Towards commoditized smart–camera design

Boštjan Murovec*, Janez Perš, Rok Mandeljc, Vildana Sulić Kenk, Stanislav Kovačič

University of Ljubljana, Faculty of Electrical Engineering, Tržaška cesta 25, SI–1000 Ljubljana, Slovenia

Abstract

We propose a set of design principles for a cost–effective embedded smart camera. Our aim is to alleviate the shortcomings of the existing designs, such as excessive reliance on battery power and wireless networking, over–emphasized focus on specific use cases, and use of specialized technologies. In our opinion, these shortcomings prevent widespread commercialization and adoption of embedded smart cameras, especially in the context of visual–sensor networks. The proposed principles lead to a distinctively different design, which relies on commoditized, standardized and widely–available components, tools and knowledge. As an example of using these principles in practice, we present a smart camera, which is inexpensive, easy to build and support, capable of high–speed communication and enables rapid transfer of computer–vision algorithms to the embedded world.

Keywords: commoditized smart camera, general–purpose smart camera,

*Corresponding author

Email addresses: `bostjan.murovec@fe.uni-lj.si` (Boštjan Murovec), `janez.pers@fe.uni-lj.si` (Janez Perš), `rok.mandeljc@fe.uni-lj.si` (Rok Mandeljc), `vildana.sulic@fe.uni-lj.si` (Vildana Sulić Kenk), `stanislav.kovacic@fe.uni-lj.si` (Stanislav Kovačič)

design principles, reference design, visual–sensor networks

1. Introduction

Capabilities of embedded processors have increased remarkably in recent years. Consequently, we have witnessed a migration of many tasks, previously considered as processing–intensive, to the domain of embedded systems. An example is computer vision, which is increasingly moving from the once–prevalent domain of desktop and industrial computers to the embedded devices — *embedded smart cameras* — and is the driving force behind the development of visual–sensor networks (VSNs) [1]. Especially in VSNs, *embedded vision* faces many challenges [2, 1, 3], due to severe limitations in computing performance, memory capacity and communication bandwidth of VSN nodes. Additionally, for VSNs to be a viable and economical alternative to centralized computer systems, nodes have to be inexpensive and easy–to–maintain. These requirements have resulted in numerous attempts to build an inexpensive yet sufficiently powerful *general–purpose* embedded smart camera, which would be able to run various computer vision algorithms.

Given the fair number of proposed designs, it is somewhat surprising that a *general–purpose* embedded smart camera, based on an open architecture is difficult to find, and even more difficult to buy. Furthermore, no solutions are available at a price range that would justify large–scale VSN deployments with several dozens or even hundreds of nodes, collaborating on large, distributed and complex tasks. As shown by surveys of the field, e.g. [4, 3, 5], there is certainly no shortage of proposed smart camera architectures, which all reached the stage of a working prototype. However, we feel that the absence of large–scale commercialization and deployment hints at more deeply rooted weaknesses in existing general–purpose smart camera architectures. We also feel that the absence of commercialization of *general–purpose smart cameras* negatively affects the deployment of large–scale VSNs and therefore warrants a special attention.

This paper aims to identify shortcomings of the existing designs. We feel that the issues raised here significantly affect chances for widespread commercial deployment of general–purpose smart cameras, especially in the context of visual sensor networks. Therefore we also present a smart camera design that, while by itself not being at the bleeding edge of technology, does address many of the architectural weaknesses discussed in the paper. As such, it should serve as an example of how the proposed principles may be applied in practice, even when inexpensive hardware is used. We do not claim that our proposition outperforms other designs; we merely claim that we took care to address every aspect of the design in accordance with the proposed principles, providing best possible environment for embedded computer vision application design, *given the hardware constraints.*

The remainder of the paper is structured as follows: in Section 2 we present a systematic overview of embedded camera designs and highlight their strengths and weaknesses. Next, we describe the proposed set of design principles in Section 3, followed by the example of their application in Section 4. In Section 5 we present a proof–of–concept camera design, which follows our principles, along with the experimental evaluation. We conclude the paper with Section 6.

2. Related work

A brief overview of embedded smart–camera technology is given in [6]; the prevailing concept appears to be close integration of a CMOS visual sensor, a microcontroller (MCU) and the supporting electronics [7], sometimes to the point of integrating them all in a single integrated circuit, as illustrated in [8]. There is a wide variety of components that can be used in embedded smart cameras, which result in different compromises between image resolution, computing power, connectivity and power requirements. However, in this paper, we focus on design principles behind a *general–purpose smart camera*. Therefore, when comparing our work to state of the art, we attributed the highest importance to the *programmer's view* of the camera. As such, we divide the published camera designs into the following four categories.

- **Low–end.** Cameras from this group are, in general, built around basic, sometimes 8–bit, MCUs, and can process images at low framerate and/or at low resolution. These cameras support only basic computer vision algorithms, such as thresholding, simple blob extraction and simple color segmentation. Consequently, their power requirements are usually low and the hardware is inexpensive, but on the other hand, they provide very little flexibility in terms of programming — the code has to be written (and perhaps optimized) for each platform separately. Often, such cameras lack even sufficient amount of memory to store the whole image at once, which promptly disqualifies them for being capable of running anything but algorithms that require only single pass over the image.

- **Mid–range.** Cameras from this group can employ a wide variety of MCUs, but they do not run a fully fledged operating system. The two main characteristics of cameras in this range are the availability of a standard–compliant C compiler for the chosen MCU platform and sufficient amount of RAM to store the whole image at a chosen resolution and bit depth. Both characteristics enable a straightforward implementation of widely–used computer–vision algorithms for the camera architecture. On the other hand, even though such designs are more flexible, in the absence of an operating system, the task of such camera remains more or less fixed once it is programmed.

- **High–end.** Cameras from this group run an operating system (frequently a variant of Linux), offering high degree of flexibility. A standard C/C++ compiler is taken for granted, as is the number of standard software libraries like OpenCV [9] for computer vision, libjpeg for JPEG image compression/decompression, and similar. Given the proper connectivity, these cameras can be maintained and upgraded remotely, even to the point of completely changing their task (completely different computer–vision algorithm) by simply instructing the camera to run a different version of executable code.

- **Heterogenous and special designs.** Cameras from this group are highly specialized for their task, and their hardware architecture reflects that. As such, they cannot be considered *general–purpose*. From the programmer's point of view, they may be more similar to low–end designs regarding their inflexibility since the code has to be written from scratch to take advantage of their capabilities, but their raw computing performance at the task they have been developed for can be on a par with or significantly better than the high–end designs.

2.1. Low–end cameras

There is a significant group of designs that are based on low–end MCUs. In [10], authors present 30 frames–per–second color tracking, whereas in [11], a tracking system with 7.5 frames–per–second is presented; both are based on an 8–bit AVR MCU and use image resolution of 176×144 pixels. Both designs have insufficient memory to store even a single full frame, and the code was highly optimized to process image data in a single pass in order to make even basic tracking feasible.

In [12], the problem was approached differently: a system–on–chip solution was used, combining 8–bit AVR MCU with an FPGA module, and external SRAM module is used to store acquired images. To take full advantage of the architecture, processing has to be split between the FPGA and MCU, requiring highly specialized and optimized code.

eCam [13] is an example of a low–end embedded camera design that is specialized to the extreme; it consists only of image sensor, a specialized controller chip (OV528 serial bridge with JPEG compression capabilities) and communication interface (wireless sensor node). Its only function is streaming of JPEG–compressed video to a base station.

2.2. Mid–range cameras

At the the lower end of this category, we find designs like a mote, described in [14]. It uses 30×30 pixel optical mouse sensor and a dsPIC microcontroller with only 16 kB of RAM, but is nevertheless able to perform Viola–Jones face detection [15], gradient–based edge detection, motion estimation, and background removal using two convex filters with slow and fast forgetting parameters to avoid ghosting [16]. An ANSI C compiler is available for the platform, and given the small resolution of image sensor, the amount of memory is sufficient to store multiple acquired images. This way, computer–vision algorithms can be adapted for use on this platform without excessive optimization that is customary for low–end devices.

