Low cost wireless image sensor networks for visual surveillance and intrusion detection applications

Congduc Pham
University of Pau, LIUPPA Laboratory
Email: congduc.pham@univ-pau.fr

***Abstract*—Wireless Image Sensor Networks (WISN) where sensor nodes are equipped with miniaturized visual CMOS cameras to provide visual information is a promising technology for situation-awareness, search&rescue applications or intrusion detection applications. In this paper, we present an off-the-shelf image sensor based on an Arduino Due board with a CMOS uCamII camera. The image sensor works with raw 128x128 image and implements a simple intrusion detection mechanism based on simple-differencing technique. The total time for getting the image, encoding it, detecting intrusion and transmitting it can be less than 2.3s. We detail the performance measures and using real hardware constraints, we also show how more accurate simulation model can be built for large-scale multi-hop networked configurations.**

***Index Terms*—Low-cost image sensors, wireless image sensor networks, simulation model**

I. Introduction

Wireless Image Sensor Networks (WISN) where sensor nodes are equipped with miniaturized visual CMOS cameras to provide visual information is a promising technology for situation-awareness, search&rescue applications or intrusion detection applications. Image can be requested on demand or in a pre-defined order in situation-awareness applications, but can also be sent upon intrusion detection in security applications. Obviously, given the low level of resources (memory and processor) of sensor nodes and the low bandwidth of the networking technologies traditionally used in wireless sensor networks (i.e. IEEE 802.15.4 for a radio throughput of 250kbps), implementing an image sensor with fast but still efficient image compression and multi-hop transmission features is still challenging.

There are a number of image sensor boards available or proposed by the very active research community on image and visual sensors: Cyclops [1], MeshEyes citemesheyes, Citric [2], WiCa [3], SeedEyes [4], Eye-RIS [5], Panoptes [6], CMUcam3&FireFly [7], [8], CMUcam4 and CMUcam5/PIXY [8], iMote2/IMB400 [9], ArduCam [10],... All these platforms and/or products are very good but are mostly based on ad-hoc development of the visual part (i.e. development of a camera board with dedicated micro-controller to perform a number of processing tasks) or are based on very powerful micro-controller/Linux-based platforms or do not have an efficient image encoding and compression scheme adapted to wireless sensor networks. Our motivations in building our own image sensor platform for research on image sensor surveillance applications are:

1) have an off-the-shelf solution so that anybody can reproduce the hardware and software components: we use an Arduino-based solution for maximum flexibility and simplicity in programming and design; we use a simple, affordable external camera to get RAW image data,
2) integrate and apply a fast and efficient compression scheme with the host micro-controller (no additional nor dedicated micro-controller) to produce robust and very small size image data suitable for large scale surveillance or search&rescue/situation awareness applications.

The image sensor that we propose works with raw 128x128 image in 8-bit/pixel gray scale. The raw image size is 16384 bytes and can be compressed with various quality factor to greatly decrease the image size for real-time multi-hop transmission. A simple intrusion detection mechanism based on simple-differencing technique shows very good results while adding no cost in the image processing. The total time for getting the image, encoding it, detecting intrusion and transmitting it can be less than 2.3s with a reasonable image quality. Using the real performance measures for various steps of the image processing, we also build more accurate wireless image sensor simulation model to study large-scale configurations. The results of the simulations are then very close to those obtained from the real platform.

The rest of the article is organized as follows. Section II describes the image sensor components: hardware components and image encoding technique. Section III presents the performance measures of the image sensor platform and Section IV will present the intrusion detection mechanism. Section V will show how an accurate simulation model can be built to study large-scale configurations. We conclude in Section VI

II. A low cost image sensor with off-the-shelves components

A. The image sensor components

We use an Arduino Due board [11] with a CMOS uCamII camera (56^o angle of view) from 4D systems [12]. The Arduino Due is a micro-controller board based on the Atmel SAM3X8E ARM Cortex-M3 running at 84MHz with 96KB of SRAM memory. The uCam is connected to the Arduino board through an UART interface at 115200 bauds. The uCamII is capable of providing both raw and JPEG format but we are not using this last feature as JPEG images have very low robustness level against packet losses. We use the raw

128x128 in 8-bit/pixel gray scale format which need 16384 bytes of memory for storage. Radio is provided by an XBee S1 IEEE 802.15.4 module. The XBee module is also connected to the Arduino with an UART but at 125000 bauds as the communication between the XBee module and the Due is not reliable at 115200 bauds given the Due's clock frequency of 84MHz [13]. Fig. 1 shows the image sensor node.

