EZ-VSN: an Open-Source and Flexible Framework for Visual Sensor Networks

Luca Bondi, Luca Baroffio, Matteo Cesana, Alessandro Redondi, Marco Tagliasacchi

***Abstract*—We present a complete, open-source framework for rapid experimentation of Visual Sensor Networks (VSN) solutions. From the software point of view, we base our architecture on open-source and widely known C++ libraries to provide the basic image processing and networking primitives. The resulting system can be leveraged to create different types of VSNs, characterized by the presence of multiple cameras, relays and cooperator nodes, and can be run on any Linux-based hardware platform, such as the BeagleBone Black. To demonstrate the flexibility of the proposed framework, we describe two different application scenarios typical of VSNs, namely object recognition and parking monitoring. The framework is then used to evaluate the benefits of two complementary paradigms for networked visual analysis recently discussed in the literature. In the traditional Compress-then-Analyze (CTA) paradigm, compressed images are transmitted from camera nodes to a central controller, where they are analyzed. In the novel Analyze-then-Compress (ATC) paradigm, camera nodes extracts and compress local features from the acquired images. Such features are transmitted to the central controller and used to perform visual analysis. We show that the ATC paradigm outperforms CTA from the consumed energy point of view, at the same target analysis accuracy in both the application scenarios.**

***Index Terms*—Visual Sensor Networks, Testbed, Features Extraction, Object Recognition, Parking Lot Monitoring**

I. Introduction

In the last few years, Visual Sensor Networks (VSNs) have gained a lot of attention within both the scientific community and the industry. Composed of many low-cost, battery-operated intelligent wireless cameras, VSNs are envisioned to play a major role in the evolution of the Internet-of-Things paradigm, supporting a vast range of applications such as object detection, recognition and tracking, video surveillance, assisted living, and many others. VSNs are particularly challenging from the research point of view. On the one hand they are used for multimedia applications, which are typically characterized by intense processing, high bandwidth usage and considerable energy consumption. On the other hand, being related to the more general Wireless Sensor Networks, VSNs have tight constraints in terms of computational, communication and energy resources. Thus, a huge part of the research on VSNs has been committed to finding efficient software solutions at every layer of the protocol stack, ranging from low-level communication protocols [1], [2], to multimedia compression schemes at the application layer [3], [4], [5].

The majority of such research studies consider a traditional scenario in which visual sensor nodes acquire images, compress them relying on some sort of encoding algorithm (e.g., JPEG) and transmit the compressed bitstream to a remote *sink* node, where the images are decoded. Often, some sort of specialized analysis algorithm is applied to the received images with the purpose of detecting, recognizing and tracking objects of interests. Therefore, such a scenario is often referred to as *Compress-then-Analyze* (CTA). Very recently, a paradigm shift has been proposed: in the *Analyze-then-Compress* (ATC) scenario, camera nodes analyze the acquired image content and extract and transmit a set of visual features from it. In a nutshell, a *detector* algorithm is run to identify salient keypoints in the image. For each keypoint, a local *feature* is produced, summarizing the patch surrounding the keypoint. Since such features can be efficiently compressed and used for automatic analysis, they constitute an excellent candidate for enabling advanced visual tasks in energy- and bandwidth-limited scenarios [6], [7], [8], [9].

Clearly, a detailed assessment of the benefits provided by any VSN-specific solution, including the improvements that the ATC paradigm is able to provide with respect to CTA, cannot be done without a reference target platform. In fact, since the very first research studies in the field of VSNs, the importance of discovering a reliable VSN hardware platform was highlighted [10]. While reliable *motes* for generic Wireless Sensor Networks have been available for a decade, the same is not true for VSN. Although several efforts to propose VSN platforms have been done in the last few years, many research studies at different layers of the protocol stack for VSNs have been validated only through simulations. This fact clearly represents an issue, given that critical VSN performance parameters, power consumption first, are governed by the hardware. Moreover, the lack of a reliable, easily accessible and low-cost VSN platform makes it difficult to compare the different solutions at the application, transport and network layer that have been proposed so far.

In this paper we propose a novel open-source platform tailored to easy, flexible and quick experimentation of software solutions for VSNs. On the software level, we propose a flexible and efficient architecture based on widely accepted, open-source standards which provides all the necessary processing and transmission primitives needed for implementing a wide class of applications. On the hardware level, the framework

The authors are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy, email: name.surname@polimi.it

The project GreenEyes acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number: 296676.

can be run on any Linux-based system: in particular, we test our design on top of a low-cost and low-power BeagleBone Black embedded computer.

The rest of the paper is structured as follows. In Section II, we briefly review the state of the art in VSN platforms design, focusing on the main requirements that a good platform should have and the different approaches to VSN platforms design. In Section III, we present and thoroughly detail the proposed system at the software level, starting from a general outlook and then focusing on the single building blocks and how they can be interconnected in a flexible way to obtain different VSN setups. In Section IV we focus on the hardware used to build the VSN platform, giving particular attention to its energy consumption and comparing it with another similar commercially available hardware platform capable of supporting the proposed software system. Section V presents representative application scenarios that can be implemented with the proposed framework. We give particular attention to the CTA and ATC paradigms for object recognition and parking monitoring, and evaluate their performance in terms of achievable frame rate and consumed energy. Finally, Section VI concludes the paper.

II. Related Work

Giving a complete review of all the proposed VSN platforms is out of the scope of this paper. The interested reader may refer to the excellent survey papers from Abas et al. [11] or Seema and Reisslein [12]. In particular, the latter identifies the requirements for a good VSN platform and thoroughly describe all the solutions proposed so far. The authors classify existing VSN platforms in three main classes:

- *General Purpose Architectures* (e.g., [13], [14], [15]), are designed similarly to a personal computer, attempting to cover all possible functionalities and peripherals an application may need. Such strategy allows to create application prototypes very quickly, but typically suffers from high power consumption, due to the overuse of standard interfaces (e.g., USB, UART, GPIO).
- *Heavily Coupled Architectures* (e.g., [16], [17]) are typically designed for a particular application, and they can be optimized with respect to that. On the down side, they are over-customized and lack flexibility, thus not being well suited for research purposes.
- *Externally Dependent Architectures* (e.g., [18], [19], [20]) are designed in a layered way, where external daughter boards can be attached to the main board to achieve basic functionalities. On the one hand, such a design allows to easily customize the platform based on the application requirements. On the other hand, a given main board can usually interoperate only with daughter boards designed for it, thus limiting flexibility. Also, such a design often results in high power consumption, since basic circuitry is usually duplicated on both the main board and the daughter boards.

Besides classifying the available VSN platforms, the work in [12] provides a list of reasonable practical requirements for a good VSN node, namely (i) low power consumption (0.5W preferable - 2W maximum) and possibility of supporting standby or sleep power modes (ii) high video throughput (up to 15 frame per seconds at CIF resolution) and (iii) low-cost (less than \$50 USD). Also, the authors point out that it should be possible to substitute key components (i.e., camera sensor, radio module) based on competitive pricing/performance in a modular manner. Finally, the ability to use extremely low-power sensors (e.g., infrared, motion sensor) to trigger frame acquisition further reducing power consumption, is considered as a plus. The survey concludes that none of the existing VSN designs meet the ideal requirements, as they all have the shortcoming of either attempting to incorporate too many components, resulting in inefficient, general-purpose architectures, or attempting to be too application-specific, resulting in architectures with very limited flexibility.

Besides such practical requirements, which are certainly critical when it comes to deploy the VSN in a real-world scenario, we believe that high flexibility constitutes itself an important requirement, rather than a performance metric. On the hardware level, such flexibility is required to easily customize the platform subject to the application requirements, e.g., by adding/removing key components as already pointed out previously. On the software level, a good VSN platform should provide the networking and computational primitives on top of which any application involving transmission and processing of multimedia data could be implemented. Ideally, such software primitives should be based on well-accepted and easily-usable standards.

Recently, a new class of small embedded computing devices based on Linux has gained a lot of popularity among researchers and developers: the Raspberry PI, the BeagleBone Black, Arduino Yun and Intel Galileo are four of the most famous Linux-based open-source development boards available nowadays. Referring to the aforementioned requirements, such platforms are certainly good candidates for building a VSN node. First, the possibility of running Linux gives great flexibility on the software level and makes the development, debugging and deployment processes easy. Second, such platforms are generally equipped with several input/outputs peripherals, thus giving high flexibility on the hardware level. Third, they are all generally characterized by a low-power consumption, and are able to be battery-operated. Last, but not least, they are extremely cheap, being commercially available for less than \$100 USD.