Cyclops sensor, presented in [17], is built around 8–bit Atmel ATmega128L MCU, with RAM expanded to 64 kB and using the image sensor with maximum resolution of 352×288 pixels. In [18], a mote for wireless sensor networks is presented, built around an ARM7–based MCU with up to 64 kB of RAM, and two 30×30 pixel image sensors. An ARM9–based camera is presented in [19], processing 160×120 pixel images. With integrated energy harvester (solar) module and a PIR sensor, the system is optimized for the task of person detection, under the assumption of low–duty–cycle operation, thus lowering power consumption. RoboVision platform in [20] comprises ARM9–based MCU and 96 kB of RAM, but is coupled with disproportionately high–resolution image sensor (1600×1200 pixels). Authors themselves note that reasonable image sizes for onboard processing are actually much smaller (160×120 pixels).

Finally, one of the most prominent embedded camera designs is the CMUCam line, especially CMUcam3 [21]. Its ARM7–based MCU and 64 kB of RAM are insufficient to process captured images at the maximum resolution and bit depth offered by the included image sensor (352×288 pixels, RGB).

However, by halving the resolution and processing only grayscale images, it can run state–of–the–art algorithms, such as Viola–Jones face detection [15]. A version of GCC–based toolchain, adapted to the camera architecture, is provided.

2.3. High–end cameras

For computationally more demanding tasks, significantly more powerful computing platforms are used, such as a RISC–processor–based design with 16 MB of RAM, presented in [22]. It is used for real–time car counting, during which it processes images at the resolution of 320×240 pixels at 30 frames per second. Developers of the system explicitly aimed at a platform without dedicated hardware. Even more powerful CITRIC platform [23] uses Intel XScale PXA270 processor clocked at up to 624 MHz, combined with 64 MB of RAM and a 1280×1024 pixel image sensor; the platform runs embedded Linux. Panoptes [24] relies heavily on standard, commoditized hardware: Intel StrongArm 206 MHz processor (running Linux) is coupled with a USB web camera (resolutions up to 640×480 pixels) and IEEE 802.11 (WiFi) wireless network interface.

2.4. Heterogenous and special designs

To cope with relatively high demands of real–time image processing, some designs use specialized hardware to deliver required computing power. The system presented in [25] relies on a DSP for real–time video compression, using optimized libraries provided by the DSP vendor. However, when running plain C++ code, the system's performance suffers significantly.

There are many examples of similar systems: [26] and [27] rely on DSPs as well, and [28] relies on specialized multimedia processors. Both [29] and [30] rely on SIMD processors, requiring specialized programming knowledge to exploit their full potential. On the other hand, the processing module is not the only component that can be customized to optimize certain performance goals: in [31], authors opt for a dual–sensor architecture to decrease power consumption of an otherwise more powerful 32–bit microcontroller; they employ low–resolution imaging sensor for basic detection, and capture image of higher resolution only when needed. Similar path is chosen by authors of the dual–camera sensor, presented in [32].

It is obvious that the computer–vision algorithms need to be adapted to take advantage of such multi–sensor setups, even if designs are based on otherwise standardized architectures, and therefore are rightly considered specialty designs.

2.5. Summary

Generally, high–end cameras employ more powerful hardware components. Consequently, their cost and power requirements are higher as well, but they are able to run more advanced computer–vision algorithms. However, there are some designs that stand out, for example [14], which has exceptionally modest hardware specifications, but is versatile and able to run even state–of–the–art algorithms, albeit under obvious hardware constraints. It is our opinion that such combination reflects a well–thought–out design that provides high value to the user (programmer, software developer).

In this paper, we strive to formalize the design principles that would have a similar effect: cost–effective, but maximally versatile general–purpose embedded camera designs, which would be appealing to a wider user base and not restricted to a single project or a single application. Finally, with regard to the previously–described criteria, the camera that we present in this paper as an practical example of applying the proposed design principles, falls into the mid–range category.

3. Proposed design principles

We believe that there may be several design issues that hinder the wider adoption of general–purpose smart cameras, especially in the context of visual sensor networks. Those issues stem from certain design principles that may be reasonable in general embedded system design, however, their justification in the context of smart cameras is not entirely straightforward.

3.1. Standardization and commoditization

The reason behind the rapid development in many areas of computer technology lies in widespread standardization, which allows unhindered competition between many different vendors, and inevitably leads to commoditization of the new technology. A classical example is the personal computer, which, after being designed and initially marketed by IBM, became extremely commoditized, with its clones produced and sold by numerous vendors.

The second historically important aspect of such development is a possible inferiority of the winning product, which is again illustrated by the dominance of the IBM–PC clones. At the time of its introduction, the PC was an office computer with inferior graphics, no sound, and modest processing capabilities. However, the effect of standardization was powerful enough to overcome deficiencies and allowed rapid development into a far more superior product.

At the moment, the field of smart cameras is experiencing a complete absence of both paradigms, which contributes to slow adoption of embedded visual systems and practically no major deployment of large–scale VSNs. Such state of affairs is not surprising, as commoditization in short run benefits the consumers of the technology and not its manufacturers. However, in the long run, a widespread adoption of a particular technology often increases manufacturer's production volume despite their lower market share.

Our proposal. Without the move to standardized and commoditized components, it is unlikely that smart cameras and VSNs will ever become ubiquitous. Furthermore, to spur the innovation in the field, especially the transfer of state–of–the–art computer vision algorithms to the embedded domain, and boost the popularity of smart cameras as much as possible, the designs should be suitable for developers of various backgrounds and funding capacities, from industrial teams to hobbyists.

In addition to being cost–effective, designs should extensively rely on parts that can be easily purchased in small quantities, and allow assembly of prototypes without highly specialized and expensive machinery. For example, the components should be available both in SMD and DIP variation of housing; the former is important for serial manufacturing, whereas the latter is breadboard–friendly and enables quick and ad–hoc development.

As demonstrated by successful start–ups on a daily basis, the momentum behind individual developers should not be underestimated. In our opinion, such adoption is critical in evolution of smart cameras, as it significantly rises the probability of *killer applications*, which are needed for the field of smart cameras to stay competitive.

3.2. Long–term stability

The long–term fate of most designs is difficult to predict, since the rush for specialized technologies usually results in use of the best and the hottest parts that are available on the market at the instant of a project kickoff; many such parts tend to be discontinued fairly frequently. As an example, the renown line of CMUcam smart cameras[1] so far consists of four members, with the first three already discontinued.

Frequent discontinuities and replacements pose a serious barrier to a wide smart–camera adoption. Without considerable support from the camera designers, computer–vision developers are forced to redesign their software in order to cope with the changes introduced by new models. Such redesigns are difficult to justify due to engineering costs for re–achieving something that has already worked well, and there is always a risk of introducing regressions. Despite the stated difficulties, a low–cost smart camera with a long–term support is yet to appear.

Our proposal. There certainly exist smart camera applications that benefit from the use of the latest technology; this is especially true in cases where cameras perform relatively standard tasks, such as real–time video compression. Nevertheless, general–purpose smart cameras could definitely benefit from the shift of focus from the bleeding–edge technology towards mature and time–proven components, especially if they are to be deployed in tasks

[1] http://cmucam.org

that will require many years of support and servicing.

3.3. Generality

The CMUcam family is also an illustrative example of products that are non–generalized by design. Models CMUcam1[2] and CMUcam2[3] were specifically intended to be used as trackers, which was also reflected by their firmware. Although it was possible to reprogram the whole camera to change its capabilities, this was not their intended use, and such practice was not officially supported. A further step in this direction has been done with CMUcam4, where firmware is not even programmed in C or other widespread programming language, but in a "C–like" language called SPIN[4]. Porting existing computer–vision code to such environment is definitely not a trivial task.

In contrast, the third member of the family, CMUcam3 [21], was fully–programmable in standard C programming language; this model has become extremely popular among computer–vision developers who required embedded capability, which clearly demonstrates the need for general–purpose designs. We expect that discontinuation of CMUcam3 will adversely affect many research groups working with embedded vision.

