for splitting the image and the prediction of the optimal detection threshold as described in sections 3.2 and 3.4. Figure 16 shows an overview of `LoadBalancing`.

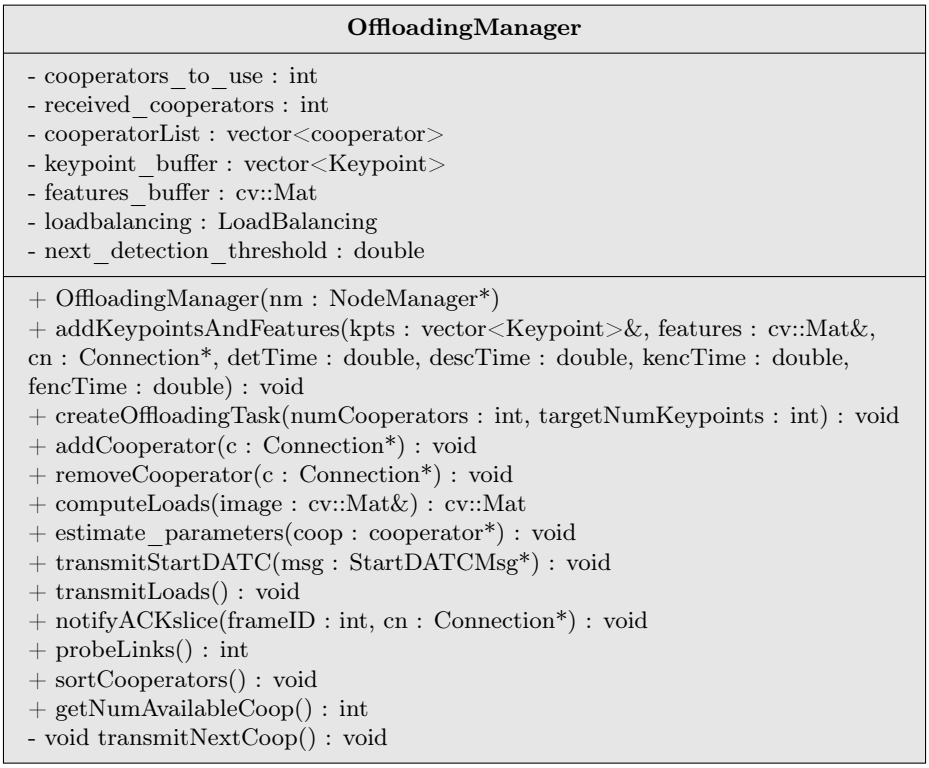

Figure 14: UML class diagram of OffloadingManager.

To formulate the linear programming problem, the information about the nodes and the keypoint spatial distribution is read from `OffloadingManager`. The keypoint spatial distributed is approximated by a number of uniformly spaced quantiles. The spatial distribution for the next frame is assumed to be identical to the previous frame. The method `lp_matrix_formulation()` generates the matrix expressions for the transmission and processing times as described in Section 3.2.1. The method `lp_create_model()` introduces the linear programming problem as described in 3.2.4, using the *lp_ solve* library. The method `lp_solve_model()` solves the linear programming problem to obtain the cut-point locations.

To predict the next detection threshold, the keypoint scores from the previous frame are read from `OffloadingManager` and used to compute the regression coefficients. The predicted detection threshold for the next freame is calculated by calling the mehtod `GetNextDetectionThreshold()`. Three different reconstruction methods are implemented: forward estimation, backward estimation and scaling. The predictor used is the *last value* predictor.

5.2.3 LoadBalancingConfig

The `LoadBalancingConfig` class contains settings for the `LoadBalancing` class. Figure 17 shows an overview of `LoadBalancingConfig`. The class also implements a method for reading a configuration from a file. The settings are the

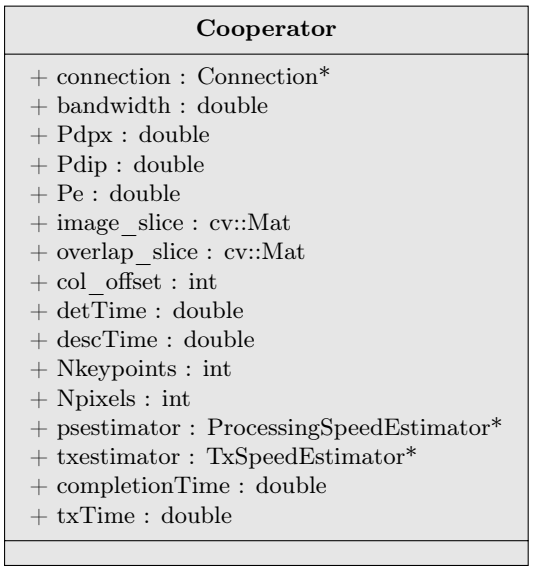

Figure 15: UML diagram of Cooperator.

following:

- reconstruction_method: {backward, forward, scaling}. Sets the detection threshold reconstruction method.
- bdr_update_coef: Exponential smoothing coefficient for the backward reconstruction regression coefficients. If 0, unweighted average.
- fdr_update_coef: Exponential smoothing coefficient for the forward reconstruction regression coefficient. If 0, unweighted average.
- scaling_coef: Threshold scaling factor for the scaling method.
- num_quantiles: Number of quantiles used to approximate the spatial distribution of the keypoints.
- solver_timeout: Timeout in seconds for the linear programming solver.
- multicast_enabled: {true, false}. Whether to consider multicast links or not.
- use_fixed_uniform_cuts: {true, false}. Assign equally sized sub-areas to each cooperator or optimize the sizes to minimize the completion time.

An example of a configuration file is the following:

```
reconstruction_method=backward
bdr_update_coef=0.05
fdr_update_coef=0.05
scaling_coef=0.9
num_quantiles=10
solver_timeout=2
```

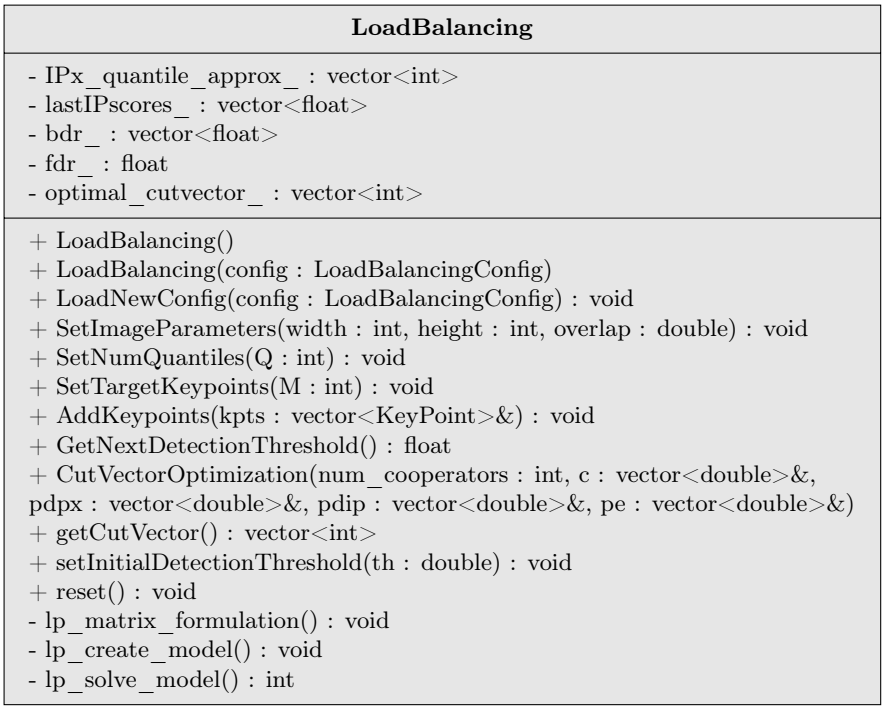

Figure 16: UML class diagram of LoadBalancing.

```
multicast_enabled=false
use_fixed_uniform_cuts=false
END
```

When `LoadBalancing` is initialized, it tries to read a configuration from a file named `loadbalancing.conf`. If the file does not exist, a default configuration is loaded.

