Real-time Image Streaming over a Low-Bandwidth Wireless Camera Network

Tim Wark [1], Peter Corke [1], Johannes Karlsson [12], Pavan Sikka [1], Philip Valencia [1]

[1] *Autonomous Systems Laboratory, CSIRO ICT Centre, Brisbane, Australia*
[2] *Digital Media Laboratory, Umeå University, Sweden*
{*tim.wark, peter.corke, johannes.karlsson, pavan.sikka, philip.valencia*}*@csiro.au*

Abstract

In this paper we describe the recent development of a low-bandwidth wireless camera sensor network. We propose a simple, yet effective, network architecture which allows multiple cameras to be connected to the network and synchronize their communication schedules. Image compression of greater than 90% is performed at each node running on a local DSP co-processor, resulting in nodes using 1/8th the energy compared to streaming uncompressed images. We briefly introduce the Fleck wireless node and the DSP/camera sensor, and then outline the network architecture and compression algorithm. The system is able to stream color QVGA images over the network to a base station at up to 2 frames per second.

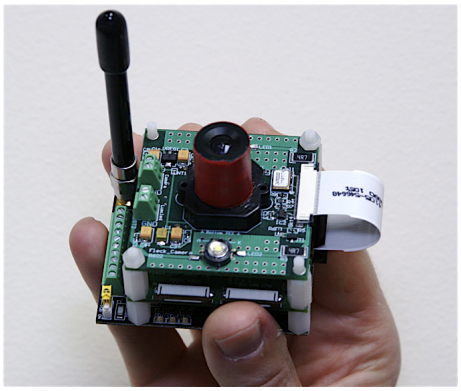


Fig. 1: Illustration of camera stack formed by a Fleck sensor node, DSP board and CCD camera.

1. INTRODUCTION

A. Background

Wireless sensor networks (WSN) have attracted increasing interest over the last few years, driven by theoretical and practical problems in embedded operating systems, network protocols, wireless communications and distributed signal processing [1]. To date the focus has been on simple scalar sensors such as light level, temperature, humidity etc. Sensors such as microphones and cameras provide very rich information about the environment, for example security or farming[2] application, but have the disadvantage of generating very large quantities of data. However the commodification of color CMOS imaging sensors and DSP chips, for mobile phone and other applications, has meant that more capable multimedia nodes are now starting to emerge in WSN applications [3], [4].

Recent work has investigated the application of video coding in WSNs for high-bandwidth networks [5], however we are particularly interested in image processing for low-power, low-bandwidth sensor networks for long-term outdoor deployments [6]. As such our particular focus has been on ways for compressing images at each node to minimise the load on the network.

B. Related Work

Webcams are now a ubiquitous (wireless) networked imaging technology, but their high power and high bandwidth communications are at odds with wireless sensor networks requirements [4].

More relevant to the sensor network requirement are small embedded cameras such as CMUcam1 and CMUcam2[7]. The CMUcam2 comprises a SX52 microcontroller, an OV6620 or OV7620 Omnivision CMOS camera and can communicate via a serial port at speeds upto 115,000 baud. Software running on the CMUcam2 can track coloured blobs at up to 50fps with an image resolution of 160×255.

The Cyclops[8] camera is similar, but uses an Atmega 128 processor and a 352×288 pixel Agilent imager. It also has 64kB of off-chip RAM and 512kB of flash memory, and an interface to a Mica mote. Its significant limitations are the small amount of memory, and the limited address space and computational ability of its processor. The Fleck camera[3], to be described in the next section, overcomes these limitations.

C. Motivation

To meet the needs for additional computation within a sensor network various approaches have been investigated. One option is to increase the capability of the nodes at the expense of cost and power consumption. Another approach is to create a hierarchical network[9] in which signals are transmitted to more capable nodes for processing. However this is expensive in terms of communications and energy. Our motivation in this work has been to develop a robust methodology for streaming image data over "Mote-class" wireless sensor networks – that

is, networks with a focus on low-bandwidth communication, to allow long hop distances for minimal energy consumption. Our key contributions are:

- Development of a new, expandable hardware platform "Fleck-3", to allow multimedia camera nodes to be easily constructed.
- A camera management framework to allow camera nodes to be dynamically added and removed from the network.
- An image compression algorithm which runs on-board nodes, designed specifically for coping with packet-loss experienced during transmission back to base.