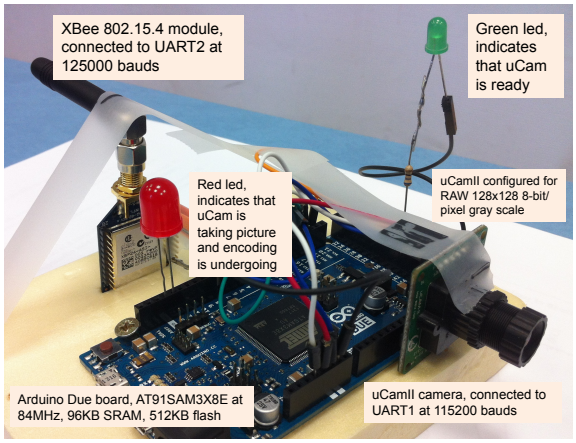

Fig. 1. Image sensor built with Arduino Due and uCAM camera

B. The image encoding process

Even if with low-power radio, such as IEEE 802.15.4, the MAC layer retransmission feature provides quite efficient and low-overhead reliability, early studies have confirmed that image communication needs to be especially tolerant to packet losses which automatically make traditional JPEG compression scheme unsuitable as it suffers from very high spatial correlation: an entire image could be impossible to decode with only a few packets missing. We therefore use an optimized encoding scheme proposed in [14] which features the 2 following key points:

1) Image compression must be carried out by independent block coding in order to ensure that data packets correctly received at the sink are always decodable.
2) De-correlation of neighboring blocks must be performed prior to packet transmission by appropriate interleaving methods in order to ensure that error concealment algorithms can be efficiently processed on the received data.

The compression scheme is a JPEG-like coder and operates on 8x8 pixel blocks with advanced optimizations on data computation to keep the computational overhead low. The combination of the fast JPEG-like proposed encoder with an optimized block interleaving method [15] allows for an efficient tuning, the so-called Quality Factor (Q), of the compression ratio/energy consumption trade-off while maintaining an acceptable visual quality in case of packet loss. The code has been ported to Arduino with little modifications. Fig. 2 shows a 128x128 image taken with the image sensor and encoded with various quality factor, from 100 down to 5. The total size of the compressed image, the number of generated packets and the PSNR compared to the original image are shown. We can see that a quality factor of 20 is visually still acceptable while providing a compression ratio of almost 12!