5.2.4 ProcessingSpeedEstimator

`ProcessingSpeedEstimator` estimates the processing speed parameters as proposed in Section 3.5.2. For each `Cooperator` in the list of cooperators kept in `OffloadingManager` there exists an instance of `ProcessingSpeedEstimator`. When the camera node receives the results from a cooperator, which include the keypoints and the time it took to detect and extract them, this information is passed to `ProcessingSpeedEstimator` to calculate the parameters.

To estimate the two parameters concerning the keypoint detection rate, one as a function of the area in pixels and the other as a function of the number of keypoints detected, a least-squares multi-parameter fit is solved. To do so we employ the GNU Scientific Library (GSL).

Figure 18 shows an overview of the `ProcessingSpeedEstimator` class.

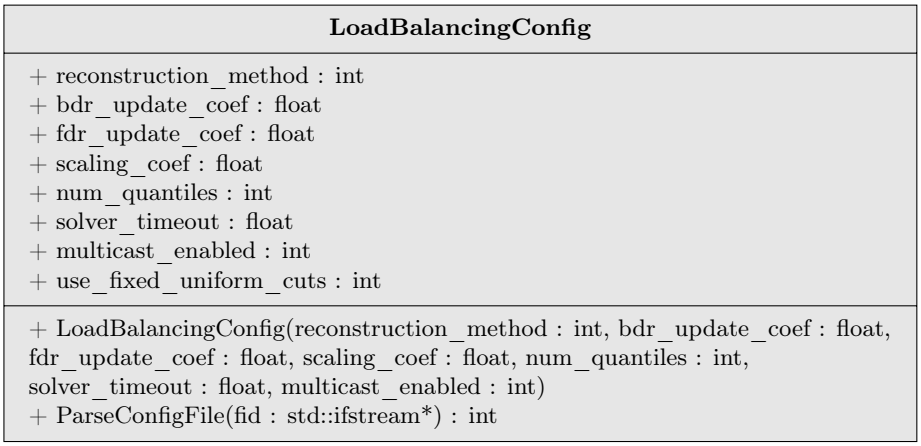

Figure 17: UML class diagram of LoadBalancingConfig.

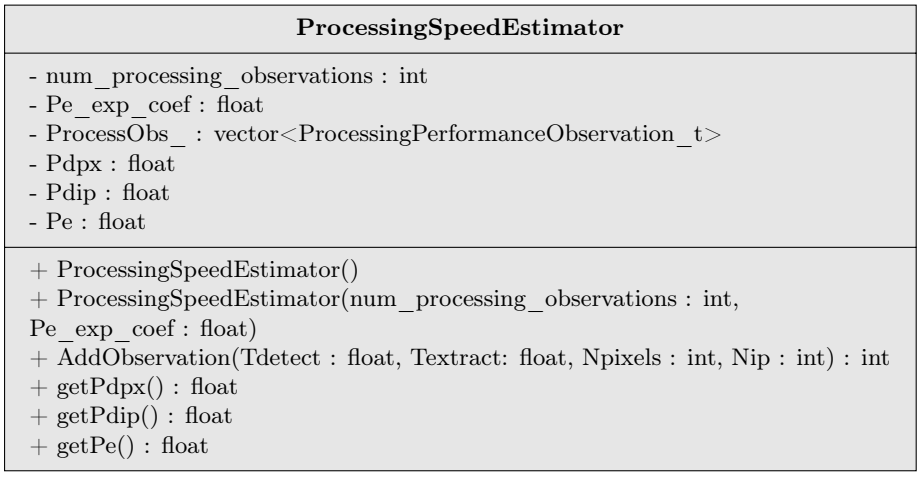

Figure 18: UML class diagram of ProcessingSpeedEstimator.

5.2.5 TxSpeedEstimator

`TxSpeedEstimator` estimates the unicast transmission speed to a cooperator node based on the time it took to transmit a slice of the image. Exponential smoothing is applied to the estimated throughput to make it robust to random fluctuations in the transmission time. The first observations constitute a training period and are averaged arithmetically to obtain the initial value for the exponential smoothing.

Figure 19 shows an overview of the `TxSpeedEstimator` class.

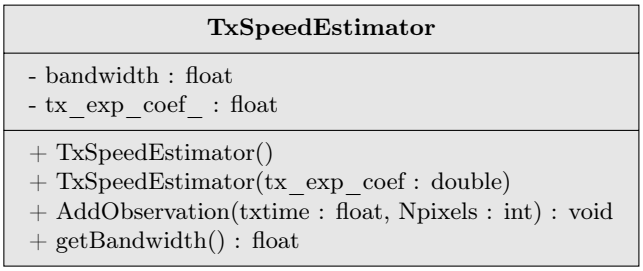

Figure 19: UML class diagram of TxSpeedEstimator.

5.3 UDP-based reliable communication module

The offloading mechanism requires multicast transmissions to transmit the overlapping area. This is not possible with the TCP-based communication system implemented in the base testbed. To make multicast transmission possible, a reliable communication protocol designed for a previous version of the testbed [33] is ported to the current testbed. The protocol was originally designed to work over IEEE 802.15.4 ZigBee. To adapt it to the current testbed, based on IEEE 802.11, the protocol is modified to work over UDP packets.

The interaction between the new communication module and the rest of the system is not altered, with the exception of the addition of a new method for transmitting multicast messages.

5.4 Slice stitching logic

When multicast transmission is used the nodes receive the images fragmented in multiple parts. The first cooperator receives a region of interest (ROI) followed by an overlap slice. The second node receives that same overlap slice followed by its ROI and another overlap slice. The last node receives an overlap slice followed by a ROI.

A cooperator needs to know how many slices need to be received before it can join them and trigger the start of the processing. A cooperator can know how many cooperators are being used by reading the field `numCooperators` of the `StartDATCMessage`. A cooperator can know its position in the scheduling order by reading the field `sliceNumber` of the received `DataCTAMessage`. Therefore, the first and the last cooperator will expect to receive two slices, while the cooperators in the middle will expect three.