In the following we introduce EZ-VSN, a flexible and open-source VSN hardware/software framework, which fulfils all the aforementioned requirements.

III. EZ-VSN System Architecture

This section provides a detailed description of the EZ-VSN software framework. We start with a general description of the proposed architecture, and how it can be used to create different instances of Visual Sensor Networks. Then, we provide a detailed description of the software building blocks that constitute the framework and how they can be interconnected to obtain different functionalities. The presented framework can be run on any Linux-based platform supporting OpenCV, regardless of the type of processor (X86 or ARM). It also

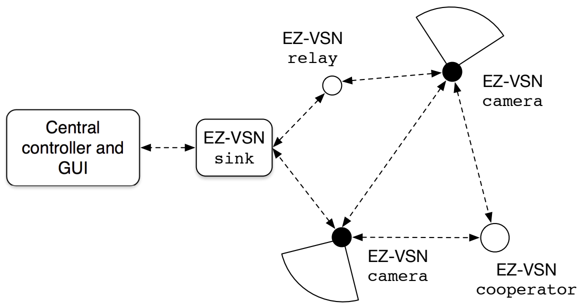


Fig. 1. EZ-VSN sample architecture. A Graphical User Interface is connected to a `sink` node to control and receive visual data from `camera` nodes. One or more `relay` nodes may be used to deliver the data between a `camera` and the `sink`. Several `cooperator` nodes may be added to the VSN and connected to `camera` nodes, to parallelize the computation of intense tasks.

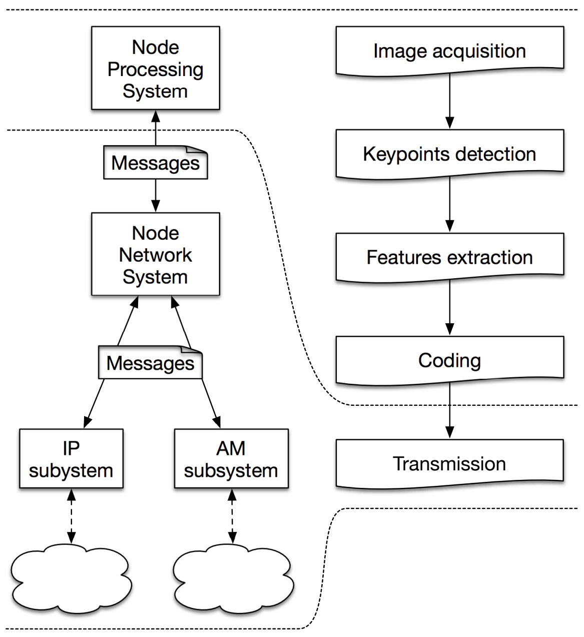


Fig. 2. EZ-VSN software architecture. On the left system components communicating with `messages`: the Node Processyng System (NPS), the Node Network System (NNS), the IP subsystem and the AM subsystem. On the right the system pipeline in the ATC case.

supports different wireless communication technologies such as IEEE 802.11 or IEEE 802.15.4. In case the latter is used, the framework also provide support for TinyOS-compliant transceivers[1]. Additional details on the tested hardware configurations are given in Section IV.

A. *EZ-VSN* `node`

With reference to Figure 1, the proposed framework allows the creation of different types of nodes in a Visual Sensor Network:

- EZ-VSN `camera`: these types of nodes play the major role in the VSN. Equipped with cameras and low power wireless communication devices (either based on IEEE 802.11 or IEEE 802.15.4 standards), such nodes acquire visual data from the environment and deliver it remotely to a central controller. Before transmission, the acquired images can be compressed in the pixel-domain (i.e., using the CTA paradigm), or processed to extract compact feature-based data (i.e., using the ATC paradigm).
- EZ-VSN `relay`: data delivery from camera nodes to the central controller can be performed either directly, or in a multi hop fashion. In the latter case, visual data is routed through one or more relay nodes. Again, such relay nodes may support different technologies for communication, such as IEEE 802.15.4 or IEEE 802.11.
- EZ-VSN `sink`: this node is responsible for collecting image or features data from the camera nodes and forwarding it to the central controller, where it can be displayed, stored or analyzed.
- EZ-VSN `cooperator`: the proposed framework also provides the possibility of adding a special type of nodes in the VSN, known as cooperators. Such nodes may not be equipped with a camera, but can be still added to the network to sense different kind of data (e.g., temperature, humidity), or as backup nodes. Their purpose is to help camera nodes in executing intense visual processing tasks, leveraging recent results in the area of parallel computation for VSN [21], [22].

[1]TinyOS is a lightweight operating system for 802.15.4 transceivers and it is widely popular among WSN developers

Regardless of its role in the network, each node runs the same software. With reference to Figure 2, the software architecture is composed of two main components, namely (i) the Node Processing System (NPS) and (ii) the Node Network System (NNS). The NPS implements the logic which operates each node in the VSN, together with several multimedia processing and encoding functionalities while data reception and transmission is managed by the NNS. Communication among these two components, and between different EZ-VSN nodes is based on Message objects.

In the following, we give a detailed description of each one of the aforementioned components.

1) Messages: The basic container for any kind of communication in EZ-VSN is a message. Each message is made of a header and a payload. The header, depicted in Figure 3, is composed of 13 bytes and contains a sequential number, the number of packets the message is made of, the packet id, the message type, source and destination unique IDs, the link technology the message is being transmitted with and the payload size. The payload varies depending on the message type. Message types defined in EZ-VSN are grouped in 3 different sets: control messages, data messages and other messages. New message types can be easily added following the existing ones as reference.

Control messages:

1) `START-CTA message`. Sent from the GUI to a `camera` through the `sink`, contains all the necessary parameters to operate in CTA mode: image width and height, JPEG quality factor, number of slices in which the image should be split before being sent. The image

0	1	2	3	4	5	6	7	8	9	10	11	12
SN	NumPkts		PktId		MT	SId	DId	LT	PayloadSize			

Fig. 3. Message header format (in bytes). SN = sequential number, unique for each message; NumPkts = number of packets a message is made of; PktId = packet id, between 0 and NumPkts-1; MT = message type; SId = source node unique ID; DId = destination node unique ID; LT = link type identifying the transmission technology used; PayloadSize = size, in bytes, of the message payload.

can be split into slices due to the fact that the system is able to work with unreliable transmission channels as 802.15.4. If a message gets corrupted only one slice is lost and visual analysis can take place on the other available slices.

2) `START-ATC message`. Sent from the GUI to a `camera` through the `sink`, contains all the necessary parameters to operate in ATC mode: detector and descriptor type, detection and description parameters, compression parameters and number of features to be sent back in a single message. As for image slices, features are sent back to the sink divided in different messages, according to the provided number of features for a single message. In this way, if a message is corrupted only a subset of the features is lost.
3) `STOP message`. Sent from the GUI to a `camera` through the `sink`, tells the `camera` to stop the existing processing tasks and flush any outgoing message.

Data messages:

1) `DATA-CTA message`. Sent from a `camera` to the GUI through the `sink`, contains the JPEG bitstream of the acquired image together with timing information about encoding and transmission time.
2) `DATA-ATC message`. Sent from a `camera` to the GUI through the `sink`, contains the bitstream of keypoints and their relative descriptors, together with timing information about detection, extraction, encoding and transmission time.

Other messages:

1) `NODE-INFO message`. Sent from a `camera` to the `sink` and from a `cooperator` to a `camera` to notify the sender existence in the network.
2) `ACK message`. Sent back from the receiver of a `DATA-CTA` or `DATA-ATC` message to probe the application bandwidth between the sender and the receiver.

2) Node Processing System: The NPS is the core of each EZ-VSN node and it is implemented following an asynchronous event-based fashion. In particular, the NPS reacts to messages received from the NNS by performing different actions. When a message is received it is first dispatched to the *Processing Queue* or to the *Service Queue*, depending on its type. The *Processing Queue* is devoted to the elaboration of computationally intensive tasks, such as image acquisition, keypoint detection, features extraction and coding. Conversely, the *Service Queue* is filled with messages that do not require intense computational resources, such as control messages. In order to guarantee a fast response time, each queue is executed in a separate thread (the *Processing Thread* and the *Service Thread*, respectively). This allows to manage control messages while executing intense processing tasks in background.