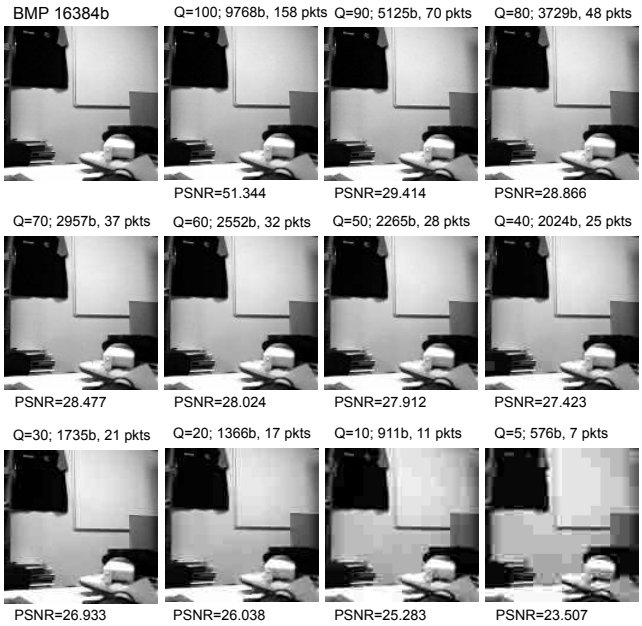

Fig. 2. 128x128 image taken by the image sensor, various quality factor

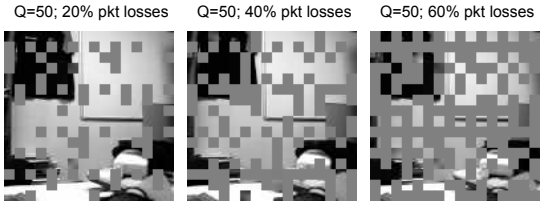

Fig. 3. Impact of packet losses on image quality

Fig. 3 shows the impact of packet losses on the image quality: 20%, 40% and 60% of packet losses. Traditional JPEG compression can hardly support more than 10% packet losses.

The encoding stage takes as input the raw image (16384 bytes). It produces for each pixel an integer variable coded as a short data type (2 bytes) on the Arduino Due. We therefore need an 32768-byte array. In total, in order to store the image data from the camera and to run the encoding process, the image sensor needs at least 48KB of RAM memory.

The packetization stage after the encoding can use various packet size. We set the maximum image payload per packet to 90 bytes (this is the maximum image payload, in practice, the produced packet size will vary according to the packetization process) because 6 bytes need to be reserved in the 802.15.4 payload for framing bytes (2B), quality factor (1B), real packet size (1B) and offset in the image (2B).

III. Performance of the Image Sensor Platform

A. Global measures

We present in this section the performance measures of the various processing stages run on the image sensor. Fig.

4 presents as a function of the quality factor Q the measured "global encode+pkt time" in column A. This is the overhead of the image encoding process including the encoding itself and the packetization stage, but without transmission.

		N	R	A	B = D - A`	C = B / N	D	E = D / N
Quality Factor Q	size in bytes (MSS=90)	Number of packets	time to read data from ucam	global encode + pkt time (measured)	global transmit time (computed)	transmit time/pkt (computed)	global encode + pkt + transmit time (measured)	encode+transmit time/pkt (in ms)
100	9768	158	1.512	1.027	1.064	0.0067	2.091	13.2342
90	5125	70	1.512	0.782	0.539	0.0077	1.321	18.8714
80	3729	48	1.512	0.704	0.384	0.0080	1.088	22.6667
70	2957	37	1.512	0.686	0.304	0.0082	0.99	26.7568
60	2552	32	1.512	0.662	0.263	0.0082	0.925	28.9063
50	2265	28	1.512	0.646	0.233	0.0083	0.879	31.3929
40	2024	25	1.512	0.657	0.207	0.0083	0.864	34.5600
30	1735	21	1.512	0.649	0.177	0.0084	0.826	39.3333
20	1366	17	1.512	0.638	0.14	0.0082	0.778	45.7647
10	911	11	1.512	0.628	0.093	0.0085	0.721	65.5455
5	576	7	1.512	0.624	0.058	0.0083	0.682	97.4286

Fig. 4. Global encoding, packetization and transmission time

The time to read the raw image data from the uCam is also shown in column R and R+A represents the latency between the snapshot taken by the camera and the time all the packets of the encoded image are produced (once again without transmission). If we take into account the transmission overhead, column D shows the "global encode+pkt+transmit time". The packetization and the transmission tasks are performed in a row for each packet. Values in column A and column D have been globally measured and can be used to derive column B which represents the time taken globally for transmitting the produced packets: more packets means higher transmission time. Fig. 5 shows the part of the encoding+packetization (column A) and the part of the transmission (column B). Stacking both parts gives column D.

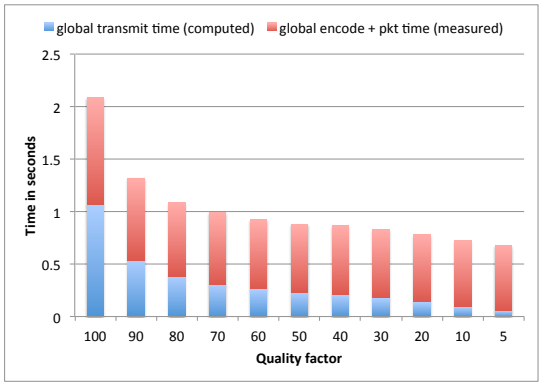

Fig. 5. Encoding time and transmission time

Column C shows the computed mean time for transmitting a packet which is quite constant. The time to read data from uCam is also constant and actually does not depend much on the uCam-Arduino connection baud rate (here 115200 bauds) because the limitation is mainly due to memory read operations from the Arduino UART ring buffer. We increased the connection baud rate to 921600 bauds and did not see any improvement in the data read time. If we use a quality factor of 20, the total time between the snapshot taken by the uCam and the end of the transmission of the image is 1.512+0.778=2.29s.