2. Hardware Platform

The number of hardware platforms for sensor networks has grown steadily and the latest TinyOS 2 distribution supports 9 platforms from 5 vendors. Polastre et al [10] provide a good overview of micro-controllers and low-power radios. The following subsections describe our custom-designed Fleck-3 platform.

A. Fleck3

The Fleck-3 is a robust platform designed for outdoor sensor networks for environmental monitoring and applications in agriculture. It is the latest in a series of devices developed since 2002[6]. The Fleck-3, consists of an Atmega128 micro-controller running at 8 MHz, a Nordic NRF905 radio transceiver with bit-rate of around 100 kbits/s, a real-time clock, a large amount of flash memory (1MB) and a temperature sensor. An integrated solar battery charging circuit and regulator, with monitoring of battery and solar voltage and current, makes the platform ideal for outdoor, long-term deployments. Our first network[11] has been powered by the sun for over 2 years now.

The Fleck-3 platform is also designed for easy expansion via add-on daughterboards which communicate via digital I/O pins and the SPI bus, and we have developed nearly 20 interfaces for various applications[12]. The DSP daughterboards used for the camera node is described next.

B. DSP Board

The DSP Daughterboard is responsible for all the onboard image processing. It contains a 150MHz TMS320F2812 Signal Processor, with on-chip 128K bytes of program FLASH memory. The DSP also has dual UARTS, SPI bus and numerous digital I/Os. The DSP is a 32-bit processor which provides a large address space required for image processing as well as high computational power. At 150 (32 bit) MIPS it dramatically outperforms the Atmega 128 at 8 (8 bit) MIPS. Power consumption however is much higher, 290mA compared to 9mA, but the DSP is more efficient when normalized to MIPS x width / current: Atmega 128L 0.9MIPS/mA, while the DSP is 2.1MIPS8/mA.

The DSP supports two interfaces: one for a camera and one for an audio codec, and these are connected via a Flat-Flex cable.

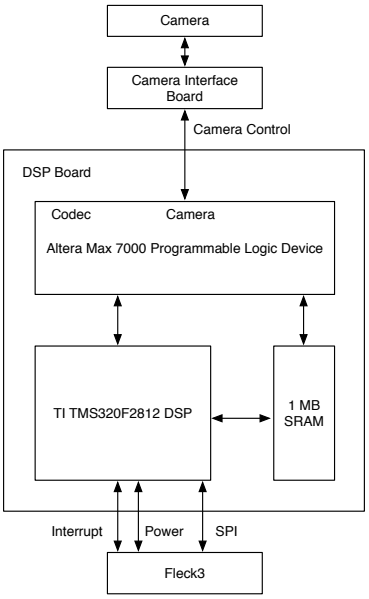


Fig. 2: The hardware architecture of a camera node formed by a Fleck sensor node, DSP board and CCD camera.

(a) Reference Image (b) Input (new) Image

Fig. 3: Illustration of reference and input images. (One hour apart.)

C. Camera Board

The Camera Daughterboard contains the sensor chip, lens holder and two ultra-bright LEDs for illumination. The sensor is an OV7640 Color CMOS from OmniVision Technologies Inc, which outputs VGA (640 x 480) or QVGA (320 x 240) images at a maximum rate of 30fps or 60fps respectively. It uses a Bayer pattern filter to achieve color, has progressive scan and supports windowed and sub-sampled images. The camera parameters such as exposure time and gain can be set by the DSP over the local IIC bus.

Combining a Fleck with the DSP board and a camera board creates a highly functional network camera node, shown in Figure 1. The hardware architecture is shown in Figure 2.