The behaviour of the *Node Processing System* strongly depends on the "role" of the node in the VSN:

- A `sink` node basically acts as a router and forwards messages from camera nodes to the central controller, and vice-versa. Since a `sink` node does not perform any processing, its *Processing Queue* is always empty and all the incoming messages are sent to the *Service Queue*.
- In a `camera` node the incoming `START-CTA` and `START-ATC` messages trigger the acquisition and processing of a new image. Therefore, such messages are delivered to the *Processing Queue*. All other message types are delivered to the *Service Queue*.
- A `cooperator` may receive a `START-ATC` and a `DATA-CTA` from a camera node, containing the information on how the cooperation should be performed and the chunk of data to process. The former message is inserted in the *Service Queue*, while the latter is managed by the *Processing Queue*.

The image processing capabilities of EZ-VSN are implemented in the NPS by relying on the open-source OpenCV[2] 2.4.11 C++ API, which allows to easily perform image acquisition and JPEG encoding/decoding. To maximize the performance of the system when executed on ARM-based platforms, we compiled OpenCV with *libjpeg-turbo*[3], so as to exploit parallelism offered by the NEON instruction set available on ARM processors, which are typically used on low-power architectures such as the BeagleBone Black or the Raspberry PI.

The NPS *Processing Thread* is also responsible for extracting and encoding local features from the acquired images. For feature extraction, OpenCV allows to use several types of extractors (including SIFT [23], SURF [24], and ORB [25]). Furthermore, we implemented an optimized version of the BRISK binary feature extractor, named BRISKOLA [26], which exploits NEON instructions and is therefore tailored to low-power ARM architectures. For what concerns features encoding, we implemented several encoding schemes, ranging from simple feature quantization and arithmetic encoding, to advanced lossless coding schemes tailored to low-cost binary features [27].

3) Node Network System: The NNS implements the networking primitives needed for receiving and transmitting messages from/to other nodes in the VSN. Each EZ-VSN node is identified by (up to) three addresses: a unique ID, an IP address (if the node is equipped with Wi-Fi) and a 16-bit TinyOS Active Message (AM) address (if equipped with a TinyOS-copmliant transceiver). Low-level handling of the two different wireless technologies is carried out by two specific components, namely the *IP subsystem* and the *AM subsystem*. The former is responsible for handling IP-based packets on the Wi-Fi interface, and it is implemented relying on the *Boost* C++

[2]http://opencv.org/
[3]http://libjpeg-turbo.virtualgl.org/

libraries[4] for asynchronous input/output (`Boost::asio`) and message serialization (`Boost::serialization`). Conversely, the *AM subsystem* is responsible for handling packets transmission/reception from the 802.15.4 transceiver. In this case, a third-party serial driver is used to communicate with the serial interface of the transceiver.

a) Transmitting Messages: When the NPS needs to transmit a message on a particular radio interface, it sets the LT (LinkType) field of the message header accordingly and delegates the message to the NNS. Here, depending on the chosen interface, the message is forwarded to one of the two radio subsystems. In case IP-based communication is requested, the message is first serialized using *Boost* and the corresponding bitstream is transmitted to the destination address. At the transport layer, we used TCP to provide reliable communication between nodes in the network. However, other IP-based transport protocols may be used easily (e.g., UDP). Differently, if the message has to be transmitted on the 802.15.4 interface, the AM subsystem implements message serialization and packetization according to the 802.15.4 MTU. The resulting bitstream is then transmitted to the transceiver through serial communication. On the 802.15.4 transceiver, packets are forwarded from the serial to the radio interface by an ad-hoc TinyOS application. At the lowest communication layers, such a design allows again for great flexibility in the routing/MAC protocols to be used. If IP-based communication over Wi-Fi is used, it is possible to use directly the tools provided by the Linux Kernel, e.g. *iproute2*, in order to modify the routing/forwarding tables of a node. In case IEEE 802.15.4 communication is used, the transceiver may be loaded with a specific TinyOS application implementing a particular protocol at the MAC or routing layer. As an example, TinyOS already contains implementation of CSMA-CA and TDMA access protocols and several routing protocols tailored to wireless sensor networks, such as RPL [28].

b) Receiving Messages: When an IP-based bitstream is received, the *IP subsystem* deserializes it and produces a message which is then passed from the NNS to the NPS. In case the bitstream is received by the *AM subsystem*, the single packets have to be ordered and re-assembled before the message can be deserialized. We implemented a specific buffer to store incoming packets and eliminate duplicates. The buffer may be also used for identifying and requesting lost packets, so as to provide a starting point for reliable transmission protocol implementation.

B. Central Controller and GUI

The central controller implements all the logic needed to control the VSN. In particular, it features a Graphical User Interface (GUI) that allows to display and analyze the information received from camera nodes, remotely modify the VSN operational parameters and monitor the performance of the VSN in real-time. The GUI is built upon QT5[5], an open-source, cross-platform framework that allows quick development and a powerful C++ support. With reference to Figure 4, the GUI is composed of several graphical components:

1) Camera Settings (Figure 4 - D): The *Camera settings* window allows to easily control each camera in the network through a set of tabs. For each camera, it is possible to change in real-time several operational parameters:

- Operative paradigm and Visual Task: each camera can work according to either CTA (i.e., transmitting images) or ATC (i.e., transmitting features) paradigm. For both operative paradigms, several parameters can be changed in real-time. Moreover, the data received by each camera (either images or features) can be used to perform either object recognition or parking monitoring. Additional details on the application scenarios and the corresponding controllable parameters are given in Section V.
- Transmission technology: each camera can be independently set to work with either 802.11 or 802.15.4 wireless transmission. In the former case, it is also possible to control the maximum transmission rate of the wireless interface, so as to mimick bandwidth constrained scenarios. The desired transmission rate is set in a specific field of `START-ATC` and `START-CTA` messages. On camera nodes, the maximum transmission rate is set according to the received value, using the *tc* Linux command for traffic shaping.

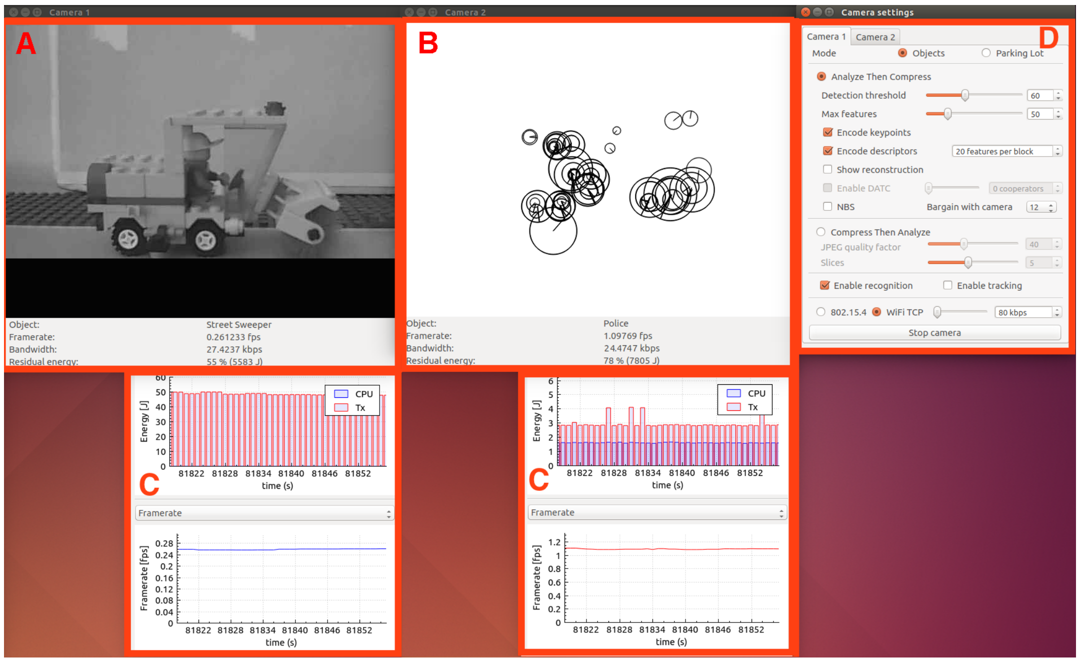
Finally for each camera a Start/Stop button is provided to trigger the transmission of a `START-ATC` or a `START-CTA` message. Upon reception of the corresponding `DATA-ATC` or `DATA-CTA` message, another start message will be transmitted automatically, unless the stop button is pressed.