6 Experimental results

This section presents an evaluation of the performance of multiple aspects of the system, including the execution times of calculations such as the linear programming optimization algorithm and the parameter estimation, the processing capabilities of the BeagleBones, the transmission bandwidth between nodes, the time it takes to process a frame and the number of detected keypoints. The tests are run on the BeagleBones with their CPU frequency limited to 300MHz, which is the lowest frequency that can be set, to better approximate to the capabilities of a sensor node. The tests are performed using a video sequence referred as "Pedestrian" trace.

6.1 Execution time of the optimization algorithm

Figure 20 shows the time it takes to solve the linear programming optimization problem that produces the optimal image cuts, as a function of the number of cooperators and the number of quantiles used to approximate the spatial distribution of the keypoints. The optimization problem has to be solved once for each image frame.

The evaluation is done on a BeagleBone with its CPU running at 300MHz. The linear programming problem is solved 100 times each number of quantiles and each number of nodes using sample data.

The results show that the execution time of the algorithm increases with the number of processing nodes, as more cuts have to be calculated. The execution time also increases with the number of quantiles used. By using more quantiles a better approximation of the spatial distribution of the keypoints is achieved, which improves the accuracy of the results of the optimization.

6.2 Execution time of the least-squares fit

To analyze the execution time of the least-squares fit algorithm, the algorithm is executed on a recorded trace of results. The results come from the analysis of the complete "pedestrian" video trace, consisting of 375 frames, processed using two cooperators. For one of the cooperators, the detection time, the extraction time, the number of keypoints detected and the number of pixels processed is recorded. The same recorded data is used for all the evaluations of the execution time of the least-squares fit, repeated for different number of considered samples. The least-squares algorithm is run on a BeagleBone with its CPU limited to 300MHz.

Figure 21 shows the time it takes to solve the least-squares fit used to obtain an estimation of the processing rate parameters, as a function of how many of the most recent samples are considered. The results show that the time required to solve the fit grows linearly with the number of samples. For each frame the fit has to be solved in the camera node once for each of the cooperators that were used.

6.3 Processing speed and transmission throughput

This section evaluates the processing capabilities of the BeagleBones and the achievable transmission throughput between nodes. The measures are repeated

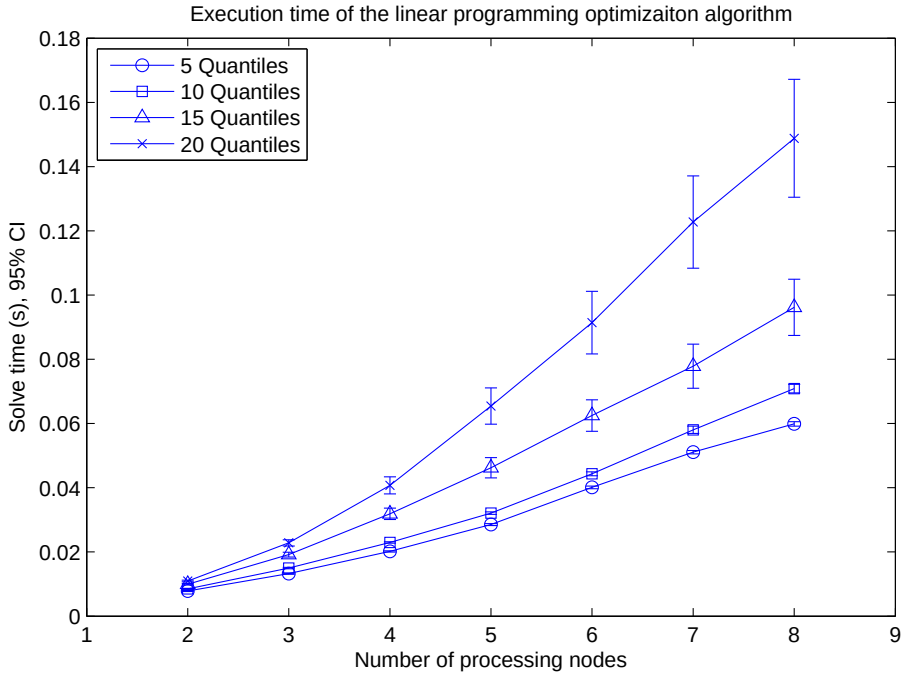

Figure 20: Execution time of the linear programming optimization algorithm measured on the BeagleBones.

for different CPU frequencies. Figure 22 presents the results. Parameters $P_{d,px}$ and $P_{d,ip}$ are related to keypoint detection process, while the parameter P_e is related to the extraction process. The transmission throughput is measured at the application layer when transmitting images to the cooperators. The unicast measure is done using the TCP communication module, while the multicast measure is done using the UDP-based reliable communication module. The multicast throughput is limited to less than 1Mbit/second because in IEEE 802.11 multicast and broadcast frames are transmitted at the lowest possible rate to assure that all types of equipment are able to receive them.

6.4 Completion time

This section seeks to evaluate the performance of the optimal image splitting algorithm. The camera node runs on a laptop computer, while the processing nodes are BeagleBones. To provide a comparison, three different allocation strategies are compared:

- *Equal split*: The image is divided in equally sized slices and each one is assigned to a different cooperator.
- *Not considering keypoint distribution*: The image cuts are calculated considering only the processing speed parameters and the throughput, but not the spatial distribution of the keypoints. Instead, the spatial distribution of the keypoints is considered uniform. The predicted best cuts within this assumption are used to divide the image.

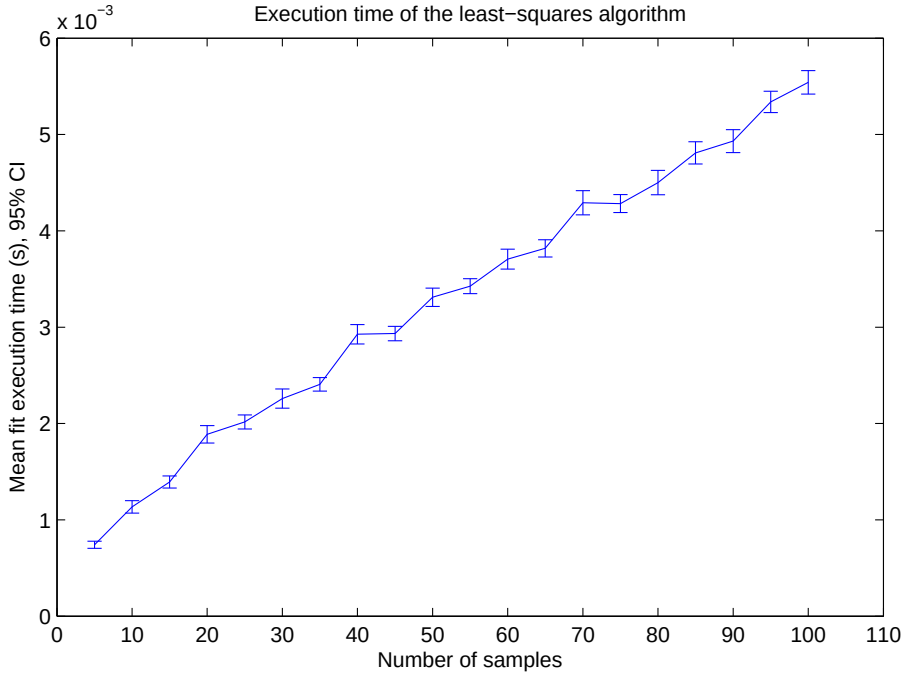

Figure 21: Execution time of the least-squares fit as a function of the number of samples.