B. Detailed measures

Fig. 6 shows the detailed timing of the encoding stage and the packetization stage, measured alone, without the glue code that would make the whole image processing tasks to run. We can see that the encoding time, shown in column F, is quite constant, except for Q=100. With the maximum quality factor value, the encoding stage has little to do, which explains the smaller encoding time. Column G shows the time taken to produce each packet. Once again, this time is quite constant and has been rounded to 3ms/packet.

	F	G	H = F + G * N	I = H + C * N	J	K = (C + G)*N
Quality Factor Q	encode time (measured)	packetization (pkt) time (measured & rounded)	encode + pkt time (computed)	encode + pkt + transmit time (computed)	RCV time (measured)	RCV time (computed)
100	0.379	0.0030	0.8530	1.917	1.708	1.538
90	0.512	0.0030	0.7220	1.261	0.799	0.749
80	0.511	0.0030	0.6550	1.039	0.599	0.528
70	0.519	0.0030	0.6300	0.934	0.447	0.415
60	0.509	0.0030	0.6050	0.868	0.39	0.359
50	0.500	0.0030	0.5840	0.817	0.349	0.317
40	0.516	0.0030	0.5910	0.798	0.317	0.282
30	0.516	0.0030	0.5790	0.756	0.278	0.24
20	0.518	0.0030	0.5690	0.709	0.231	0.191
10	0.516	0.0030	0.5490	0.642	0.177	0.126
5	0.518	0.0030	0.5390	0.597	0.131	0.079

Fig. 6. Detailed timing

Column H shows the computed encoding and packetization time using only values of column F and G. Compared to column A, we can see the part of the glue code. Adding column H to the previously measured transmission time per packet (column C), we have in column I a computed value of the global encode+pkt+transmit time. Column I can be compared to column D.

C. Received latency

In column J, we show the receive time measured for a 1-hop scenario at a sink which will decode and display the image. The receive time represents the amount of time between the first packet received and the last packet received. In column K, we computed the receive time by using formula K=(C+G)*N which basically sums up the mean time for sending a packet (column C) and the packetization time required between each packet (column G) times the number of packets (column N). Fig. 7 plots the values of columns J (blue curve) and K (red curve); we can see that the 2 curves are very close.

Fig. 7 also shows the 1-hop image display latency that represents the amount of time between the snapshot at the source image sensor and the display of the image at the sink 1-hop away (column R+D+J). The smaller the latency, the more responsive is the system. We will discuss the multi-hop issues later on in Section V. Since the time to transmit a packet is not very large, we can actually see that having high value for Q is not very penalizing in term of receive latency.