2) Camera view (Figure 4 - A,B): The *Camera view* window shows the content received from camera nodes. If the camera is set to work according to CTA, a VGA image is shown (Figure 4 - A). Conversely, if the camera is working in ATC mode, no pixel-domain image can be shown and local features are drawn in the camera view (Figure 4 - B). Each *Camera view* window also reports camera-specific information, such as the estimated end-to-end application bandwidth, the estimated residual energy and other information related to the selected visual application (see Section V for additional details).

3) Performance monitor (Figure 4 - C): This window allows to monitor in real-time application-specific performance and operational parameters. In particular, the following metrics can be monitored:

- Application frame rate: a dedicated timer is started each time a `START-ATC` or a `START-CTA` message is transmitted from the GUI, and stopped once the corresponding `DATA-ATC` or `DATA-CTA` message is received from the camera. The elapsed time t_f, and the corresponding frame rate $F_\mathrm{f} = 1/t_\mathrm{f}$ is shown.
- Energy consumed per frame: as explained in Section III-A1, each camera inserts in each `DATA-CTA` message the time elapsed for image encoding $t_\mathrm{cpu}^\mathrm{CTA}$ and in each `DATA-ATC` message the time elapsed for features extraction and encoding $t_\mathrm{cpu}^\mathrm{ATC}$. On the `sink` node, a dedicated timer is used to estimate the transmission time of data messages from camera nodes t_tx. The transmission and processing times

[4]www.boost.org
[5]http://www.qt.io/



Object: Street Sweeper
Framerate: 0.261233 fps
Bandwidth: 27.4237 kbps
Residual energy: 55 % (5583 J)

Object: Police
Framerate: 1.09769 fps
Bandwidth: 24.4747 kbps
Residual energy: 78 % (7805 J)

Camera settings

Camera 1 | Camera 2

Mode: [✓ Objects] Parking Lot

[✓ Analyze Then Compress]
Detection threshold: 60
Max features: 50
[✓ Encode keypoints]
[✓ Encode descriptors] 20 features per block
Show reconstruction
Enable DATC 0 cooperators
NBS Bargain with camera 12

Compress Then Analyze
JPEG quality factor: 40
Slices: 5

[✓ Enable recognition] Enable tracking

802.15.4 [✓ WiFi TCP] 80 kbps

Stop camera

Fig. 4. EZ-VSN GUI screenshot in a two camera setup. A) Camera view in CTA mode: the compressed image is displayed. B) Camera view in ATC mode: keypoints are displayed. C) Performance monitor: in the upper part the energy consumption, in the lower part the framerate. D) Camera settings.

are forwarded from the `sink` node to the GUI, which estimates the energy consumed per frame multiplying the estimated times by the corresponding power consumptions, e.g. $E_f = t_{tx} \cdot P_{tx} + t_{cpu} \cdot P_{cpu}$. Note that P_{tx} and P_{cpu} are obtained from Table II.

The *Performance monitor* is particularly useful to understand the interplay between operational parameters and system performance and to assess the benefits of a specific system configuration from both the energy and application framerate point of view, while varying the parameters in a real-time fashion.

IV. Hardware Architecture

As explained in Section III, the proposed software framework can be potentially executed on any architecture, ranging from standard workstations and laptops to low-power microcomputers. Again, such flexibility on the platform used to run the framework is extremely useful: the former class of platforms may be easily used for development and debugging purposes, while the latter is tailored to real-world VSN deployments and testing. Since the source code of the framework does not change between the two configurations, all phases of development, testing and debugging are simplified.

In this Section, we give details on the hardware used for implementing real-world VSN running the proposed software. Although several options are available, in this paper we focus on the BeagleBone Black Revision C embedded microcomputer. Such platform features a 1 GHz ARM Cortex-A8 processor, 512 MB of RAM memory, 4 GB of eMMC flash memory and is capable of running Linux distributions based on the 3.8+ kernel (e.g., Debian, Ubuntu and Android). The Cortex-A8 processor provides excellent computational capabilities, yet consuming less power than other similar platforms on the market. As an example, the ARM11 processor mounted on the Raspberry PI, which may constitute another viable option for implementing a VSN node, exhibits a lower peak DMIPS/MHz value at a higher power consumption (see Section IV-B).

The BeagleBone Black also features a great range of peripherals connections including USB, Ethernet and two 46 pin expansion headers which expose several input/output interfaces (analog, GPIO, SPI and I2C), thus making extremely easy to connect external sensor devices and communication modules. Several plug-in "capes" specifically designed for the BeagleBone are also available, allowing developers to quickly augment the board capabilities with additional functionalities (camera, battery-power, inertial navigation, GSM/GPRS communication and many others)[6]. Despite such powerful capabilities, the BeagleBone Black has two fundamental features: first, it is extremely cheap (available for less then \$50USD). Second, it is characterized by a power consumption of less than 2W (the instantaneous power consumption varies depending on activity mode and processor speed) and may also leverage the different power modes available for the ARM Cortex-A8 microprocessor (including standby and sleep modes). Furthermore, the credit-card size and the capability of being battery operated make the BeagleBone Black a perfect candidate for implementing low-power applications which require complex processing, as it is the case of Visual Sensor Networks.

A. Image acquisition and wireless communication

In order to acquire images and videos, an EZ-VSN node can be equipped either with any USB camera compatible with Video4Linux Version 2 (V4L2), or with acquisition devices

[6]http://elinux.org/Beagleboard:BeagleBone_Capes

TABLE I
COMPARISON OF POWER CONSUMPTION IN DIFFERENT POWER MODES

	BeagleBone Black @1000MHz	Raspberry PI @700MHz
Boot (Peak) [W]	2.28	2.31
Idle (Average) [W]	1.04	1.58
Active (Average) [W]	1.30	1.70
Sleep (Average) [W]	0.26	N/A

created on purpose for the BeagleBone Black, such as the ultra low-power RadiumBoard HD Camera Cape[7]. Clearly, the choice of the camera device depends on the application pixel resolution and/or power consumption requirements. Similarly, several options are available to provide an EZ-VSN node with wireless communication capabilities. Both IEEE 802.15.4 or IEEE 802.11 compliant USB dongles can be used, as well as specifically designed add-on capes, again providing high flexibility. In particular, VSN applications which require high-bandwidth may leverage wireless communication based on the IEEE 802.11 WiFi standard: this can be obtained on an EZ-VSN node by connecting a low-power USB WiFi adapter such as the Netgear WNA1100 or the Tp-Link TL-WN725N, at the cost of increasing energy consumption. For those applications where bandwidth can be traded off for energy, an EZ-VSN node may support the use of IEEE 802.15.4-compliant devices to achieve low-power wireless communication. Again, several options are available in this case: here we use a MEMSIC TelosB node, which can be easily attached to the BeagleBone's USB port. The reason for choosing the TelosB lies on the fact that it supports many of the operating systems for WSNs (TinyOS, Contiki, FreeRTOS), further increasing the degree of flexibility of the entire system, and at the same time giving the opportunity to reuse already existing code for networking primitives and routing/transport protocols. In particular, we relied on TinyOS for operating TelosBs coupled to EZ-VSN nodes, due to its wide popularity among WSNs developers.

B. *Energy profiling*

To assess the energy performance of an EZ-VSN node, we performed several experimental measurements coupling the BeagleBone Black with different image acquisition and wireless communication devices and analyzing the different power modes available on the ARM Cortex-A8 microprocessor. All the power consumption tests were performed using an Adafruit INA219 DC current sensor connected over I2C to an Arduino Mega 2560. The power to the device under test is provided by a third party 5V 3A external supply. The 0.1Ω measurement resistor of the INA219 is placed in series between the external power supply and the +5V pin of the BeagleBone. The INA219 onboard ADC measures the dropout voltage across the resistor and the voltage supplied to the device under test, thus giving enough information to compute the current drawn by the device and its power consumption.