Our proposal. The design of embedded smart cameras should be as generic as possible. Instead of offering firmware with a pre–built functionality, the expectation should be that camera's software will be developed almost from scratch by *application* developers, not *camera* developers. The

[2]http://cmucam.org/projects/cmucam1
[3]http://cmucam.org/projects/cmucam2
[4]http://www.cmucam.org/projects/cmucam4/wiki/Firmware

latter should provide libraries and APIs for hardware–independent image acquisition, image debugging and communication, preferably as C source code that links with application's code, but should not impose any mandatory boilerplate design of camera's firmware.

3.4. Specialized technology

Many designs rely on technologies that are too specialized, such as DSP processors and FPGA circuits. Although these offer an unparalleled boost of cameras' capabilities, very few computer–vision developers can actually use them efficiently. This is directly tied to the available knowledge– and code–base in the computer–vision community, which has at its access a large pile of a carefully crafted C and Matlab code that uses only generic processing hardware.

Our proposal. In contrast to the current practice, development of smart cameras needs to be decoupled from development of applications that run on them, since the two require different expertise. Computer–vision development requires a dedicated computer–vision specialist. If such an engineer needs to master additional specialty areas and hardware–specific skills, application development becomes impractically demanding, since a whole–team worth of expertise is required to develop even a simple practical solution. In this context, truly open nature of an embedded camera becomes extremely important. Cameras should be able to run standard computer–vision C code nearly out of the box, and offer a simple C application programming interface (API) for acquiring images directly into memory buffers specified by developer. Different approaches almost inevitably confine camera's operation to one or at most a few applications that its own developers are willing to implement.

3.5. Power supply

There appears to be consensus that embedded smart cameras need to be as energy–efficient as possible in order to enable battery–powered applications, and the majority of designs nominally strive to achieve this goal. However, as shown in Table 1, most of the proposed solutions cannot operate on battery power for long periods of time, especially when their power requirements in the fully–operational state is considered.

Design	Power [mW]	Autonomy [days]
[21] CMUcam3	650	0.57
[29] Xetal	600	0.6
[12] (WSN node)	500	1.33
[23] CITRIC	1000	1.5$^{\clubsuit}$
[13] eCAM	231	1.6
[32] (dual–camera)	150$^{\diamond}$	2.5
[30] IC3D	100	3.75
[18] (WSN mote)	99–363	1–3.8
[19] (with PIR sensor)	50–650$^{\diamond}$	7.5–0.58
[17] Cyclops	23–65†	5.7–16†
[31] MeshEye	12‡	31‡

† rich power options to adapt to actual needs

‡ average; very low duty–cycle regime of operation

$^{\diamond}$ low duty–cycle regime, wakeup by a secondary sensor

$^{\clubsuit}$ authors' own estimation

Table 1: Power consumption of some of the designs when processing data (contrary to sleep mode) — exceptions are described in the footnotes. Where possible the consumption without communication was taken into account. Autonomy is based on 9 Wh capacity, which is an equivalent of two AA alkaline batteries without any additional energy source.

A general–purpose smart camera by definition cannot be designed with only one particular application in mind, therefore power calculations should be based on its fully–operational state. Designs that conserve power by, for example, resorting to extremely low frame rates or additional sensors for wakeup, such as [31, 19, 32], thus allowing a camera to conserve power by frequently shutting down major parts of its architecture, should be more appropriately referred to as a low–duty–cycle *application* instead of a low–power *design*.

Therefore, it seems that the ubiquitous goal of designing a general–purpose battery–powered smart camera cannot be achieved using the current technology. Even the extremely permissive scenarios with intermittent operation (as envisioned in [31]) require monthly battery changes. In a large network, this may be uneconomical in terms of battery and labor costs, unless there is absolutely no alternative. For a comparison, one should consider the similar task of changing the burned–out light bulbs: the rush to substitute incandescent light bulbs with alternatives was motivated in part by a desire to reduce the labor costs, associated with frequent bulb changes. Frequent battery changes may be impractical due to service interruption, and environmentally harmful due to a pile of discarded batteries. Finally, even though battery–powered designs seem attractive as a solution for a variety of applications, there is rarely an absolute need to use battery power, at least in urban environments – not to mention the indoor installations. In those, the economics of the recurring cost of battery maintenance, compared to the initially higher but fixed cost of wiring up the cameras, may quickly become questionable.

Autonomy in the environments, where there is absolutely no possibility of externally supplied power, can be extended using the solar power, such as described in [19, 33], but this requires reserve capacity for the periods when there is not enough solar radiation.

Our proposal. While there undoubtedly exist applications where battery–powered operation is unavoidable, there certainly exist many more where wired power works just as well. There are inherent advantages to the wired power supply — mainly in relaxing other, often prohibitive, constraints, as we demonstrate later on. Therefore, while the research into energy efficiency of smart cameras remains a noble and worthwhile goal, there is no need to restrict camera architectures solely to battery power, even in the case of visual–sensor networks. To cope with broad range of situations, a truly open and general–purpose camera architecture should support both power options — the battery power and the wired power.

3.6. Communication

Another common design goal is wireless communication. In theory, battery–powered and wirelessly–communicating nodes enable rapid deployment of VSNs. In practice, the push for battery power necessitates resorting to low–power communications, based on the IEEE 802.15.4 standard (ZigBee, 6LoWPAN), which have limited bandwidth and range. For example, in [30], 5 m is described as a practical operating distance; therefore, a square area of 400 m^2 requires sixteen nodes just to overcome the limitations in communication range, with associated sixteen battery packs to be changed on a regular basis.

Our proposal. As with the power–supply issue, there certainly exist applications where wired communication is completely acceptable — even at the price of a more complicated initial setup. In this case, the long–term benefits of the wired solution are even more obvious, as wired communication enables significantly higher bit rates than low–power wireless solutions. As we show later on, when a node is designed with wired power in mind, it is trivial to extend its power–supply lines with additional high–speed communication lines at little extra cost. Additionally, wired communication is less exposed to opportunistic hacking attacks as physical access is needed to tap into wired communication lines.

When power lines are already in place but installation of communication lines is unfeasible, the wired power supply may be efficiently supplemented with a high–bandwidth wireless communication of a decent range (IEEE 802.11 WiFi). Of course, for nodes that *must* communicate wirelessly and rely on battery power, low–power wireless communication is more or less the only option.

When considering a truly open implementation of an embedded smart camera, incompatible communication options may be difficult to overcome. Many standard solutions, such as ZigBee, 6LoWPAN, WiFi and BlueTooth exist in the form of modules that can be connected to standard SPI, I^2C or UART interfaces on MCUs; however, even in such cases, hardware from different vendors likely requires different, vendor–specific, code paths in camera's software.

In contrast, there are alternative solutions that do not require such explicit level of adaptation. This is especially true when standard UART communication interfaces are used with less complex, time proven, communication options, such as RS–232 and RS–485, or a truly widespread and standard interface, such as Ethernet. Consequently, communication tasks can be abstracted away to the point where they do not represent an obstacle to a widespread adoption anymore. Our proof–of–concept implementation shows that this is indeed possible.

3.7. Image resolution

Existing embedded–camera designs frequently exhibit a noticeable discrepancy between the resolution of imaging sensor and capabilities of the associated MCU, and especially the amount of RAM available for image storage and processing. Sensors often deliver color images of VGA or higher resolutions, whereas storing a whole such image requires 1 MB of RAM or more, which is well beyond the capacity of the low–end MCUs.

Our proposal. Many state–of–the–art computer–vision algorithms do not rely on color information at all, and therefore a monochrome sensor is completely sufficient. Typical RAM capacities from 32 kB to 128 kB correspond to monochrome images of resolutions from 50×50 to about 250×100, which should allow simultaneous storage of multiple images during processing. Ideally, there should be support for image acquisition with adjustable resolution, to enable online balancing between the available memory and desired visual details.

Even though, intuitively, low resolutions seem insufficient for any computer vision application, the contrary is successfully demonstrated by designs like [14, 18] (30×30–pixel applications). Another example is CMUCam1 with the image resolution of 80×143 pixels [20]. Finally, if the resolution is too high for the MCU that processes the data (to the point that the whole image cannot be buffered in RAM), the camera is effectively downgraded from the mid–range design to the low–end one, since the majority of ready–made computer vision algorithms are prevented from running on such images.