- *Optimal split*: The image is divided using the predicted optimal cuts, considering the spatial distribution of the keypoints. This constitutes the complete solution implemented in this work.

In all cases the detection threshold is predicted such that the amount of keypoints detected is close to a target number.

The evaluation is done both for the case where only unicast transmission is possible and the case where multicast transmission is possible. The evaluation is performed by analyzing the same video sequences for the different cases. The video sequence, referred as "Pedestrian" trace, shows a street intersection with pedestrians moving horizontally across the field of view, covering and uncovering objects. This leads to important changes in the spatial distribution of the interest points. The sequence is about 15 seconds long and is analyzed taking 5 frames per second, resulting in a total of 70 frames.

6.4.1 Unicast offloading

In this case the TCP-based unicast transmission module is used. The overlapping areas are transmitted twice. The image size is 1024x768 pixels and the target number of keypoints is 500. Up to three coperators are considered.

Figure 23 shows the mean completion times for different offloading strategies and number of cooperators. The completion time is the worst finish time out of all the cooperators and includes the time to transmit the image to the cooperators, the time to process it and the time to transmit the results back to the camera node. It is shown that the optimal allocation reduces the completion

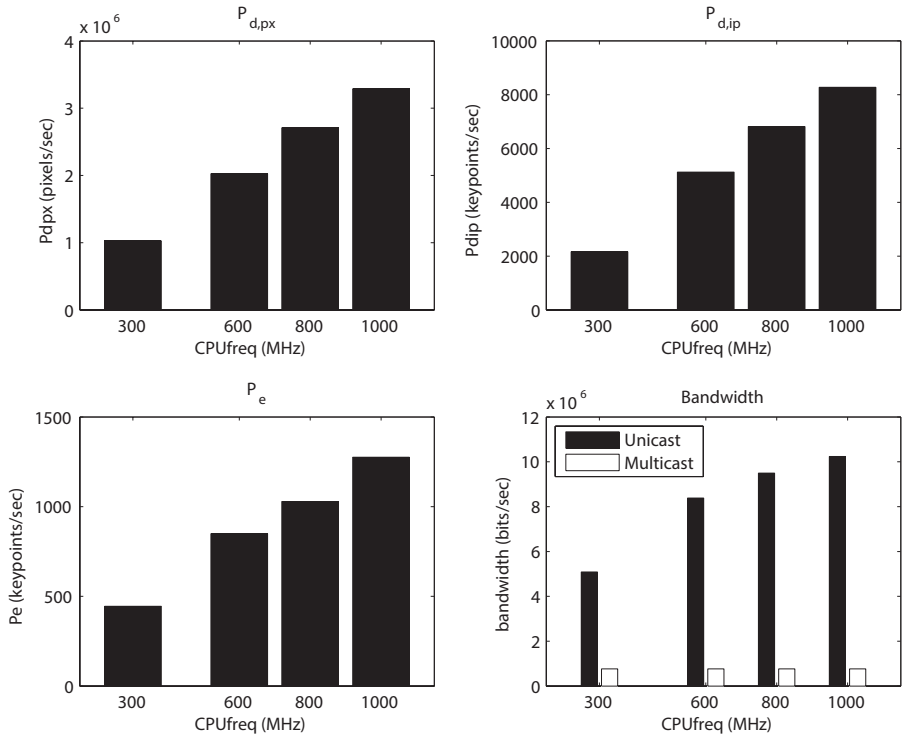

Figure 22: Processing rate parameters and transmission throughput measured on the BeagleBones for different CPU frequencies.

time respect to the two other sub-optimal allocation strategies. When three cooperators are used the equal split strategy does not reduce the completion time respect to the case where only two cooperators are used, while the other strategies slightly reduce the completion time.

Figure 24 shows the evolution of the cut locations across the frames of the video sequence for the three allocation strategies with two cooperators. The pixels below the cut are processed by the first cooperator, while those above are processed by the second. One can see how the optimal splitting strategy produces large variations of the location of the image cuts to adapt to the changes of the image content. Similarly, Figure 25 shows the same for the case where three cooperators are used.

Figure 26 shows the completion times for each frame for the three different strategies when two cooperators are used. The optimal allocation consistently achieves the smallest completion time.

6.4.2 Multicast offloading

In this case the UDP-based reliable communication module is used. The overlapping areas are only transmitted once by multicasting them. As shown in Section 6.3, the maximum multicast throughput is less than 1Mbit/second. For that reason, to evaluate this case the same throughput limit is applied for unicast transmissions. Due to the reduced transmission speed the image size is set

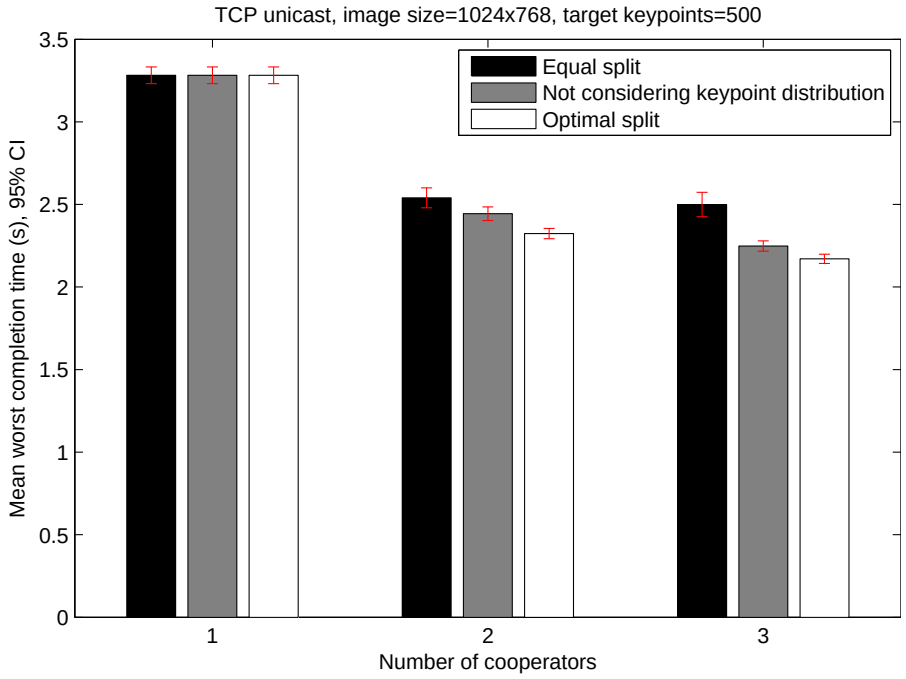

Figure 23: Time to process a frame with different number of cooperators and different splitting algorithms.