As a first test, we compared the energy performance of the BeagleBone Black with the one of the Raspberry PI 1 Model B, a $20USD, credit-card-sized platform which features

TABLE II
EZ-VSN NODE POWER CONSUMPTION IN DIFFERENT HARDWARE CONFIGURATION [W]

Attached Devices	Status	0.3GHz	0.6GHz	1GHz
None	Idle	0.72	0.80	1.04
	Active	0.79	0.95	1.30
Logitech QuickCam VGA camera	Idle	1.06	1.14	1.38
	Active	1.61	1.83	2.13
Logitech C170 720p camera	Idle	0.94	1.02	1.26
	Active	1.21	1.42	1.82
Logitech C310 720p camera	Idle	1.24	1.32	1.56
	Active	1.87	2.08	2.42
RadiumBoard HD camera cape	Idle	1.24	1.32	1.56
	Active	1.57	1.69	1.93
Memsic TelosB	Active/Tx/Rx	0.93	1.01	1.25
Netgear WNA1100 WiFi Adapter	Idle	0.88	0.96	1.20
	Active	1.17	1.25	1.49
	Tx	1.69	1.76	2.05
	Rx	1.28	1.43	1.58
Tp-Link TL-WN725N WiFi Adapter	Idle	0.87	0.95	1.19
	Active	1.02	1.10	1.34
	Tx	1.44	1.54	1.84
	Rx	1.12	1.28	1.50
HD camera cape + Memsic Telosb	Idle	1.43	1.51	1.75
	Active	1.78	1.90	2.14
HD camera cape + WiFi Tp-Link	Idle	1.68	1.76	2.00
	Active	1.87	1.99	2.23

a 700MHz ARM microprocessor and may thus constitute a viable alternative for being used as a basis for the EZ-VSN node. Table I reports the average power consumption for the two tested architectures, in different activity modes when no external devices are attached. As one can see, the BeagleBone Black is a clear winner for a number of reasons: first, it exhibits lower power consumption in all activity modes; second, differently from the Raspberry PI, it supports a sleep mode where the power consumption is as low as 260 mW; finally, it features a more powerful processor with tuneable base-clock frequency.

Having said that, we provide in Table II a detailed analysis of the BeagleBone Black power consumption when coupled with different devices, and in different configurations of microprocessor speed. The operating system running on the BeagleBone Black during the tests was Linux Debian 3.8.13[8]. Columns 3 to 5 report the power consumption when increasing the BeagleBone Black microprocessor speed: clearly, the power consumption increases as the processor speed increases. Rows 2 to 5 shows the power consumption when different camera models are attached to the platform. We tested the energy performance of EZ-VSN with four different camera models, namely (i) a Logitech QuickCam VGA camera, (ii) a Logitech C170 720p camera, (iii) a Logitech C310 720p camera and (iv) a RadiumBoard HD Camera Cape. The first three models use USB to communicate with the BeagleBone, while the latter uses the custom 46 pin connectors. As one can see from Table II, the different camera models are characterized by very different power consumptions: as an example, the QuickCam VGA camera requires more power than the Logitech C170, even if the latter acquires images with a better quality. This justifies the adoption of a flexible architecture

[7]http://www.radiumboards.com/HD_Camera_Cape_for_BeagleBone_Black.php

[8]http://rcn-ee.net/deb/wheezy-armhf/v3.8.13-bone72/

that can support different camera models, so that they can be substituted based on the lowest cost/energy consumption. Rows 6 to 8 report the power consumption of the BeagleBone coupled with a 802.15.4 and two 802.11 communication devices. As expected, 802.15.4 communication requires less power than Wi-Fi, which can be extremely high especially when transmitting data. Finally, the last two rows of Table II show the total power consumption of a complete EZ-VSN node, equipped with the RadiumBoard HD Camera Cape and with (i) a Memsic TelosB for 802.15.4 low power communication (row 8), and (ii) a Tp-Link TL-WN725N Wi-Fi USB dongle. The power consumption in such configurations varies from a minimum of 1.43 W to a maximum of 2.23 W. We also tested the EZ-VSN node energy performance with other operating systems, which are omitted here for space reasons. The complete set of measurements is available at www.greeneyesproject.eu.

V. Application Scenarios

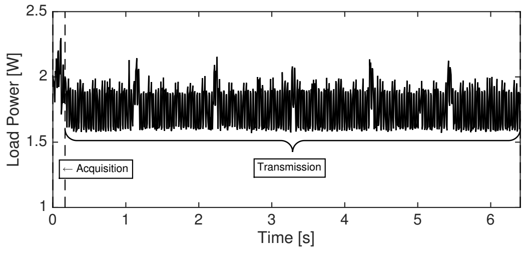
The flexibility of the proposed framework can be leveraged to implement several different VSN applications. In particular, we focus on visual analysis applications, where the content acquired by camera nodes is processed and analyzed to extract high-level semantic concepts. In this section we provide two representative examples that have been implemented using the EZ-VSN framework: object recognition and parking lot monitoring. In addition to describing how each application scenario was implemented, we also exploit the framework to assess and compare the performance of the novel Analyze-then-Compress paradigm and the traditional Compress-then-Analyze approach. Such comparison demonstrates the flexibility of the proposed system in implementing and evaluating practical solutions for VSNs. The following scenarios and the corresponding performance evaluation have been carried out with an EZ-VSN camera node in the configuration that requires the lowest amount of energy, that is coupled with the RadiumBoard HD Camera Cape and the IEEE 802.15.4 compliant TelosB dongle. On the latter, we relied on the CSMA/CA protocol with data acknowledgment provided by TinyOS at the MAC layer and a routing protocol with ad-hoc static routes hard-coded on each node of the VSN.

A. Object recognition

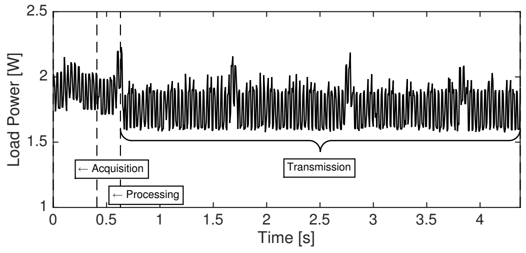
Many monitoring applications for VSNs require to recognize a particular object. Upon recognition other actions may be performed, such as tracking the particular object or launching an alarm. Object recognition is typically performed following these steps: first, local visual features are extracted from the image under processing. Such features are then matched against a database of labeled features extracted from known objects to find the most similar one. The particular type of features to be used and how matching is performed can vary from case to case, but the general process remains the same. It is clear that, in the VSNs scenario, deciding *where* to perform feature extraction plays a crucial role. In traditional systems, the CTA paradigm is followed. That is, the acquired image is first compressed relying on a proper algorithm (e.g., JPEG), and then transmitted wirelessly to a central controller which performs feature extraction and matching. Conversely, in the ATC paradigm, the camera node may directly perform features extraction on the acquired image, and then transmit a compressed version of such features to the central controller for further matching. We implemented both the CTA and ATC paradigms for object recognition using EZ-VSN. The following paragraphs give details on the role of each node in the VSN, depending on the chosen paradigm.

1) Compress-then-Analyze: Upon receiving a `START-CTA` command from the central controller, a camera node acquires a new image and compresses it with JPEG. The bitstream generated by the JPEG encoder is then encapsulated in either 802.11 or 802.15.4 frames (depending on the communication technology chosen from the GUI) and transmitted to the sink node in a `DATA-CTA` message. Note that several application parameters, such as the image resolution, the JPEG compression quality factor and the wireless technology to adopt for image transmission are all contained in the `START-CTA` message and can therefore be changed in real-time from the GUI. Upon receiving the `DATA-CTA` message, the sink node forwards it to the central controller. Here, the received image is displayed on the GUI and an object recognition algorithm is executed. In particular, SIFT features are extracted from the image and matched with the ones extracted from a database of labeled images. Features matching is performed following the best practices in the area of object recognition, using the ratio-test and removing outliers through a geometric consistency check with RANSAC. This allows to rank the images in the database, with the most similar ones at the top of the list. The label corresponding to the majority of objects in the first ten positions of the ranked list is returned as the recognized object, as shown in Figure 4 - A.