3. System Architecture

A. Overview

The overall system architecture is outlined in Figure 4. The camera network comprises one or more camera nodes in a star-network configuration with a single base node. The base node has a serial connection to a gateway computer and performs two main functions: 1. Any packet received over the radio will be sent over the serial port; 2. Any packet received over the serial port will be sent out over the radio. The gateway computer not only provides the link for sending images back

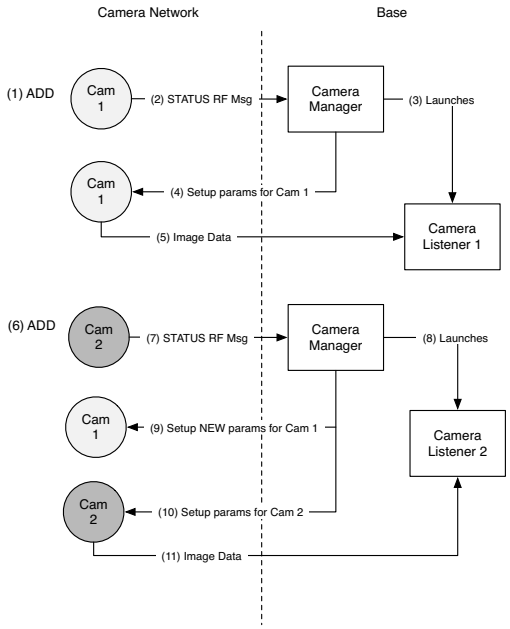


Fig. 5: Illustration of interaction between camera network and base which takes place when new cameras are added to the network.

to a database, but also runs the camera manager program. This is described in more detail below.

B. Camera Manager

The role of the camera manager program is to allow dynamic control of the network, as illustrated in Figure 5. Whenever a new camera is added to the network, specially assigned control messages are broadcast to the network. When these messages get back to base, the manager will spawn a new listener process for this camera.

The other key role of the camera manager is to dynamically reschedule all cameras in the network as cameras are added and removed. Thus, as more cameras are added the duty cycle for each camera will be reduced proportionally. As cameras in the network receive their scheduling parameters from the manager, they will begin sending out compressed images according to the newly allocated schedule.

It should be noted that given our focus on outdoor applications, where solar energy is our key source of energy [11], the camera duty cycles we used were far less than the maximum duty cycle possible given continuous power, so adding more cameras to a network would typically not change the duty cycles. These energy considerations are discussed more in Section 5-B.

C. Operating System

All software running on the Fleck-3 mainboard used the TinyOS [13] operating system and was written in the NesC [14] language. TinyOS is an event-driven, component-based OS especially developed for platforms such as sensor network nodes with very limited resources.

4. IMAGE COMPRESSION ALGORITHM

Due to the limited bandwidth of a wireless sensor network (typically $<$ 100kbps), a key aspect of our work has been compression of images at the nodes. The additional computational energy required to compress images is more than offset by the reduction in data required to be sent over the radio. This is discussed in more detail in Section 5-B. The other key advantage of this approach is greatly reducing the amount of data in the air, allowing for many more cameras to be added to the network.

A. Encoding

The main steps in the image encoding algorithm are outlined in Figure 6. As an initial step, the packed YUV422 image sent from the camera is unpacked and chroma subsampled to create a YUV420 image with 16-bit pixels which the DSP can more efficiently manipulate.

The image is then treated as a series of 8×8 blocks. Each block is compared with a previously saved reference image stored on the DSP external SRAM. Three approaches are taken in the way blocks are dealt with at the encoder (camera node) side:

1) **Skip block:** If the mean-square error (MSE) between this block and corresponding block in the reference frame is below some threshold then don't transmit the block.
2) **Intra-block encode:** Encoding is performed by encoding the block without using any reference.
3) **Inter-block encode:** Encoding is performed by encoding the difference between the current block and the corresponding block in the reference frame.