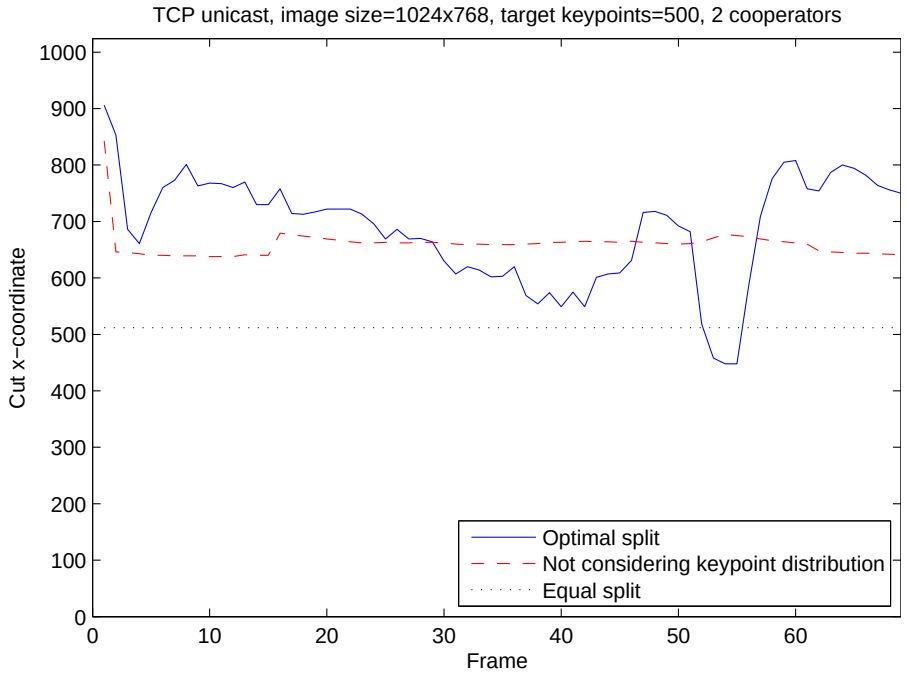

Figure 24: Evolution of the slice cut across frames for 2 cooperators and different splitting algorithms.

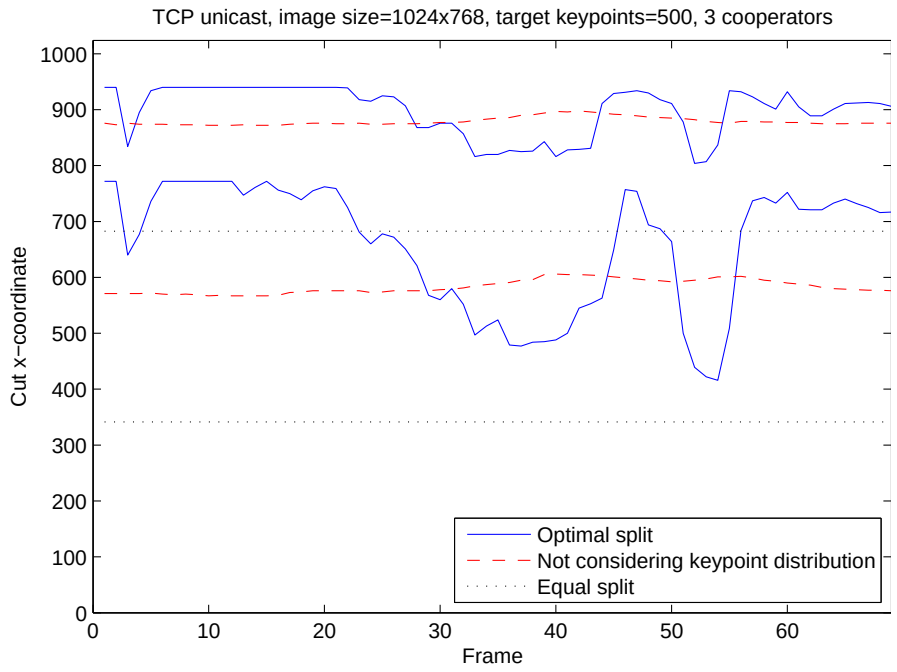

Figure 25: Evolution of the slice cuts across frames for 3 cooperators and different splitting algorithms.

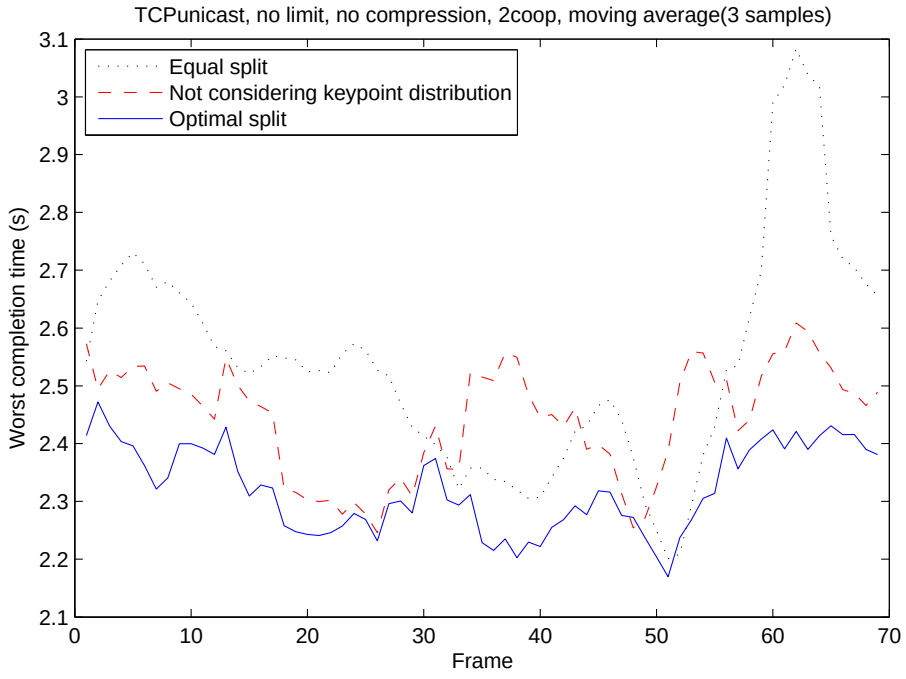

Figure 26: Worst cooperator completion time across frames for 2 cooperators and different splitting algorithms.