2) Analyze-then-Compress: If the ATC mode is selected, the central controller transmits a `START-ATC` message to a camera node. Upon reception, the camera node acquires an image and extracts local visual features from it. For feature extraction, we used an optimized version of the BRISK algorithm [26], tailored to low-power NEON-enabled ARM architectures. Optionally, the extracted features can be compressed with an ad-hoc arithmetic encoder, specifically designed to exploit the redundancy between the elements of each feature [27]. Similarly to the CTA case, the `START-ATC` message contains several parameters, such as the type of feature extractor algorithm to be executed, the feature detection threshold, the maximum number of features to be transmitted and if such features should be compressed or not. Features data is then encapsulated in a `DATA-ATC` message which is then transmitted back to the sink. Here, the received features are decoded and matched with the ones contained in the database, as done for CTA. Since in this case it is not possible to display the received image, as the pixel-domain information is not transmitted in ATC, the received features are represented as black circles with a radius proportional to their scale parameter, as shown in Figure 4 - B. Additionally, an approximation of the image starting from the knowledge of the visual features only can be computed and shown. In particular, we followed the approach presented in [29], where

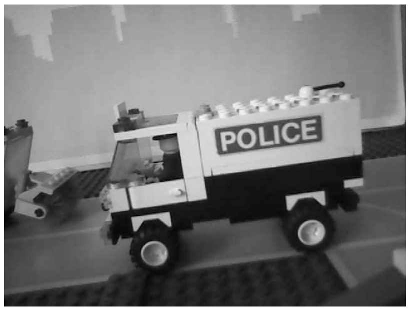


(a)

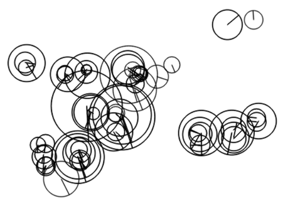


(b)

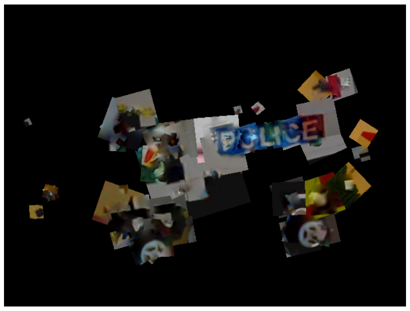
Fig. 5. Power consumption of the proposed VSN for object recognition in CTA mode (a) and ATC mode (b) at a target application mAP of 0.58. For CTA, this corresponds to a JPEG quality factor equal to 1. The same accuracy is reached in ATC with the transmission of 100 BRISK descriptors, reducing the transmission time and the corresponding overall energy consumption.



(a)



(b)



(c)

Fig. 6. *Camera view* representing the same sample object in CTA mode (a), ATC mode (b) and the image reconstructed from local features (c)

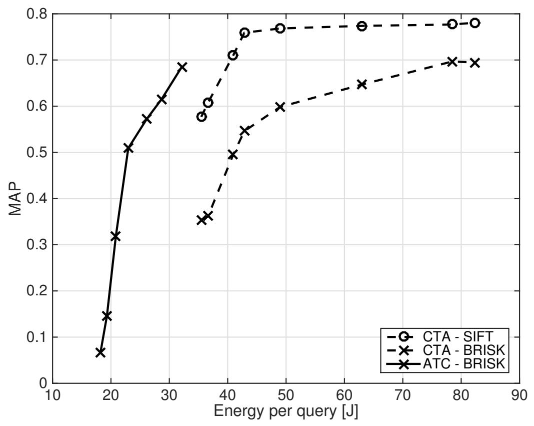


Fig. 7. Object recognition application. Mean Average Precision vs Energy in ATC and CTA mode. The energy is measured integrating power over time during acquisition, extraction (ATC only), coding and transmission phases. For the ATC case 20, 40, 60, 80, 100, 150, 200 BRISKOLA features are used. For the CTA-SIFT case and the CTA-BRISK case the JPEG quality factor was varied in the range 1,3,5,7,10,20,30,40.

the image is reconstructed as a composition of image patches, whose local features match the ones received from the camera node. Figure 6 shows an example of image reconstruction from features.

3) Performance evaluation: We used the EZ-VSN framework to assess the performance of CTA and ATC paradigms for object recognition. We focus in particular on two performance measures:

- *Energy consumed per image:* with the same measuring setup described in Section IV, we measured the energy spent by a camera node in the acquisition, processing and transmission phases for both the CTA and ATC paradigms. Figure 5(a) and 5(b) show the measured instantaneous power consumption in two particular cases. Note that the total energy spent per image can be obtained by integrating the power consumption over one duty cycle.
- *Accuracy of object recognition:* we relied on the ZuBuD[9] dataset, which consists of 1005 color images of 201 buildings of the city of Zurich. Each building has five VGA images (640x480), taken at random arbitrary view points under different seasons, weather conditions and by two different cameras. A separate archive containing 115 query images of the same buildings is available as query dataset. Such query images, or the features extracted from them, are used as queries and transmitted to the sink from a camera node. As usual in evaluating the performance of object recognition / image retrieval systems, we use the Mean of Average Precision **(MAP)** to measure the quality of the ranked list obtained on the central controller after feature matching.

In the case of CTA, we repeated the test several times, each time changing the JPEG quality factor and consequently the amount of data to be delivered and the energy spent in the transmission process. At the central controller, we evaluated the MAP using both SIFT and BRISK features. The former allows to obtain the best accuracy performance at the cost of time-consuming processing. The latter is a faster alternative

[9]http://www.vision.ee.ethz.ch/showroom/zubud/index.en.html

that trades off accuracy for computational time [30]. For what concerns ATC, we repeated the test several times, each time changing the number of features to be transmitted from the camera to the sink node. Figure 7 shows the energy-accuracy performance of CTA and ATC. As one can see, ATC allows to spend less energy with respect to CTA, at the same target accuracy. The energy savings vary from a minimum of 25% at a MAP of 0.7 to a maximum of about 50% at a MAP of 0.37. Clearly, if the energy constraints are not tight, transmitting a JPEG encoded image allows to use sophisticated feature extraction algorithms at the sink node, thus maximizing the application accuracy. As mentioned above, even if ATC is used a reconstruction of the original image may be obtained. The quality of the reconstructed image increases as the number of features received from the camera node and the size of the database of patches increase. As a rule of thumb, 100 descriptors are enough to obtain a "good" image reconstruction [31]. Considering the ZuBuD dataset and the results in Figure 7, this translates to a MAP higher than 0.55 for acceptable reconstruction.

B. Parking monitoring

In the context of Smart Cities, parking lot monitoring (also known as smart parking) has recently gained a lot of attentions from both the scientific community and the industry [32]. By combining data from parking lots with web based services and intelligent displays, smart parking allows drivers to find vacant parking lots near their destinations quickly and easily. This provides several benefits, such as fewer CO^2 emissions from cars, reduced traffic congestion and finally, less stressed and happier citizens. VSNs constitute a natural choice for the scenario of parking lot monitoring, as one single camera can cover several parking lots, thus making the system scalable. Similarly to the case of object recognition, assessing if a parking lot is vacant or not is usually accomplished by extracting local features from the acquired image and processing them with a specialized classification algorithm. Again, such features may be extracted at the central controller after image transmission (CTA) or directly at the camera nodes (ATC).

1) Compress-then-analyze: In CTA, a camera acquires an image covering several parking lots, compresses it with JPEG and transmits it to a central controller in a `DATA-CTA` message. Such process is triggered by the central controller with a `START-CTA` message. Here, the image is processed according to the following steps:

- Parking space detection: let C be the number of parking spaces present in the image. Since the camera is likely to be statically deployed, we assume to know a-priori, for each parking space, the pixel coordinates of its center $\mathbf{x}_c$, together with its width w_c and height h_c, with $c = 1 \dots C$.
- Feature extraction: for each parking space, a square subregion of size $\min(w_c, h_c)$ centered in $\mathbf{x}_c$ is extracted from the image. Each subregion is converted from RGB to HSV colorspace, and the hue component is used to populate a histogram $\mathbf{h}_c$ (see Figure 8). Such histograms are used as local features to determine whether or not a parking space is vacant.