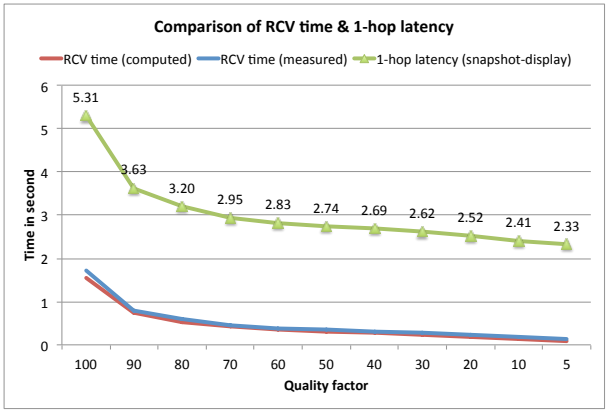

Fig. 7. Image received time and 1-hop image display latency

D. Impact of a slower micro-controller

The encoding time and the transmission time may depend on the micro-controller type and speed. Regarding the transmission time, we use a traffic generator to measure the minimum time spent in the send function and the minimum time between 2 sends as the payload size is varied. 100 packets are sent and the mean is computed. We compared the Arduino Due which features an Atmel SAM3X8E ARM Cortex-M3 running at 84MHz to an Arduino MEGA built around an ATmega2560 running at 16MHz. The code of the traffic generator, as well as the communication library for the XBee radio module, is exactly the same on both platforms. Fig. 8 shows the comparison. We can notice that the Arduino MEGA has smaller send time while the time between 2 sends is higher. The smaller send time may be explained by the simpler architecture of the Arduino MEGA. The higher time between 2 sends is due to the higher processing time of the program control and communication code. However, we can see that actually the impact of a slower micro-controller on the sending overheads is very small.

Payload size in bytes	mean between send (Arduino Due), ms	mean send (Arduino Due), ms	mean between send (Arduino MEGA), ms	mean send (Arduino MEGA), ms
10	1.99	1.97	2.09	1.92
20	2.79	2.76	2.95	2.59
30	3.68	3.46	3.92	3.41
40	4.4	4.36	4.63	4.29
50	5.2	5.16	5.51	5.11
60	6.02	5.97	6.26	5.97
70	6.95	6.74	7.17	6.61
80	7.61	7.57	7.93	7.51
90	8.37	8.36	8.72	8.26
100	9.2	9.18	9.61	9.07

Fig. 8. Comparison of sending performance on Due and MEGA boards

For the encoding time, we run the image encoding code on the Arduino MEGA but since the MEGA board does not have enough memory to realize the image encoding and packetization on an entire real image from the uCAM, we only use a small part of the real image data and modified the encoding code to only use a fraction of the real data. However, the entire encoding and packetization computations are performed. Fig. 9 plots the encoding time for both the Due and the MEGA. For the Due board, the encoding time was previously shown in column F of Fig. 6. We can see that the encoding process takes almost 4 times more on the MEGA than on the Due, i.e. about 2s instead of a bit more than 500ms. The time to read data from the uCAM is still 1.512s. If we assume that the MEGA could have enough memory to handle the image data, we could extrapolate the 1-hop image display latency with the MEGA by adding about 1.5s to the Due's 1-hop image display latency shown in Fig. 7.

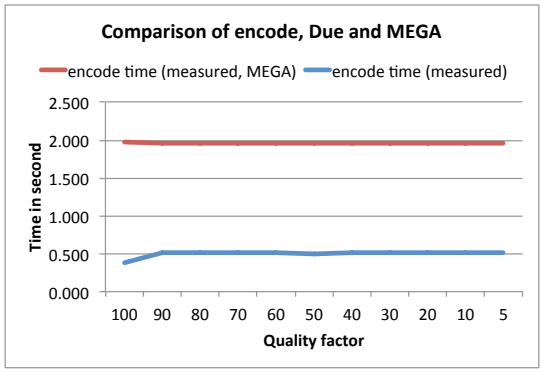

Fig. 9. Comparison of encoding time on Due and MEGA boards

IV. Visual intrusion detection

We implemented a simple intrusion detection mechanism based on "simple-differencing" of pixel: each pixel of the image from the uCAM is compared to the corresponding pixel of a reference image, taken previously at startup of the image sensor and stored in memory. When the difference, in absolute value, is greater than PIX_THRES we increase the number of different pixels, N_DIFF. When N_DIFF is greater than NB_PIX_THRES we assume an intrusion and trigger the transmission of the image. For the intrusion test shown in figure 10, we set PIX_THRES to 35 and NB_PIX_THRES to 300. In doing so, we were able to systematically detect a single person intrusion at 25m without any false alert. Every 5 minutes, the image sensor takes a new reference image to take into account light change conditions. You can visit [16] that describes the tools we use for displaying the received images and how we can share them in real time with a smartphone.

Fig. 10. Left: reference image; Right: intruder detected

The "simple-differencing" method is very light-weight and only adds 1 addition (to compute the pixel difference), 1 comparison (to compare with PIX_THRES) and 1 variable incrementation in case the difference is greater than PIX_THRES.