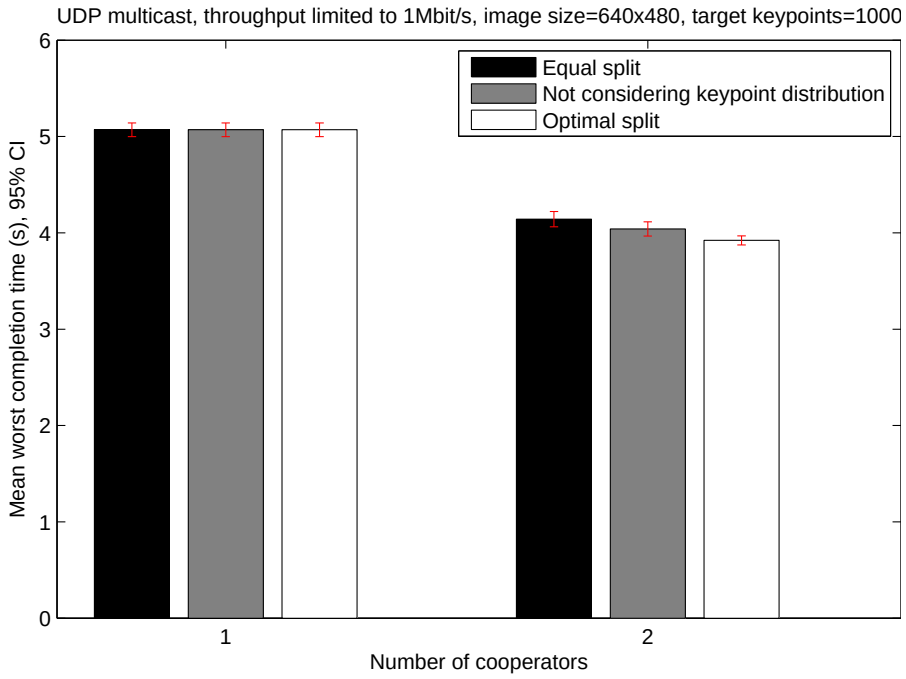

Figure 27: Time to process a frame with different number of cooperators and different splitting algorithms.

at 640x480 pixels. The target number of keypoints is 1000. Only up to two cooperators are considered, as there would be no improvement in using more cooperators due to the minimum slice size restriction with this image size.

Figure 27 shows the mean completion times for the different offloading strategies. The optimal split presents a slight reduction of the completion time, although the difference is small because most of the completion time is spent transmitting the image.

6.5 Number of interest points detected

This section analyzes the threshold prediction to produce the desired number of keypoints. The test is performed by processing all of the 375 frames of the pedestrian trace. The size of the images is 640x480 pixels and the target number of keypoints is 200. The trace is analyzed by one single cooperator. The number of cooperators does not influence the prediction of the detection threshold. The camera node records the number of keypoints detected and the predicted threshold for every frame.

Figure 28 shows the evolution of the detection threshold across the frames. The threshold adapts to the characteristics of the video sequence to achieve the target number of keypoints. Significant changes in the threshold occur when objects in the field of view are covered or uncovered. The threshold reconstruction method is the backward scheme. The other reconstruction methods produce similar results.

Figure 29 shows an histogram of the number of keypoints detected. The

mean number of detected keypoints is 201. Figure 30 shows a QQplot of the number of detected keypoints against a normal distribution, which suggests that a normal distribution is a good approximation of the number of detected keypoints.

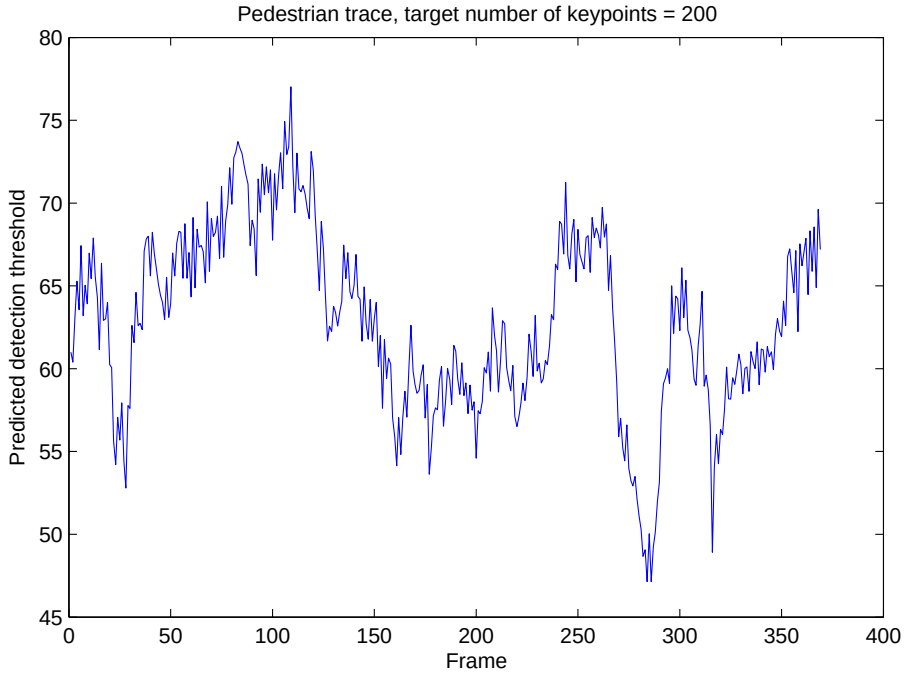

Figure 28: Predicted detection threshold for the pedestrian trace when the target number of keypoints is 50, using the backward reconstruction method.

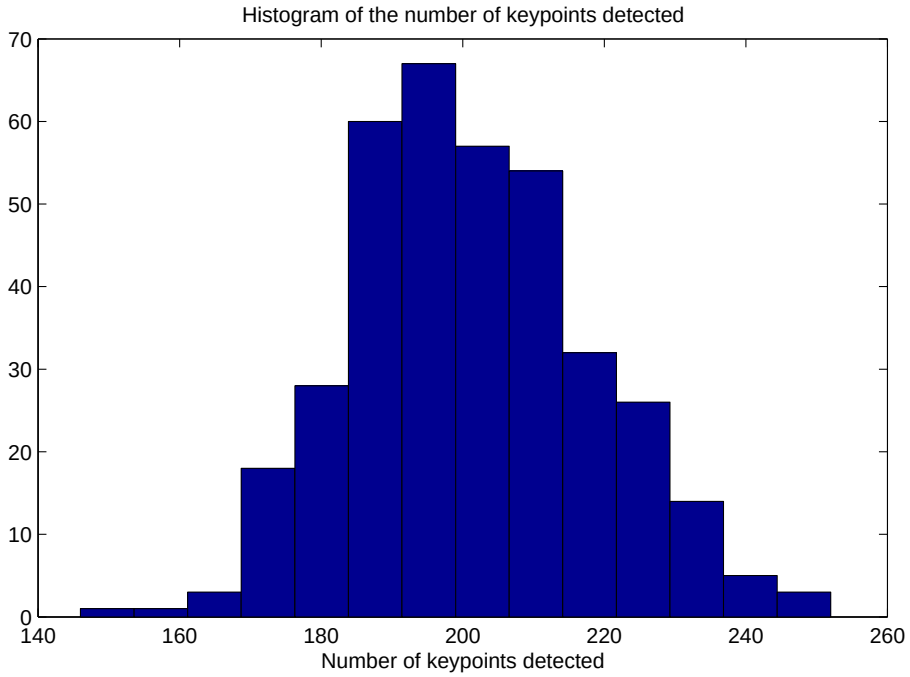

Figure 29: Histogram of the number of detected keypoints for the pedestrian trace when the target number of keypoints is 200, using the backward reconstruction method.