3.8. Image acquisition

The main task of general–purpose smart cameras — and general–purpose VSN nodes — is reliable image processing and extraction of information required by downstream users (machines or people). Such concept is not far from the field of machine vision, even if the setup does not operate in an industrial environment. Consequently, the application of machine–vision guidelines [34] may be beneficial for many potential applications.

One of the main guidelines in machine vision is the use of artificial illumination for scene normalization, which is mostly avoided in smart–camera designs due to power consumption of illuminators. The situation is contradictory, since the lack of scene normalization requires more CPU–intensive algorithms for image processing with adequately higher power consumption. Another important lesson from the field of machine vision is that the optics should be adapted to the problem at hand, thus making the best use of the available image resolution.

Our proposal. We advise that the machine–vision guidelines are applied whenever possible. The wired power, if feasible, is a game–changer in this aspect; it on one hand enables use of illumination, which simplifies computation, and on the other hand allows use of a more powerful CPU than in the case of a low–power design. Use of appropriate and possibly exchangeable lens (in contrast to cameras with integrated lens, such as those usually built into mobile phones) allows adaptation of optical–system parameters to a specific application. This includes, but is not limited to, compensating for the previously–suggested lower image resolution by optimizing camera's field of view. Furthermore, optical filters may be used to increase camera's relative sensitivity to a particular wavelength (typically wavelength of camera's own illuminator), thus reducing the interference from uncontrolled light sources, such as sunlight.

3.9. Image debugging

Development of computer–vision applications differs from the usual embedded programming in at least one important aspect. During debugging, the computer vision algorithm developer is not only interested in the plain numerical values of MCU's registers and memory locations, but also benefits from being able to display raw or partially–processed images stored in MCU's memory. Many smart–camera designs place no emphasis on this functionality, thus making embedded computer–vision development more challenging than necessary.

Our proposal. Although image visualization is a higher–level concept than inspection of memory locations, it is possible to make image debugging semi–transparent by providing software interface for transferring image buffers from the smart camera to the host PC, where they can be displayed. While such solutions requires sufficient communication capabilities, they need to be implemented only in the development version of the smart camera. We demonstrate the concept of image debugging by piping live image buffers from our proof–of–concept smart camera to the host PC and displaying them in Matlab.

4. Implementation of the proposed principles

This section outlines possible implementations that reflect the previously discussed design principles. Following our classification of embedded–camera designs in Section 1, we focus on the low–end and mid–range category, as these represent significantly bigger challenge in that respect. Our recommendations are summarized in Table 2.

4.1. Microcontroller and development toolchain

The choice of MCU depends heavily on its integrated peripherals for CCIR image acquisition (Sections 4.4 and 4.5) and communication interfaces.

Aspect	Viable solutions
Microcontroller	Min. 64 kB RAM, 256 kB FLASH, two UART/serial modules. High–speed host USB module (only for UVC video). 32–bit CPU core with GCC–based or other standard C toolchain. C++ is desired but non–mandatory.
Power supply	A pair of exclusive switching–mode voltage regulators for wired and battery power (for each desired output voltage).
Communication	RS–485 for wired communication. Arbitrary wireless module with standard serial interface. Software abstraction for network independent API.
Imaging sensor	Analog CCIR (for monochrome low–resolution imaging) or USB camera supporting the UVC standard.
Video digitizer (for analog video)	Microcontroller's built–in A/D converter capable of at least 1 MSamples/s, together with other required periphery.
Optics	M12 interchangeable lens. Focal length depending on application.
Illumination	NIR LED diodes with optional diffusor and NIR filter.

Table 2: Standardized and commoditized technology for imaging and processing aspects of embedded smart–camera designs.

Multiple serial/UART interfaces are necessary to implement a variety of standard communication options, such as RS–232 and RS–485 (Section 4.3). The majority of 6LoWPAN, ZigBee and BlueTooth modules and some Ethernet interfaces can also be connected to serial interfaces.

MCU should have enough RAM to store all simultaneously–needed images (Section 3.7) and other variables. In our view, 64 kB of RAM is a minimum, but 128 kB is much better, especially if Matlab Coder[5] is used to translate Matlab code to the C language.

Since image processing is computationally intensive task, a MCU based on a 32–bit CPU should be chosen; generally, it is less prudent to use 16–bit or lesser CPU architectures, especially if power supply is not an issue. In addition, MCUs that provide sufficiently advanced peripherals usually come with 32–bit CPUs anyway, and wider buses also enable faster access to RAM and peripherals. Hardware support for floating–point arithmetic is a bonus, but not mandatory for many tasks.

CPU should be supported by a GCC[6]–based development toolchain to facilitate straightforward porting of an existing computer vision code; both native–C code, as well as code generated by Matlab Coder. Availability of a C++ toolchain, which is less common in the embedded world, makes it possible to use code based on OpenCV [9].

4.2. Multiple power–supply options

Notwithstanding the dilemma between the wired and battery power, it is beneficial, if the camera can be powered from a variety of power sources. In

[5] http://www.mathworks.com/products/matlab–coder/
[6] http://gcc.gnu.org

addition to being flexible per se (allowing installation in variety of environments), such feature comes handy when autonomy is being extended using rechargeable batteries, solar power (as for example in [33] and [19]) or powered from other unreliable sources, like car battery during engine cranking.

In the case of a wired power, the minimal upper end of input voltage range should be 12 V, which is used by many illuminators. Furthermore, camera operation on wider voltage ranges is easily achievable, which is needed for seamless integration into various environments. For example, MAX5033[7] voltage regulator supports input voltages between 7.5 V and 76 V; this includes 42 V, which is emerging as the new standard for car power supply.

4.3. Communication solutions

In contrast to frequently used low–power wireless solutions that mostly benefit battery–powered nodes, there exist excellent wired alternatives. One of them is the time–honored RS–485 standard, which is used in various industrial and consumer setups, but appears to be completely overlooked by embedded smart–camera engineers. RS–485 has been devised for industrial environments that require robustness to electromagnetic interference, distances up to 1.2 km and bandwidths up to 10 Mbit/s.

For example, members of MAX485 family[8] offer bandwidth of 2.5 Mbit/s and bus topology with up to 128 nodes at the price of $3–5 in small quantities; this is the price of a bare ZigBee transceiver integrated circuit, and one third of a price of a full ZigBee transceiver module. Several members of

[7] http://datasheets.maxim–ic.com/en/ds/MAX5033.pdf
[8] http://datasheets.maxim–ic.com/en/ds/MAX1487–MAX491.pdf

MAX308x family[9] provide bandwidth of 10 Mbit/s and bus topology with up to 256 nodes for nearly the same price[10]. Both models exist in DIP and SMD variants. Transceivers attach to standard UART interfaces. Connection between RS–485 devices consists of a three–wire bus, where one of the wires is a ground wire, and may be shared with power supply lines. In total, four wires are enough to provide both power supply and RS–485 communication to the node.

In our tests, we easily achieved bit rate of 3.5 Mbit/s over 125 m of unshielded three–wire mains cable using MAX–485 transceiver that is declared for maximal throughput of 2.5.Mbit/s[11]. Although we certainly do not suggest to use the transceiver beyond its specifications, the experiment demonstrates that industry–certified and expensive RS–485 cables are not necessary for embedded–camera setups, which makes RS–485 an extremely attractive low–cost, high–range communication solution.

4.4. Imaging sensor

Due to popularity of digital–cameras and smartphones, many digital imaging sensors are commercially available; however, each has a different, non–standard, communication protocol. Such sensors are readily connectable to MCUs, but if a sensor is withdrawn from the market, the embedded design becomes obsolete. In addition, only high–volume customers can easily obtain

[9] http://datasheets.maxim–ic.com/en/ds/MAX3080–MAX3089.pdf

[10] The choice of bandwidth is more influenced by slew–rate limitation, EMI emission and length of connection than price.