We measure the time to get data from the uCAM when the ”simple-differencing” method is included and compare it to the previous value. We did not observe any difference: the time to read data from the uCAM is still 1.512s. Note that the ”simple-differencing” process is performed on the raw image. Once the intrusion is detected, a low quality factor can be chosen to reduce the image latency if necessary.

V. Building more accurate simulation models

The simulation model we built for our previous contributions on criticality-based scheduling is developed under the OMNET++/Castalia framework. The model integrates the image encoding scheme and allows for ”real” image packet transmissions under the communication stack and physical radio models (IEEE 802.15.4 in non-beacon CSMA mode). The simulation configuration file indicates the image file that will be transmitted upon intrusion detection. An implementation of a geographical routing protocol enables multi-hop transmission of image packets from an image sensor source to a predefined sink node which will then decode and display the received image. We randomly deployed 150 sensor nodes with random direction of sight in a 400mx400m area and reproduce our image sensor features and constraints:

- camera angle of view of 56^o,
- maximum capture rate of 0.58fps,
- depth of view of 25m,
- 128x128 image (the one of figure 2)
- Quality Factor Q is set to 50, 2265B, 28 packets
- time before image data can be processed is 1.512s,
- encoding time is 500ms.

Random intrusions are introduced in the simulation model and nodes can detect an intrusion if the intruder is covered by their field of view at the time of the image capture. Upon intrusion detection, a node will first broadcast an alert message and will send the image to the sink. Fig. 11 shows the screenshot of the simulation with one image sent by node 60 to node 3 (sink). Node 93 serves as relay node.

In order to have realistic communication overheads, we set the packet overhead at the source to 11ms, i.e. 8ms for the transmission overhead due to OS and communication library (column C) and 3ms for the image packetization time (column G). When the packet size is smaller than 90 bytes, we set the packet overhead to half of its value, i.e. 5.5ms. We showed in [17], [18] that the transmission overhead can not be neglected because it can greatly reduce the sender throughput. Therefore, theoretically, for an image composed of 28 packets, the sending time at the source is 28x11ms=308ms. At 1-hop, the receive latency would be 27*11ms=297ms or 26*11ms+5.5ms=291.5ms when the last packet is smaller than 90 bytes. In the simulation scenario depicted in Fig. 11, the first packet sent by the image source (node 60) at time 86.3608 has been received by node 93 at 86.3653 (the difference is the MAC/PHY overhead). The last packet received by node 93 was at 86.6555. Therefore the receive latency at the first relay node (node 93) is 86.6555-86.3653=290.2ms because the last packet was indeed smaller than 90 bytes.

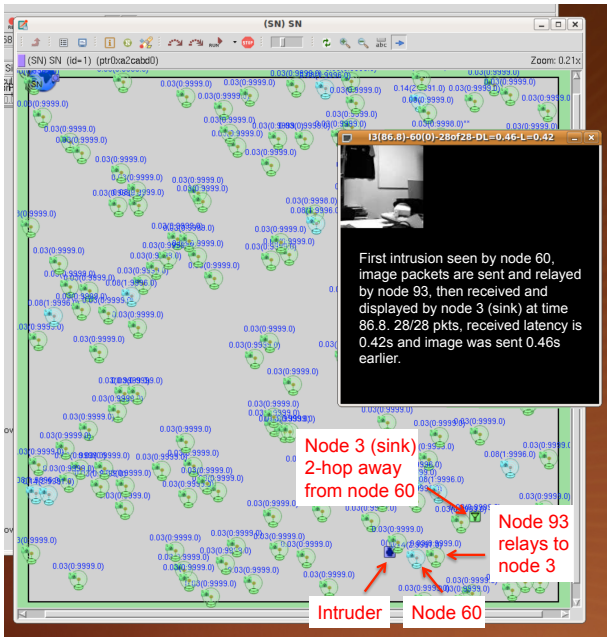

Fig. 11. Screenshot of the simulation environment with image transmission

If we compared the 1-hop receive latency found by simulation model, i.e. 290.2ms, to the 1-hop receive latency found by real experimentation with our image sensor (column J, line Q=50 of Fig. 6), i.e. 349ms, we can see that the simulation model still under-estimate the 1-hop receive latency because not all processing costs are taken into account, especially those for receiving data packets from radio/UART and decoding the image packets at the sink. However, if we do not take at the sender side the packet transmission cost, as many simulation studies did previously, we will erroneously conclude with very unrealistic image transmission capabilities for these low-resource platforms.