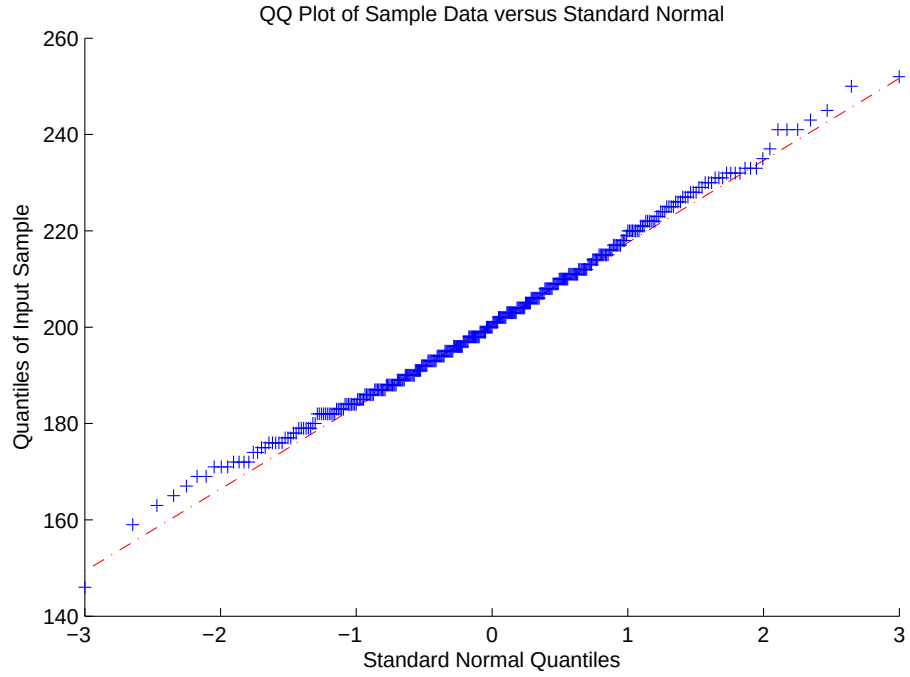

Figure 30: QQ plot of the number of detected keypoints against a normal distribution.

7 Conclusion and future work

7.1 Conclusion

In this thesis an algorithm to optimally distribute image processing tasks among nodes of a visual sensor network has been implemented on a real testbed system. The goal is to leverage the processing power of other nodes of the network to minimize the time required to process an image, which allows for real-time processing of video in low-powered nodes. Distributing the processing tasks also achieves to balance the power consumption of the nodes, which is important in energy constrained arrangements. The images captured by a camera node are split in multiple slices and each one of them is sent to a different cooperator node for processing. The cooperators extract visual features from the images and return their descriptors to the camera node.

The implemented algorithm seeks to optimize the size of the images in order to minimize the global completion time, defined as the worst completion time out of all the cooperators. The optimal distribution is that where all the cooperators finish their task at the same time. The optimization is modeled as a linear programming problem that takes into account the time required to transmit the images to the nodes, the time it takes to detect the interest points and the time it takes to extract their descriptors. These times are dependent on the communication channel conditions, the processing capabilities of the nodes and the image content. All of them can vary along the time. The system is able to estimate them and adapt to the changes in real-time.

The solution is evaluated on the testbed. The results show the importance of an optimal allocation of the tasks. The optimal allocation effectively reduces the completion time compared to simpler sub-optimal allocation methods.

7.2 Future work

In the current implementation the images are not compressed before sending them to the cooperator nodes. As a result, a significant part of the completion time is spent transmitting the image slices instead of processing. This limits the achievable improvement of using more cooperators. To improve the performance of the system, image compression could be considered. The compression scheme should be optimized for feature extraction, like the one proposed in [12]. If compression is used, the completion time optimization algorithm implemented in this thesis should be modified to account for the compression. The compression ratio would be dependent on the image content. One could measure the compression ratio for blocks along the horizontal direction of the image. This information would be used to model the transmission time in the optimization algorithm.

The offloading algorithm could also be extended to optimally schedule to process simultaneously the information from multiple camera nodes.

Further development of the testbed could also include routing capabilities. For instance, currently the camera needs direct communication to the cooperators. The bandwidth and energy costs of multihop routing should be taken into account when offloading tasks to cooperators.

References

[1] S. Soro and W. Heinzelman. A survey of visual sensor networks. *Advances in Multimedia*, 2009.

[2] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding (CVIU)*, 110(3):346–349, 2008.

[4] Edward Rosten, R. Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(1):105–119, Jan 2010.

[5] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Computer Vision–ECCV 2010*, pages 778–792. Springer, 2010.

[6] S. Leutenegger, M. Chli, and R.Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555, Nov 2011.

[7] O. Miksik and K. Mikolajczyk. Evaluation of local detectors and descriptors for fast feature matching. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 2681–2684, Nov 2012.

[8] A. Canclini, M. Cesana, A. Redondi, M. Tagliasacchi, J. Ascenso, and R. Cilla. Evaluation of low-complexity visual feature detectors and descriptors. In *Digital Signal Processing (DSP), 2013 18th International Conference on*, pages 1–7, July 2014.

[9] A. Redondi, L. Baroffio, M. Cesana A, Canclini, and M. Tagliasacchi. Briskola: Brisk optimized for low power arm architectures. In *IEEE International Conference on Image Processing 2014*, 2014.

[10] V. Bharadwaj, D. Ghose, and T. G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–17, 2003.

[11] M. A. Khan, G. Dán, and V. Fodor. Characterization of SURF and BRISK interest point distribution for visual processing in sensor networks. In *Proc. of 18th Intl. Conf. on Digital Signal Processing, Jul 2013.*

[12] Jianshu Chao, Hu Chen, and E. Steinbach. On the design of a novel jpeg quantization table for improved feature detection performance. In *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pages 1675–1679, Sept 2013.

[13] J.J. Ahmad, H.A. Khan, and S.A. Khayam. Energy efficient video compression for wireless sensor networks. In *Information Sciences and Systems, 2009. CISS 2009. 43rd Annual Conference on*, pages 629–634, March 2009.

[14] A. Redondi, L. Baroffio, J. Ascenso, M. Cesano, and M. Tagliasacchi. Rate-accuracy optimization of binary descriptors. In *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pages 2910–2914, Sept 2013.

[15] A. Redondi, M. Cesana, and M. Tagliasacchi. Rate-accuracy optimization in visual wireless sensor networks. In *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pages 1105–1108, Sept 2012.

[16] L. Baroffio, M. Cesana, A. Redondi, M. Tagliasacchi, and S. Tubaro. Coding visual features extracted from video sequences. *Image Processing, IEEE Transactions on*, 23(5):2262–2276, May 2014.

[17] V. Sulic, J. Pers, M. Kristan, and S. Kovacic. Efficient feature distribution for object matching in visual-sensor networks. *Circuits and Systems for Video Technology, IEEE Transactions on*, 21(7):903–916, July 2011.

[18] Hailong Li, V. Pandit, and D.P. Agrawal. Gaussian distributed deployment of relay nodes for wireless visual sensor networks. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 5374–5379, Dec 2012.

[19] E. Eriksson, G. Dán, and V. Fodor. Prediction-based load control and balancing for feature extraction in visual sensor networks. In *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014.

[20] E. Eriksson, G. Dán, and V. Fodor. Real-time distributed visual feature extraction from video in sensor networks. In *Proc. of IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), May 2014*.