[11] Transmission–reception of 3 GB of data without a single bit error. Configuration consisted of one transmitter and one receiver. Practically achievable bandwidth degrades by increasing number of nodes but experiment nevertheless demonstrates a huge safety margin by exceeding the official bandwidth limits by 40%.

such sensors, whereas smaller groups often find such purchases challenging. To the best of our knowledge, there are only two widespread standards for imaging sensors that are not subject to the stated concerns: the time–honored analog video[12], and the USB Video Class (UVC) specification[13].

UVC cameras deliver images in digital format, and may thus seem more suitable for embedded designs. However, although not required by the UVC standard, most of available UVC cameras mandate high–speed (480 Mbit/s) USB hosts, whereas many viable MCUs with built–in USB support can only cope with full–speed (12 Mbit/s) USB connectivity. In addition, UVC cameras generally deliver color images of high resolutions that require too much RAM for storage and processing (see Section 3.7). Consequently, we see UVC cameras as viable imaging sensors only in combination with high–end MCUs that come with sufficient amount of RAM and high–speed USB support.

Considering all this, analog cameras present a viable imaging option. A monochrome (CCIR) version is sufficient for many practical purposes (Section 3.7). CCIR cameras of a size of a coin cost around $6, and can be obtained both in DIP or SMD housing.

4.5. Video digitizer

CCIR image of a sufficient quality for many computer–vision applications can be acquired solely using MCU's internal peripherals, without any active external circuits, such as analog filters and amplifiers. The required peripherals are an A/D converter and a voltage comparator with a voltage reference for detection of sync pulses. An A/D with sampling rate of 1 Msample/s

[12] http://pdfserv.maxim–ic.com/en/an/AN734.pdf
[13] http://www.usb.org/developers/devclass_docs/USB_Video_Class_1_1.zip

allows acquisition at horizontal resolution of 50 pixels, whereas with faster sampling it is possible to extend resolution up to the sensor's physical limit.

Two adjacent interlaced video frames may be combined to double the horizontal resolution at the same sampling rate, although the method is suitable only for quasi–static images due to characteristic blurring that occurs with fast moving objects. The full CCIR vertical resolution of 288 pixels is achievable regardless of the A/D sampling rate. The stated capabilities match the guidelines of Section 3.7.

Figure 1 shows two images acquired using the presented approach; resolutions are 50×50 and 100×250 — the lower and the upper end of resolutions recommended in Section 3.7.

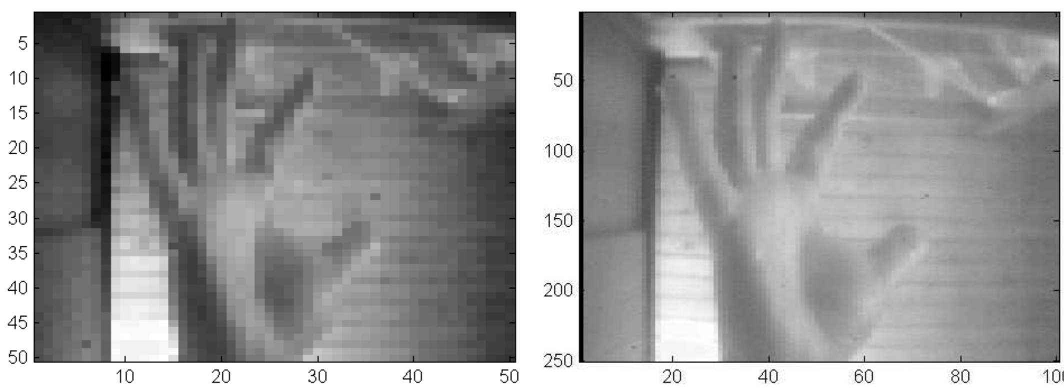


Figure 1: Acquired images of a human hand without use of illuminators (left: 50×50 pixels, right: 100×250 pixels).

4.6. Optics, illumination and filters

According to the guidelines in Section 3.8, we recommend equipping cameras with exchangeable M12–type lens, which are a standard for low–cost (board) cameras. Several models cost around $5 and fit the previously–

mentioned low–cost CCIR cameras. When the system is subjected to unwanted visible light, NIR LED illuminators in combination with visible–light blocking NIR filter aid in scene normalization. Figure 2 demonstrates benefits of this approach, which makes segmentation of an object of interest (a hand) much easier.

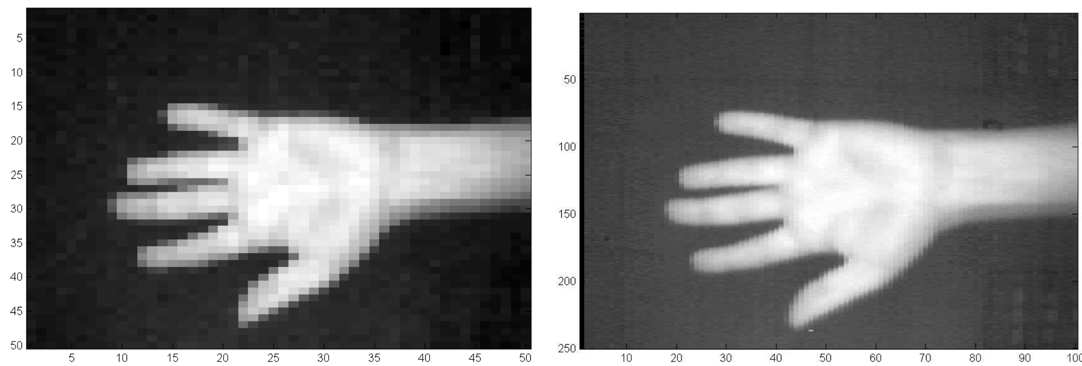


Figure 2: Scene normalization with illumination. Separation of object of interest (a hand) from background becomes much easier compared to images in Figure 1, which were acquired using the same optics and at same resolution (left: 50×50 pixels, right: 100×250 pixels).

5. Proof–of–concept embedded smart camera

We illustrate the presented design principles on a proof–of–concept low–cost embedded smart camera, which can be used as a standalone entity or in role of a VSN node. Figure 3 presents the conceptual scheme.

5.1. Hardware

The MCU of our choice (U1 in Figure 3) is a high–end member of Microchip's PIC32 family[14]. It comprises 32–bit MIPS CPU, 512 kB of FLASH,

[14] http://www.microchip.com/pagehandler/en–us/family/32bit/

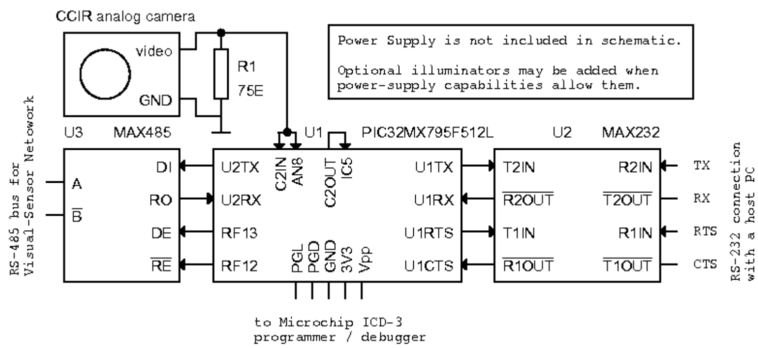


Figure 3: A proof–of–concept embedded smart camera.

128 kB of RAM, 10–bit A/D converter with 1 Msample/s, two voltage comparators, a settable voltage reference, six UART modules, four SPI modules and five I^2C modules. Microchip offers a free graphical integrated development environment MPLAB–X[15] with assembler and C toolchain. The chosen MCU is available only in SMD housing, but it can still be used on breadboard with an aid of Sparkfun adapter[16].

The preference for Microchip over other families and vendors is influenced by long–term stability of their products, especially in contrast to ARM processors, where different models are introduced and withdrawn fairly quickly. Also, the zero–price entry barrier for almost fully–featured development environment is an important factor of consideration.

From a hardware standpoint, a plain MCU can be converted into a smart camera simply by adding a CCIR camera and resistor R1 (Figure 3), therefore enabling the capture of up to 50×250 pixel images at 25 frames per second, and up to 100×250 pixel images by combining two interlaced video frames.