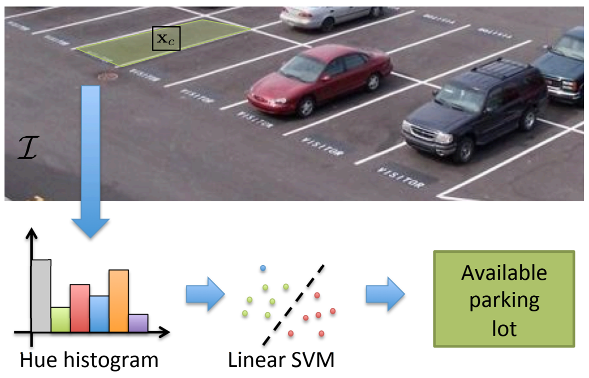
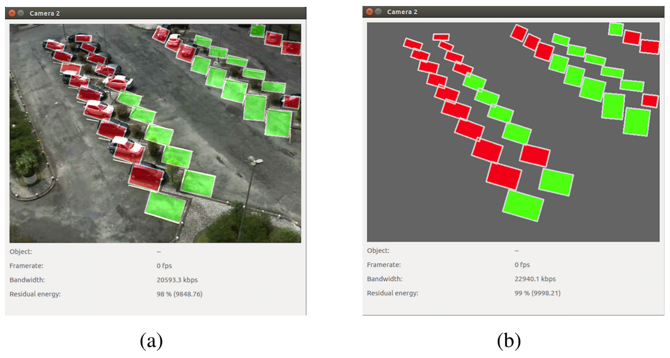


Fig. 8. Parking Lot monitoring: a feature vector is computed by means of a histogram of pixel hue values. A linear SVM is employed to classify each parking lot as either vacant or occupied.



Object: --
Framerate: 0 fps
Bandwidth: 20593.3 kbps
Residual energy: 98 % (9848.76)

Object: --
Framerate: 0 fps
Bandwidth: 22940.1 kbps
Residual energy: 99 % (9998.21)

Fig. 9. Camera view in *Parking Lot monitoring* mode for (a) the CTA paradigm and (b) the ATC paradigm.

- Parking space classification: a linear SVM classifier is trained with normalized hue-histograms and regularization constant $K = 100$. The classifier is then used to decide if a particular parking space is vacant or not (Figure 9(a))

2) Analyze-then-compress: From the GUI, the operation of the VSN can be changed in real-time by issuing a `START-ATC` message. In this case, the camera node will process the acquired image by extracting the hue-histograms. The location and size of each parking space, the number of bins to use in each histogram feature and the quantization parameters (i.e., the number of bits to be used for representing the value of each histogram bin) are all contained in the `START-ATC` message. Upon computation of the histogram features, the camera node transmits them to the central controller using a `DATA-ATC` message. There, the SVM classifier is used to infer the occupancy of each parking lot (Figure 9(b)).

3) Performance evaluation: Similarly to the case of object recognition, we evaluated the performance of the two paradigms from both the energy and accuracy points of view also for the scenario of parking monitoring. The energy consumed per image is measured with the same setup used before, while the accuracy evaluation has been carried out by relying on the Pk-lot dataset provided in [33], which contains images of parking lots under different weather conditions (overcast, sunny and rainy periods), divided in three different datasets (UFPR04, UFPR05, PUCPR, containing respectively 28, 37

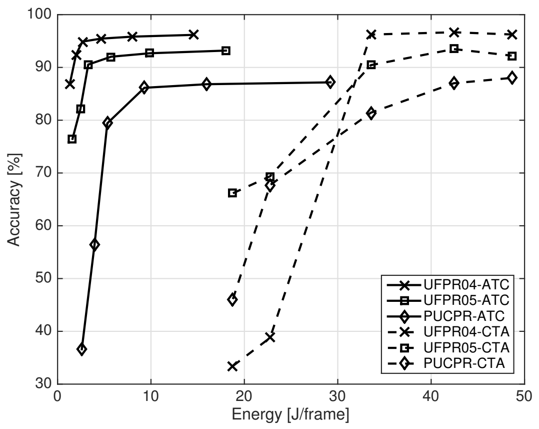


Fig. 10. ParkingLot monitoring application. Accuracy vs Energy in ATC and CTA mode. The energy is measured integrating power over time during acquisition, extraction (ATC only), coding and transmission phases. For the ATC case 6, 12, 23, 45, 90 and 180 bins are used. For the CTA case JPEG quality factor is set to 7,10,20,30 and 40

and 100 parking lots). Each image in the datasets is labeled with information on the vacant/occupied parking lots and each dataset is divided in a training and testing set. We used the training set to train the SVM classifier, and all experiments were then performed on the testing sets. We pre-loaded images from the datasets on the camera nodes and resized them at VGA resolution. As done for the object recognition scenario, for CTA we varied each time the JPEG quality factor in the range {7,10,20,30,40,}. For ATC, we used hue histograms with increasing number of bins in the range {6,12,23,45,90,180}. This allowed to draw different curves in the energy-accuracy plane, which are illustrated in Figure 10. As one can see, also in this case ATC outperforms CTA for all the tested datasets. In particular, at the maximum achievable accuracy ATC consumes just one quarter of the energy needed for CTA. Again, a clear tradeoff between energy consumption and task accuracy is evident. Note also that the SVM classifier used here was trained with images captured with different lighting and weather conditions. Multiple classifiers, specific to the particular lighting and weather condition, may be trained and used to boost the accuracy performance.

VI. Conclusions

Reliable architectures for testing VSN solutions are key to the success and the dissemination of such technology. In this paper, we have presented EZ-VSN, a complete and flexible framework to allow quick experimentation in the field of visual sensor networks. The proposed system is composed of (i) a low-cost, low-power hardware platform supporting both IEEE 802.11 and IEEE 802.15.4 wireless technology, (ii) the software needed to operate the hardware platform, including a set of image processing and networking primitives based on widely accepted and open-source C++ standards and (iii) a cross-platform Graphical User Interface to control in real-time the performance of the visual sensor network. We have demonstrated the flexibility of EZ-VSN implementing two different application scenarios and evaluating the benefits of the novel ATC paradigm compared to the traditional CTA one. The complete framework is made publicly available for research purposed at www.greeneyesproject.eu. As future research directions, we plan to use the framework to test in a real-life scenario several recently proposed solutions in the field of visual sensor networks, including distributed extraction of visual features from overlapping fields of view, and joint optimized multi-view encoding and routing of local features.

References

[1] E. Bentley, L. Kondi, J. Matyjas, M. Medley, and B. Suter, “Spread spectrum visual sensor network resource management using an end-to-end cross-layer design,” *Multimedia, IEEE Transactions on*, vol. 13, no. 1, pp. 125–131, Feb 2011.

[2] C. Li, J. Zou, H. Xiong, and C. W. Chen, “Joint coding/routing optimization for distributed video sources in wireless visual sensor networks,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 21, no. 2, pp. 141–155, Feb 2011.

[3] N. Kouadria, N. Doghmane, D. Messadeg, and S. Harize, “Low complexity dct for image compression in wireless visual sensor networks,” *Electronics Letters*, vol. 49, no. 24, pp. 1531–1532, November 2013.

[4] P. Wang, R. Dai, and I. Akyildiz, “A spatial correlation-based image compression framework for wireless multimedia sensor networks,” *Multimedia, IEEE Transactions on*, vol. 13, no. 2, pp. 388–401, April 2011.

[5] S. Paniga, L. Borsani, A. Redondi, M. Tagliasacchi, and M. Cesana, “Experimental evaluation of a video streaming system for wireless multimedia sensor networks,” in *Ad Hoc Networking Workshop (Med-Hoc-Net), 2011 The 10th IFIP Annual Mediterranean*, June 2011, pp. 165–170.

[6] A. Redondi, L. Baroffio, M. Cesana, and M. Tagliasacchi, “Compress-then-analyze vs. analyze-then-compress: Two paradigms for image analysis in visual sensor networks,” in *Multimedia Signal Processing (MMSP), 2013 IEEE 15th International Workshop on*, Sept 2013, pp. 278–282.

[7] B. Girod, V. Chandrasekhar, D. Chen, N.-M. Cheung, R. Grzeszczuk, Y. Reznik, G. Takacs, S. Tsai, and R. Vedantham, “Mobile visual search,” *Signal Processing Magazine, IEEE*, vol. 28, no. 4, pp. 61–76, July 2011.

[8] L. Baroffio, M. Cesana, A. Redondi, and M. Tagliasacchi, “Performance evaluation of object recognition tasks in visual sensor networks,” in *Teletraffic Congress (ITC), 2014 26th International*, Sept 2014, pp. 1–9.