[21] A Shokripour and M. Othman. Survey on divisible load theory. In *Computer Science and Information Technology - Spring Conference, 2009. IACSITSC '09. International Association of*, pages 9–13, April 2009.

[22] Y. Yang and H. Casanova. Umr: a multi-round algorithm for scheduling divisible workloads. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 9 pp.–, April 2003.

[23] J. Sohn and T.G. Robertazzi. Optimal time-varying load sharing for divisible loads. *Aerospace and Electronic Systems, IEEE Transactions on*, 34(3):907–923, Jul 1998.

[24] Suriayati Chuprat, S. Salleh, and S. Goddard. Real-time divisible load theory: A perspective. In *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on*, pages 6–10, Sept 2009.

[25] Suriayati Chuprat, S. Salleh, and S.K. Baruah. Evaluation of a linear programming approach towards scheduling divisible real-time loads. In *Information Technology, 2008. ITSim 2008. International Symposium on*, volume 1, pages 1–8, Aug 2008.

[26] Bradley, Hax, and Magnanti. *Applied Mathematical Programming*. Addison-Wesley, 1977.

[27] C. Guéret, C. Prins, and M. Sevaux. *Applications of optimization with Xpress-MP*. Editions Eyrolles, 2000.

[28] International Telecommunications Union (ITU). Introduction to ASN.1. Online, 2014.

[29] O. Dubuisson. *ASN.1, Communication between Heterogeneous Systems*. OSS Nokalva, 2000.

[30] Telecommunication standarization sector of ITU (ITU-T). *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*, 2002.

[31] Telecommunication standarization sector of ITU (ITU-T). *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*, 2002.

[32] L Baroffio, A Canclini, M Cesana, A Redondi, M Tagliasacchi, G Dán, E Eriksson, V Fodor, J Ascenso, and P Monteiro. Enabling visual analysis in wireless sensor networks.

[33] A. Aurensanz. A reliable transmission protocol for distributed extraction of visual features using brisk and surf. Master's thesis, KTH, School of Electrical Engineering, 2014.

Appendices

A ASN.1 definitions of the messages

```
VSNTestBed DEFINITIONS AUTOMATIC TAGS ::= BEGIN

CooperatorInfo ::= SEQUENCE {
  coopId    INTEGER,
  ipAddress OCTET STRING,
  port      INTEGER,
  status    CoopStatus
}

DataATCMessage ::= SEQUENCE {
  frameID      INTEGER,      -- Identifies which frame the features belong to.
  blockNumber  INTEGER,      -- Features are grouped into blocks.
  numBlocks    INTEGER,      -- Tot. number of blocks
  detTime      REAL,         -- Time spent for keypoint detection
  descTime     REAL,         -- Time spent for features description
  kencTime     REAL,         -- Time spent for encoding keypoints
  fencTime     REAL,         -- Time spent for encoding features
  featuresData OCTET STRING, -- Descriptor data.
  kptsData     OCTET STRING  -- Keypoints data
}

DataCTAMessage ::= SEQUENCE {
  frameID      INTEGER,     -- Identifies which frame the data belongs to.
  sliceNumber  INTEGER,     -- An image would be divided in several slices
                            -- each of which can be decoded into useful
                            -- sub-area of the complete image.
  topLeft      Coordinate,  -- Describes where the slice is positioned in the
                            -- original image.
  dataSize     INTEGER,     -- The length of the data bit string.
  encTime      REAL,        -- time spent for JPEG encoding
  data         OCTET STRING -- JPEG encoded image data.
}

--Sent from server to camera. Specifies the details of, and initiates CTA
--mode operation
StartCTAMessage ::= SEQUENCE {
  framesPerSecond  INTEGER,  -- Valid options are: -1=as high as
                             -- possible, 0=one-shot, <=1=specific rate.
  qualityFactor    INTEGER,  -- JPEG quality factor to be used.
  frameHeight      INTEGER,  -- Height of the frame to be captured.
  frameWidth       INTEGER,  -- Width of the frame to be captured.
  numSlices        INTEGER
}

--Sent from server to camera. Specifies the details of, and initiates ATC
```

```
--mode operation
StartATCMessage ::= SEQUENCE {
  framesPerSecond      INTEGER,          -- Valid options are: -1=as high
                                         -- as possible, 0=one-shot,
                                         -- <=1=specific rate.
  detectorType         DetectorTypes,    -- Type of detector to be used.
  detectorThreshold    REAL,             -- Detection threshold.
  descriptorType       DescriptorTypes,  -- Type of descriptor to be used.
  descriptorLength     INTEGER,          -- Length of each descriptor.
  maxNumberOfFeatures  INTEGER,          -- Maximum number of features to
                                         -- transfer back to the server.
  rotationInvariant    BOOLEAN,          -- Use rotation invariant
                                         -- descriptors.
  coding               CodingChoices,    -- Type of coding to be performed on
                                         -- the descriptors.
  transferCoordinates  BOOLEAN,          -- Should coordinates be transferred?
  transferScale        BOOLEAN,          -- Should scale be transferred?
  transferOrientation  BOOLEAN,          -- Should orientation be transferred?
  numFeaturesPerBlock  INTEGER
}

StartDATCMessage ::= SEQUENCE {
  framesPerSecond      INTEGER,          -- Valid options are: -1=as high
                                         -- as possible, 0=one-shot,
                                         -- <=1=specific rate.
  detectorType         DetectorTypes,    -- Type of detector to be used.
  detectorThreshold    REAL,             -- Detection threshold.
  descriptorType       DescriptorTypes,  -- Type of descriptor to be used.
  descriptorLength     INTEGER,          -- Length of each descriptor.
  maxNumberOfFeatures  INTEGER,          -- Maximum number of features to
                                         -- transfer back to the server.
  rotationInvariant    BOOLEAN,          -- Use rotation invariant
                                         -- descriptors.
  coding               CodingChoices,    -- Type of coding to be performed on
                                         -- the descriptors.
  transferCoordinates  BOOLEAN,          -- Should coordinates be transfered?
  transferScale        BOOLEAN,          -- Should scale be transfered?
  transferOrientation  BOOLEAN,          -- Should orientation be transfered?
  numFeaturesPerBlock  INTEGER,
  numCooperators       INTEGER,          -- 0 auto
  offloading           OffloadingChoices
}

ACKsliceMessage ::= SEQUENCE {
  frameID INTEGER
}

CoopStatus ::= ENUMERATED {
  online,
  offline
```

```
}

CodingChoices ::= ENUMERATED {
  none,
  entropyCoding
}

OffloadingChoices ::= ENUMERATED {
  polimi,
  kth
}

Coordinate ::= SEQUENCE {
  xCoordinate  INTEGER,
  yCoordinate  INTEGER
}

--Whichever descriptors we support.
DescriptorTypes ::= ENUMERATED {
  sift,
  surf,
  brief,
  brisk,
  orb,
  freak
}

--Whichever detectors we support.
DetectorTypes ::= ENUMERATED {
  fast,
  star,
  sift,
  surf,
  orb,
  brisk,
  mser
}

END
```