[15]http://www.microchip.com/pagehandler/en–us/family/mplabx/
[16]http://www.sparkfun.com/products/9713

The other two integrated circuits in Figure 3 add connectivity options to the mote; part U2 takes care of RS–232 connection with the host PC[17] for image debugging and exchange of processing results, while part U3 drives RS–485 bus network for VSN connectivity[18]. When wired power is used, it is possible to use the RS–485 ground wire as power ground, therefore four wires are enough to provide both power supply and connectivity to visual–sensor network.

Our design requires extremely little specialized skills and can be built even by hobbyists, which lowers the bar for low–volume applications. Total price for the parts in Figure 3 together with omitted voltage regulators, lens and illuminators, but without the PCB and housing, is about $40 for purchases in small quantities.

5.2. Software

Applications are developed in the standard C programming language. Support for standard C code enables reuse of large code–base that exists in the computer–vision community. The camera itself does not force any boilerplate code or application skeleton for proper operation. In addition, Matlab code can be straightforwardly ported to the camera by compiling into C code using Matlab Coder. Our tests indicate that Microchip's toolchain successfully compiles the resulting ports of Matlab code.

For decoupling application and camera development, we developed a soft-

[17]Depending on the actual RS–232 driver, a couple of additional discrete components not shown in the scheme may be needed.

[18]RS–485 bus must be terminated at both ends with 120 Ω resistor. When nodes' power supplies are floating against each other, an RS–485 ground wire must be added to the bus, and ground point of each node must be connected to the bus ground through 1 kΩ resistor.

ware library that offers platform– and imaging–sensor–independent API for image acquisition. It is integrated into an application purely in form of source code, without any binary objects. It offers C interface for image acquisition into arbitrary RAM locations. The image resolution is specified at each acquisition; it can be chosen from the resolutions that are available as per Section 4.5. The library also handles acquisition of images with a prescribed time delay, thus relieving the application developers from having to deal with timers. Furthermore, it takes care of video signal sampling and operates as a set of interrupt routines that run in the background, thus making CPU available for execution of application's code even while the image acquisition is in progress.

5.3. Illustrative examples

A simple test of image–processing capabilities was done using a motion detector that acquires two 50×50 images with a prescribed time delay. For each grabbed image, the following intermediate results are derived:

- image of vertical first–order derivatives (1.22 ms),
- image of horizontal first–order derivatives (1.21 ms),
- gradient image; per–element integer squared root of summed squares of vertical and horizontal derivative (20.02 ms),
- blurred image; obtained from gradient image using 5×5 averaging filter, together with scaling so that only additions are performed (2.58 ms).

The total execution time of the above steps is 25.03 ms; the timings are for code compiled without optimization, which demonstrates the level of performance offered by the free version of the toolchain.

The final image is produced by thresholding the absolute differences between two consecutive blurred images (1.60 ms). Motion is detected if the sum of pixels in this image (0.59 ms) exceeds another prescribed threshold. Please note that this is not a state–of–the–art motion detector but rather a test–bed for timing common image–processing operations. According to the results, the camera is capable of running the described detector at full 25 frames–per–second, and still have slightly more than 12 ms (30 %) of spare time during each frame.

Computing variance of the resulting image (duration 31.94 ms) is also an instructive measure of performance as it involves floating point arithmetic. The chosen MCU does not offer hardware floating–point support, which makes the computation rather slow. The last performance test involving the motion detector is computing histogram of the resulting image (duration 0.97 ms) and associated entropy using a look–up table together with a proper scaling that enables operation in the integer domain (duration 0.13 ms). For illustration, the code optimization that comes with the licensed version of the toolchain reduces image–processing steps from 25 ms to 8.1 ms, whereas variance computation time drops from 32 ms to 17.7 ms.

The RAM capacity of the camera allows storage of all intermediate results, which makes it possible to transfer and examine them on the host PC using the image–debugging facilities, as shown in Figure 4. Furthermore, we have implemented two–way debugging facilities, which enable two images to be uploaded from the host PC for processing on the camera. This way, algorithms can be tested on a prescribed set of images (e.g., a standard dataset) in the actual working environment instead on an emulator.

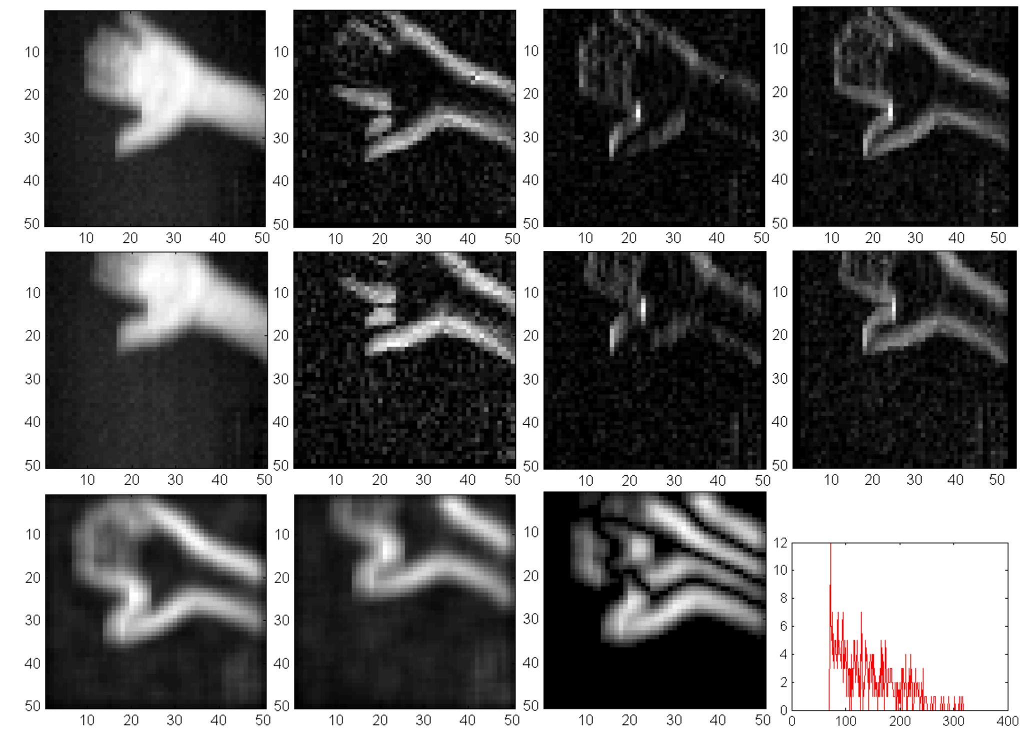


Figure 4: Image debugging of the motion detector. First row, from left to right: the first input image, vertical first–order derivatives, horizontal first order derivatives, gradient image. Second row: the second input image and its intermediate results. Third row: blurred variations of the first and second gradient image, thresholded difference of blurred images, histogram of the last image (with omitted frequency of pixels with zero value).

5.4. Additional tests

To illustrate the abilities of our proof–of–concept design, we ported several well–known algorithms to the developed platform and measured their performance in terms of processing time per frame. We also ran the same tests on a range of hardware platforms to establish how they compare to our smart camera.

5.4.1. People tracker

With only minimal modifications we successfully ported a basic people–tracking application that was originally developed in the C programming language under Linux. The algorithm is based on sequential image differencing and blob tracking, with the Munkres assignment algorithm [35] as the final tracking stage. The code on the camera runs at 20–25 frames–per–second at resolution 50×50.

5.4.2. Object–recognition scheme and people detector

We also implemented a simple object–recognition scheme, using either HOG descriptor [36] or region covariance descriptor [37]. For each frame, it performs a descriptor extraction and calculates its distance to the reference descriptor. The recognition runs with 5.6 and 7.4 frames–per–second, respectively (at resolution of 50×50 pixels). Finally, we run the combination of the HOG descriptor and the linear SVM classifier, trained off–line as a people detector (training was performed in advance on a PC). This test was done at the resolution of 50×100 pixels – which is supported by the camera as well. The test ran at approximately 2.5 frames–per–second. The detailed timings are provided in the next section (Table 3).