[9] L. Baroffio, M. Cesana, A. Redondi, M. Tagliasacchi, and S. Tubaro, “Coding visual features extracted from video sequences,” *Image Processing, IEEE Transactions on*, vol. 23, no. 5, pp. 2262–2276, May 2014.

[10] I. Akyildiz, T. Melodia, and K. Chowdhury, “Wireless multimedia sensor networks: Applications and testbeds,” *Proceedings of the IEEE*, vol. 96, no. 10, pp. 1588–1605, Oct 2008.

[11] K. Abas, C. Porto, and K. Obraczka, “Wireless smart camera networks for the surveillance of public spaces,” *Computer*, vol. 47, no. 5, pp. 37–44, May 2014.

[12] A. Seema and M. Reisslein, “Towards efficient wireless video sensor networks: A survey of existing node architectures and proposal for a flexi-wvsnp design,” *Communications Surveys Tutorials, IEEE*, vol. 13, no. 3, pp. 462–486, Third 2011.

[13] S. Hengstler, D. Prashanth, S. Fong, and H. Aghajan, “Mesheye: A hybrid-resolution smart camera mote for applications in distributed intelligent surveillance,” in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, April 2007, pp. 360–369.

[14] W.-C. Feng, E. Kaiser, W. C. Feng, and M. L. Baillif, “Panoptes: Scalable low-power video sensor networking technologies,” *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 1, no. 2, pp. 151–167, May 2005. [Online]. Available: http://doi.acm.org/10.1145/1062253.1062256

[15] D. Lymberopoulos and A. Savvides, “Xyz: A motion-enabled, power aware sensor node platform for distributed sensor network applications,” in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, ser. IPSN ’05. Piscataway, NJ, USA: IEEE Press, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1147685.1147762

[16] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava, “Cyclops: In situ image sensing and interpretation in wireless sensor networks,” in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, ser. SenSys ’05.

New York, NY, USA: ACM, 2005, pp. 192–204. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098939

[17] R. Kleihorst, A. Abbo, B. Schueler, and A. Danilin, "Camera mote with a high-performance parallel processor for real-time frame-based video processing," in *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, Sept 2007, pp. 69–74.

[18] A. Kandhalu, A. Rowe, and R. Rajkumar, "Dspcam: A camera sensor system for surveillance networks," in *Distributed Smart Cameras, 2009. ICDSC 2009. Third ACM/IEEE International Conference on*, Aug 2009, pp. 1–7.

[19] L. Nachman, J. Huang, J. Shahabdeen, R. Adler, and R. Kling, "Imote2: Serious computation at the edge," in *Wireless Communications and Mobile Computing Conference, 2008. IWCMC '08. International*, Aug 2008, pp. 1118–1123.

[20] J. Schiller, A. Liers, and H. Ritter, "Scatterweb: A wireless sensornet platform for research and teaching," *Comput. Commun.*, vol. 28, no. 13, pp. 1545–1551, Aug. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2004.12.044

[21] A. Redondi, M. Cesana, M. Tagliasacchi, I. Filippini, G. Dan, and V. Fodor, "Cooperative image analysis in visual sensor networks," *Ad Hoc Networks*, vol. 28, no. 0, pp. 38 – 51, 2015.

[22] E. Eriksson, G. Dan, and V. Fodor, "Real-time distributed visual feature extraction from video in sensor networks," in *Distributed Computing in Sensor Systems (DCOSS), 2014 IEEE International Conference on*, May 2014, pp. 152–161.

[23] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

[24] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, Jun. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.cviu.2007.09.014

[25] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Proceedings of the 2011 International Conference on Computer Vision*, ser. ICCV '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2564–2571. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2011.6126544

[26] L. Baroffio, A. Canclini, M. Cesana, A. Redondi, and M. Tagliasacchi, "Briskola: Brisk optimized for low-power arm architectures," in *Image Processing (ICIP), 2014 IEEE International Conference on*, Oct 2014, pp. 5691–5695.

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

[28] T. Potsch, K. Kuladinithi, M. Becker, P. Trenkamp, and C. Goerg, "Performance evaluation of coap using rpl and lpl in tinyos," in *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, May 2012, pp. 1–5.

[29] E. d'Angelo, A. Alahi, and P. Vandergheynst, "Beyond bits: Reconstructing images from local binary descriptors," in *Pattern Recognition (ICPR), 2012 21st International Conference on*, Nov 2012, pp. 935–938.

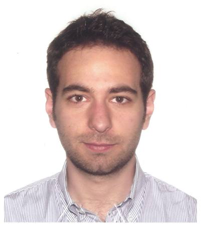
[30] A. Canclini, M. Cesana, A. Redondi, M. Tagliasacchi, J. Ascenso, and R. Cilla, "Evaluation of low-complexity visual feature detectors and descriptors," in *Digital Signal Processing (DSP), 2013 18th International Conference on*, July 2013, pp. 1–7.

[31] P. Weinzaepfel, H. Jegou, and P. Pérez, "Reconstructing an image from its local descriptors," in *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*, 2011, pp. 337–344. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2011.5995616

[32] L. Sanchez, L. Munoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, and D. Pfisterer, "Smartsantander: Iot experimentation over a smart city testbed," *Computer Networks*, vol. 61, no. 0, pp. 217 – 238, 2014, special issue on Future Internet Testbeds - Part I.

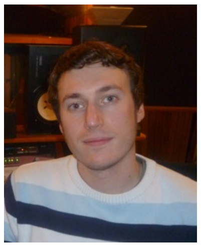
[33] P. R. de Almeida, L. S. Oliveira, A. S. B. Jr., E. J. S. Jr., and A. L. Koerich, "Pklot - a robust dataset for parking lot classification," *Expert Systems with Applications*, vol. 42, no. 11, pp. 4937 – 4949, 2015.

Luca Bondi received the MS degree in Computer Engineering in December 2014 from Politecnico di Milano. He is currently pursuing a PhD degree at the Department of Electronics, Informatics and Bioengineering, Politecnico di Milano. His research activities are focused on deep neural networks for efficient visual features extraction and compression.



Luca Baroffio received the M.Sc. degree (2012, cum laude) in Computer Engineering from Politecnico di Milano, Milan, Italy. He is currently pursuing the Ph.D. degree in Information Technology at the "Dipartimento di Elettronica e Informazione - Politecnico di Milano", Italy. In 2013 he was visiting scholar at "Instituto de Telecomunicações", Lisbon, Portugal. His research interests are in the areas of multimedia signal processing and visual sensor networks.

Matteo Cesana is currently an Assistant Professor with the Dipartimento di Elettronica, Informazione e Bioingegneria of the Politecnico di Milano, Italy. He received his MS degree in Telecommunications Engineering and his Ph.D. degree in Information Engineering from Politecnico di Milano in July 2000 and in September 2004, respectively. From September 2002 to March 2003 he was a visiting researcher at the Computer Science Department of the University of California in Los Angeles (UCLA). His research activities are in the field of design, optimization and performance evaluation of wireless networks with a specific focus on wireless sensor networks and cognitive radio networks. Dr. Cesana is an Associate Editor of the Ad Hoc Networks Journal (Elsevier).



Alessandro Redondi received the MS in Computer Engineering in July 2009 and the Ph.D. in Information Engineering in 2014, both from Politecnico di Milano. From September 2012 to April 2013 was a visiting student at the EEE Department of the University College of London (UCL). His research activities are focused on algorithms and protocols for Visual Sensor Networks.

Marco Tagliasacchi is currently Assistant Professor at the "Dipartimento di Elettronica e Informazione - Politecnico di Milano", Italy. He received the "Laurea" degree (2002, cum Laude) in Computer Engineering and the Ph.D. in Electrical Engineering and Computer Science (2006), both from Politecnico di Milano. He was visiting academic at the Imperial College London (2012) and visiting scholar at the University of California, Berkeley (2004).

His research interests include multimedia forensics, multimedia communications (visual sensor networks, coding, quality assessment) and information retrieval. Dr. Tagliasacchi co-authored more than 120 papers in international journals and conferences, including award winning papers at MMSP 2013, MMSP2012, ICIP 2011, MMSP 2009 and QoMex 2009. He has been actively involved in several EU-funded research projects. He is currently co-coordinating two ICT-FP7 FET-Open projects (GreenEyes - www.greeneyesproject.eu, REWIND - www.rewindproject.eu).