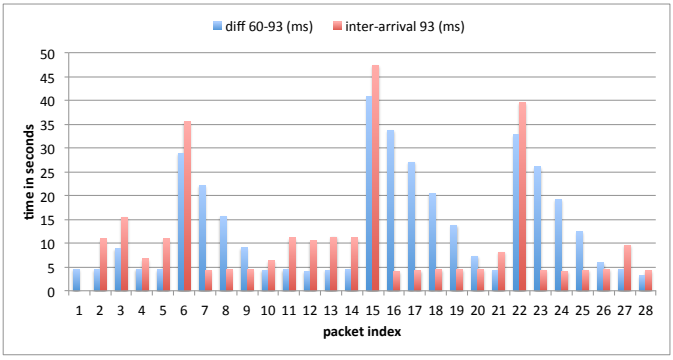

Fig. 12. 1-hop inter-arrival time in simulation model

Fig. 12 shows the difference between the send time at node 60 and the receive time at node 93 (diff 60-93 graph) as well as the inter-arrival time of packets at node 93. Normally, ”diff 60-93” is about 4.4ms representing the MAC/PHY overhead, and the inter-arrival time at node 93 is about 11ms. We can see that in many cases, we have much higher values for ”diff 60-93” due to radio contention and CSMA back-off mechanism

at the source. When this is the case (packet 6 for instance), the inter-arrival time at node 93 increases. However, this inter-arrival time would then decrease for next packets because they are already in radio buffer at the sender side so they will only have the MAC/PHY overhead, making the inter-arrival time of next packets at node 93 to be close to 4.4ms (packets 7-10 for instance).

Regarding the packet relaying time at intermediate nodes, [18] showed that among the well-known sensor platforms used by the research community, the MicaZ sensor has the smallest relaying time which was experimentally measured at about 16ms for an 100-byte packet. This is the value that we set in the simulation model for the relaying time at the routing layer. The first relay node would normally receive 1 packet every 11ms and will be able to relay it in 16ms. Therefore, the 2-hop receive latency for a 28-packet file would theoretically be (28-1)*16ms=432ms. Each additional relay node would add a 16ms delay to the receive latency at the sink. All these predictions are done assuming no errors nor channel contention in transmitting packets. In the simulation, we found a 2-hop receive latency of 420ms: the first packet was received by node 3 at 86.38 while the last packet was received and displayed at 86.8.

Fig. 13 shows the difference between the receive time at node 93 and the receive time at sink node 3 (diff 93-3 graph) as well as the inter-arrival time of packets at sink node 3. Normally, the inter-arrival time at sink node 3 is about 16ms representing the packet relaying overhead. Here we can see the same behavior than previously shown in Fig. 12: due to channel contention, packets tend to queue up at the sender side (here it is the sending part of the relaying node 93) creating periodic bursts of packets arriving back-to-back at the receiving side, only paced by the MAC/PHY overheads.