Matlab code for covariance descriptor and the corresponding distance is presented in Figure 5. It was ported to our platform using Matlab Coder. Note that the code includes the computation of generalized eigenvalues, a task that is seldom implemented on MCUs. We feel that the possibility of running code that was developed in Matlab truly constitutes the Holy Grail of the embedded computer vision, and illustrates the versatility of the proposed approach.

```matlab
% covariance descriptor for the
% central (32x32) region of a
% 50x50 pixel, 8-bit image.
function C = cov_descriptor (I)

 % Copy, crop, convert
 If = single(I(8:41,8:41));

 % Convolution masks
 f1 = [-1 0 1];
 f2 = [-1 2 -1];

 % Derivatives
 Ix = conv2(f1, If);
 Iy = conv2(f1,1,If);
 Ixx = conv2(f2, If);
 Iyy = conv2(f2,1,If);

 % Features
 If = If(2:33,2:33);
 If = If(:);
 coordinates = 1:32;
 X = repmat(coordinates,32,1);
 Y = repmat(coordinates',1,32);
 X = X(:);
 Y = Y(:);
 Ix = Ix(2:33,3:34);
 Ix = Ix(:);
 Iy = Iy(3:34,2:33);
 Iy = Iy(:);
 Ixx = Ixx(2:33,3:34);
 Ixx = Ixx(:);
 Iyy = Iyy(3:34,2:33);
 Iyy = Iyy(:);
 F = [If, X, Y, Ix, Iy, Ixx, Iyy];

 % Descriptor
 C = cov(F);

% Distance between the two
% covariance descriptors
function D = distance (C1, C2)
 D = sqrt(sum(log(eig(C1,C2)).^2));
```

Figure 5: Matlab code for computing the region covariance descriptor and generalized–eigenvalues distance. This code is directly portable to our camera using Matlab Coder.

5.5. Comparison to widely–used hardware platforms

In addition to testing our camera design, we ran the same battery of tests on the following hardware platforms:

- **Axis 207W**, an ARM9TDMI–based IP camera running Linux (discontinued).
- **Axis P1346**, a HD IP camera with a CRIS ARTPEC–3 CPU. This and the previous camera are commercial products, manufactured by Axis Communications. They run Linux, and for licensing reasons, the development toolchains are provided for both. We tested the cameras without the video clients connected, therefore, their CPU load before the test was negligible.
- **Raspberry Pi**, a single–board computer, intended as a tool for teaching computer science, powered by ARM11–based ARM1176JZF–S processor at 700 MHz, running Linux.
- **A high-end PC** with Intel Core i7 950 CPU at 3.07 GHz, running Linux.

In all cases, GCC compiler for each platform was used to compile the same C code, with full optimization enabled, except for the covariance test on Axis 207W, where the optimization produced broken code. In all cases, the code utilized only a CPU; on PC, only a single core of a quad–core CPU was used. Table 3 presents a comparison of results between all the tested architectures. Note that we do not present the detailed results of the tracker test, as the performance varies significantly with the number of objects detected. In all cases, the camera draws 160 mA of current at 12 V without the infrared LED illuminator. This value includes the energy cost of voltage conversion. The whole setup consumes 360 mA with the illuminator turned on. (The mote without the illuminator can be powered from 5 V with an equal current consumption.)

Platform	Covariance [ms]	HOG [ms]	HOG+SVM [ms]
Axis 207W	2025†	1845	3848
Axis P1346	262	261	574
Raspberry Pi	2.71	3.48	7.48
Intel PC	0.09	0.14	0.44
Our camera	135	180	395

† optimization disabled

Table 3: Comparison of the running times for the three algorithms on the five tested architectures. The values represent the average time needed for the actual computation. Covariance and HOG tests were run at 50×50 pixel resolution, while the people detector (HOG+SVM) was run at 50×100 pixels.

5.6. Exemplary communication abstraction

Section 3.6 mentions the idea of isolating core application's code from communication details. To verify the feasibility of the approach, we developed second smart–camera prototype by connecting a CCIR camera to Microchip's PIC32 Ethernet Starter Kit[19], which consists of the same PIC32 MCU combined with all hardware required for establishing an Ethernet connection. In conjunction with Microchip's TCP/IP stack[20], such a node quickly becomes TCP/IP compliant.

[19]http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2615&dDocName=en545713
[20]http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2505&param=en535724

On both prototypes, a simple server application (written in C) services requests from the host–PC client (written in Matlab), such as acquiring images at different resolutions, reading and writing of image buffers, sending configuration information, etc. For both RS–232 and Ethernet, a set of routines for sending and receiving data, data availability testing, buffer–full state indication, etc., was developed using the same function prototypes. For each connection type, we prepared a C header file containing preprocessor macros that translate into appropriate function calls.

The end result are two camera prototypes that, in spite of having two radically different connection types, share the same application code. This shows that isolation of embedded smart–camera applications from details of communication options and hardware implementations is indeed both possible and beneficial.

6. Conclusion

In this paper, we identified several issues in the field of embedded smart cameras, that in our opinion hinder widespread adoption of these devices. Our research was partially motivated by our background in computer vision; to us, a general–purpose embedded smart camera is a vehicle for unobtrusive and gradual introduction of computer–vision technology into everyday use. Therefore, we are especially concerned with issues such as long–term design stability, commoditization, generalization, rapid application development and code reusability.

We certainly do not suggest that widely–discussed issues such as low power and wireless communications do not require any attention from the embedded camera community. However, we feel that there are significant but overlooked opportunities for embedded camera designers to use proven and well–established technologies to deploy computer vision and smart cameras into widespread use. To illustrate our point, we presented our camera design, which does not further state–of–the–art technology–wise, but on the other hand possesses many properties we would like to see in a modern, leading–edge smart camera. It is flexible in a sense that it allows the application of machine vision principles to solutions based on such camera. It allows reasonably fast communications, thus it can be used as a part of a larger camera network. It enables computer–vision engineers to reuse a large body of code developed for other platforms, and it allows computer–vision scientists to quickly develop new algorithms using widely–used engineering tools, such as Matlab. Finally, since it is made up of relatively standard or widely used components, it should allow solution providers to deploy and sell products based on such camera with relatively low risk of quick obsolescence. We are not proposing our camera as a reference design for embedded smart cameras, but we hope that the principles outlined in this paper gain wider adoption in the embedded smart camera community and find their way into next generation of cameras.

References

[1] S. Soro, W. Heinzelman, A Survey of Visual Sensor Networks, Advances in Multimedia 2009 (2009) 21 pages.

[2] Y. Charfi, N. Wakamiya, M. Murata, Challenging Issues in Visual Sensor Networks, IEEE Wireless Communications 16 (2009) 44–49.

[3] I. T. Almalkawi, M. Guerrero Zapata, J. N. Al-Karaki, J. Morillo-Pozo, Wireless multimedia sensor networks: Current trends and future directions, Sensors 10 (2010) 6662–6717.

[4] B. Tavli, K. Bicakci, R. Zilan, J. M. Barcelo-Ordinas, A survey of visual sensor network platforms, Multimedia Tools and Applications (2011) 1–38.

[5] B. Rinner, T. Winkler, W. Schriebl, M. Quaritsch, W. Wolf, The evolution from single to pervasive smart cameras, in: Second ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC 2008), pp. 1–10.

[6] Z. Zivkovic, R. Kleihorst, Smart Cameras for Wireless Camera Networks: Architecture Overview, in: H. Aghajan, A. Cavallaro (Eds.), Multi–Camera Networks, Academic Press, 2009, pp. 497–510.

[7] B. Rinner, W. Wolf, An Introduction to Distributed Smart Cameras, Proceedings of the IEEE 96 (2008) 1565–1575.

[8] W. Caarls, P. P. Jonker, H. Corporaal, SmartCam: Devices for Embedded Intelligent Cameras, in: PROGRESS 2002, Proceedings of 3rd Seminar on Embedded Systems.

[9] G. Bradski, The OpenCV Library, Dr. Dobbs Journal of Software Tools, 2000.

[10] H.-N. Nguyen, A.-C. Lee, Real Time Tracking Multiple YUV 24–Bit Color Objects with 8–Bit MCU–Based Embedded Vision System, in:

Proceedings of the Fourth International Conference on Innovative Computing, Information and Control (ICICIC'09), pp. 160–164.

[11] A. Aggarwal, Embedded Vision System, in: Proceedings of IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications (MESA'08), pp. 618–621.

[12] A. Kerhet, M. Magno, F. Leonardi, A. Boni, L. Benini, A low-power wireless video sensor node for distributed object detection, Journal of Real-Time Image Processing 2 (2007) 331 – 342.

[13] C. Park, P. H. Chou, ecam: ultra compact, high data-rate wireless sensor node with a miniature camera, in: Proceedings of the 4th international conference on Embedded networked sensor systems, SenSys '06, pp. 359–360.

[14] M. Camilli, R. Kleihorst, Demo: Mouse Sensor Networks, the Smart Camera, in: The Fifth ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC'11), pp. 1–3.

[15] M. Viola, M. J. Jones, P. Viola, Fast Multi–View Face Detection, in: Proceedings of Computer Vision and Pattern Recognition (CVPR'03).

[16] M. Cristani, M. Farenzena, D. Bloisi, V. Murino, Background Subtraction for Automated Multisensor Surveillance: A Comprehensive Review, EURASIP Journal on Advances in Signal Processing 2010 (2010) 24 pages.

[17] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin,

M. Srivastava, Cyclops: In Situ Image Sensing and Interpretation in Wireless Sensor Networks, in: SenSys, ACM Press, 2005, pp. 192–204.

[18] I. Downes, L. B. Rad, H. Aghajan, Development of a mote for wireless image sensor networks, in: Proc. of COGnitive systems with Interactive Sensors (COGIS).

[19] M. Magno, D. Brunelli, P. Zappi, L. Benini, A solar-powered video sensor node for energy efficient multimodal surveillance, in: 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD '08), pp. 512–519.

[20] V. Rana, M. Matteucci, D. Caltabiano, R. Sannino, A. Bonarini, Low cost smartcams design, in: IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTImedia 2008), pp. 27–32.

[21] A. Rowe, A. Goode, D. Goel, I. Nourbakhsh, CMUcam3: An Open Programmable Embedded Vision Sensor, Technical Report CMU–RI–TR–07–13, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2007.

[22] M.-Y. Chiu, R. Depommier, T. Spindler, An Embedded Real–time Vision System for 24–hour Indoor/Outdoor Car–Counting Applications, volume 3, IEEE Computer Society, 2004, pp. 338–341.

[23] P. Chen, P. Ahammad, C. Boyer, S.-I. Huang, L. Lin, E. Lobaton, M. Meingast, S. Oh, S. Wang, P. Yan, A. Yang, C. Yeo, L.-C. Chang, J. D. Tygar, S. Sastry, Citric: A low-bandwidth wireless camera net-

work platform, in: Second ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC 2008), pp. 1–10.

[24] W.-C. Feng, E. Kaiser, W.-C. Feng, M. Le Baillif, Panoptes: Scalable low-power video sensor networking technologies, in: In MULTIMEDIA 03: Proceedings of the eleventh ACM international conference on Multimedia, ACM Press, 2003, pp. 562–571.

[25] M. Bramberger, R. P. Pflugfelder, A. Maier, B. Rinner, B. Strobl, H. Schwabach, A smart camera for traffic surveillance, in: In Proceedings of the First Workshop on Intelligent Solutions in Embedded Systems, pp. 153–164.

[26] H. Andrian, K.-T. Song, Embedded CMOS Imaging System for Real–Time Robotic Vision, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005), pp. 1096–1101.

[27] M. Bramberger, R. P. Pflugfelder, A. Maier, B. Rinner, B. Strobl, H. Schwabach, A Smart Camera for Traffic Surveillance, in: Proceedings of the First Workshop on Intelligent Solutions in Embedded Systems (WISES), pp. 153–164.

[28] T. Loten, R. Green, Embedded Computer Vision Framework on a Multimedia Processor, in: Proceedings of 23rd International Conference on Image and Vision Computing New Zealand (IVCNZ'08), pp. 1–5.

[29] R. Kleihorst, A. Abbo, B. Schueler, A. Danilin, Camera Mote with a High-Performance Parallel Processor for Real–Time Frame-based Video

Processing, in: First ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC'07), pp. 109–116.

[30] R. Kleihorst, B. Schueler, A. Danilin, M. Heijligers, Smart Camera Mote with High Performance Vision System, in: ACM SenSys Workshop on Distributed Smart Cameras.

[31] S. Hengstler, H. Aghajan, A Smart Camera Mote Architecture for Distributed Intelligent Surveillance, in: ACM SenSys Workshop on Distributed Smart Cameras.

[32] D. Xie, T. Yan, D. Ganesan, A. Hanson, Design and implementation of a dual-camera wireless sensor network for object retrieval, in: International Conference on Information Processing in Sensor Networks (IPSN '08), pp. 469–480.

[33] T. Celik, H. Kusetogullari, Solar-powered automated road surveillance system for speed violation detection, IEEE Transactions on Industrial Electronics 57 (2010) 3216–3227.

[34] B. G. Batchelor, P. F. Whelan, Machine Vision Systems: Proverbs, Principles, Prejudices & Priorities, in: In Proceedings of Applied Machine Vision Conference, pp. 7–19.

[35] H. W. Kuhn, The Hungarian Method for the Assignment Problem, in: Naval Research Logistics Quarterly, volume 2(1–2), pp. 83–97.

[36] N. Dalal, B. Triggs, Histograms of Oriented Gradients for Human Detection, in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), volume 1, pp. 886–893.

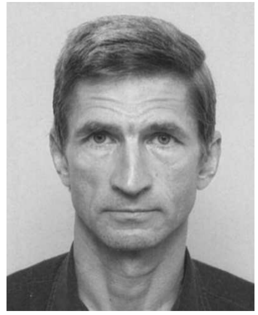
[37] O. Tuzel, F. Porikli, P. Meer, Region Covariance: A Fast Descriptor for Detection and Classification, in: European Conference on Computer Vision (ECCV), volume 7.

Boštjan Murovec received B.Sc., M.Sc. and Ph.D. degrees in Electrical engineering at the Faculty of Electrical Engineering, University of Ljubljana, in 1996, 1999 and 2002, respectively. He is currently an assistant professor at the Machine Vision Laboratory at the Faculty of Electrical Engineering, University of Ljubljana. His research interests lie in image processing, pattern recognition, embedded design, combinatorial optimization and sequence analysis.

Janez Perš received B.Sc., M.Sc. and Ph.D. degrees in Electrical engineering at the Faculty of Electrical Engineering, University of Ljubljana, in 1998, 2001 and 2004, respectively. He is currently an assistant professor at the Machine Vision Laboratory at the Faculty of Electrical Engineering, University of Ljubljana. His research interests lie in image–sequence processing, object tracking, human–motion analysis, dynamic–motion–based biometry, and in autonomous and distributed systems.

Rok Mandeljc received his B.Sc. degree in electrical engineering from the Faculty of Electrical Engineering, University of Ljubljana, Slovenia, in 2010. He is currently pursuing the Ph.D. degree as Junior Researcher at Machine Vision Laboratory at the Faculty of Electrical Engineering, University of Ljubljana. His research interests include computer vision, image processing and data fusion in context of multi–view and multi–modal person localization and tracking.

Vildana Sulić Kenk received her B.Sc. and Ph.D. degrees from the Faculty of Electrical Engineering of the University of Ljubljana, Slovenia in 2006 and 2011, respectively. Currently, she is a researcher at the Machine Vision Laboratory at the Faculty of Electrical Engineering, with her interests in computer vision, image processing, human–motion analysis and visual–sensor networks.



Stanislav Kovačič is a professor, a vice dean and the head of the Laboratory for Machine Vision at the Faculty of Electrical Engineering, University of Ljubljana. He was a visiting researcher in the GRASP Laboratory at the University of Pennsylvania, the Technische Fakultät der Friedrich–Alexander–Universität in Erlangen, and at the Faculty of Electrical Engineering and Computing, University of Zagreb. His research is focused on various aspects of image and video analysis with applications in medicine, industry and sports.