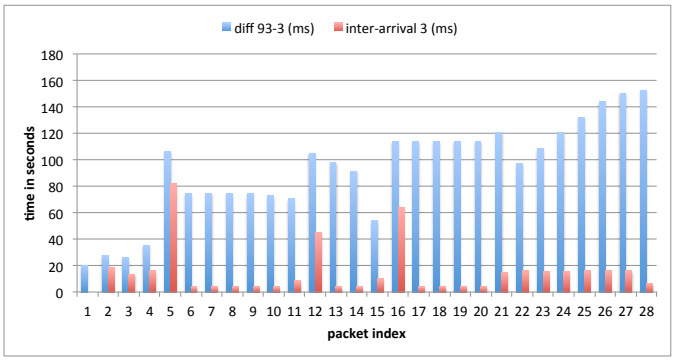

Fig. 13. 2-hop inter-arrival time in simulation model

VI. Conclusions

We presented a low-cost image sensor built from off-the-shelves electronic components for maximum flexibility and availability to the research community. We integrate an efficient image encoding method to produce robust and small-size encoded image suitable for transmission on low-bandwidth radio such as IEEE 802.15.4. The performance measures of the image sensor to operate the image encoding and transmission process show that the total image latency can be less than 2.3s with a reasonable image quality. We targeted visual surveillance applications where the camera can be deployed for situation-awareness or intrusion detections. A simple-differencing approach for intrusion detections shows very good results while adding no additional processing cost. In order to build large-scale visual surveillance systems, we use the experimental measures to adapt the simulation models for more accurate and realistic results regarding multi-hop image transmissions. All the source codes are available on [16]. In future works, we will address the energy consumption issues and integrate low-power consumption techniques for the Arduino board and the radio module in order to also investigate the usage of the image sensor in the context of the Internet of Things for domestic or industrial usage.

Acknowledgment

The author would like to thank V. Lecuire from CRAN laboratory for the image encoding code that has been included in both the simulation model and the image sensor.

References

[1] M. Rahimi et al., “Cyclops: In situ image sensing and interpretation in wireless sensor networks,” in *ACM SenSys*, 2005.
[2] P. Chen et al., “Citric: A low-bandwidth wireless camera network platform,” in *IEEE ICDSC 2008*.
[3] 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 *AVSS 2007*.
[4] Evidence Embedding Technology, “Seed-eye board, a multimedia wsn device. http://rtn.sssup.it/index.php/hardware/seed-eye,” accessed 20/12/2013.
[5] A. Rodriguez-Vazquez et al., ”The eye-ris cmos vision system” in *Analog Circuit Design*, 2008.
[6] 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(2), May 2005.
[7] A. Rowe, D. Goel, and R. Rajkumar, “Firefly mosaic: A vision-enabled wireless sensor networking system,” in *RTSS 2007*.
[8] CMUCam, “Cmucam: open source programmable embedded color vision sensors. http://www.cmucam.org/,” accessed 19/12/2014.
[9] S. Paniga, L. Borsani, A. Redondi, M. Tagliasacchi, and M. Cesana, “Experimental evaluation of a video streaming system for wireless multimedia sensor networks,” in *10th IEEE/IFIP Med-Hoc-Net*, 2011.
[10] ArduCam, “Arducam. http://www.arducam.com,” accessed 19/12/2014.
[11] Arduino Due, “http://arduino.cc/en/pmwiki.php?n=main/arduinoboarddue,” accessed 19/12/2014.
[12] D. Systems, “ucamii. http://www.4dsystems.com.au/product/ucam_ii/,” accessed 19/12/2014.
[13] J. Foster, “Xbee cookbook issue 1.4 for series 1 with 802.15.4 firmware, www.jsjf.demon.co.uk/xbee/xbee.pdf”. Accessed 4/12/2013.
[14] V. Lecuire, L. Makkaoui, and J.-M. Moureaux, “Fast zonal dct for energy conservation in wireless image sensor networks,” *Electronics Letters*, vol. 48, no. 2, 2012.
[15] C. Duran-Faundez and V. Lecuire, “Error resilient image communication with chaotic pixel interleaving for wireless camera sensors,” in *ACM Workshop on Real-World Wireless Sensor Networks*, 2008.
[16] C. Pham, “An image sensor board based on arduino due and ucamii camera. http://www.univ-pau.fr/~cpham/wsn-model/tool-html/imagesensor.html,” accessed 19/12/2014.
[17] C. Pham, V. Lecuire, and J.-M. Moureaux, “Performances of multi-hops image transmissions on ieee 802.15.4 wireless sensor networks for surveillance applications,” in *IEEE WiMob*, 2013.
[18] C. Pham, “Communication performances of ieee 802.15.4 wireless sensor motes for data-intensive applications: A comparison of waspmote, arduino mega, telosb, micaz and imote2 for image surveillance,” *Journal of Network and Computer Applications*, vol. 46, 2014.