For non-skip blocks the encoding process very similar to JPEG compression. The block is converted to frequency space using a 2D discrete cosine transform (DCT type-II), quantized and zigzag scanned. Finally it is encoded using run length encoding (RLE) and Huffman encoding. Both intra and inter-block encoding is done and the type producing the least bits is selected for transmission. For the input and reference images shown in Figure 3, the types of blocks selected are shown in Figure 8.

B. Packet structure

The structure of radio packet contents is shown in Figure 9. In each radio packet an integer number of blocks are transmitted. The benefit of this approach is that no resynchronization markers are needed in the Huffman encoded bitstream. The packet header contains information about the index of the first block within the image and the number of blocks in the packet. Following this are the bit-encoded blocks, each preceded by a block type: 1 for skip, 00 for intra-, and 01 for inter-block. A radio packet of length 32 bytes typically holds several blocks.

C. Decoding

The decoding algorithm is shown in Figure 7 which is implemented by a Java application which processes the packets at the base and updates the image. Since predictive coding is

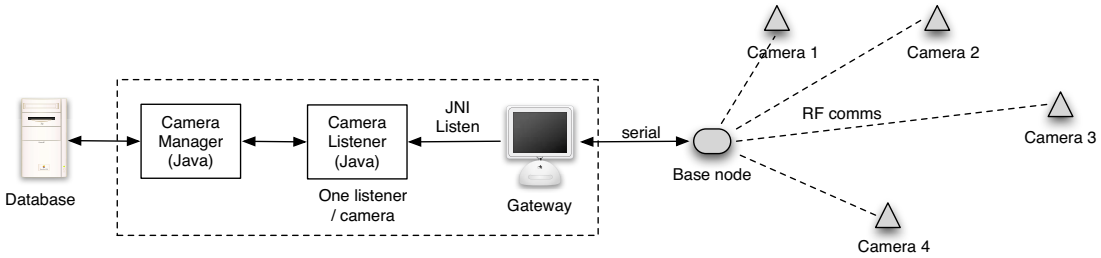


Fig. 4: Outline of network architecture. The camera manager process will spawn a new camera listener for each new camera that is detected by the base.

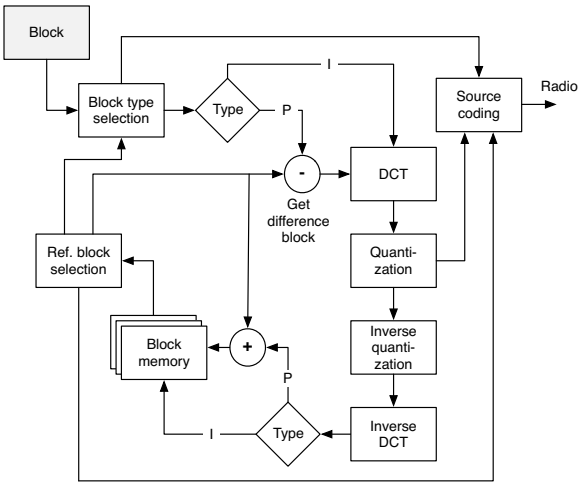


Fig. 6: Simplified block diagram of encode algorithm.

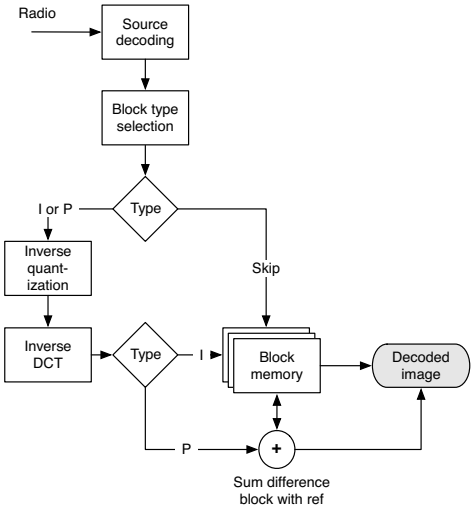
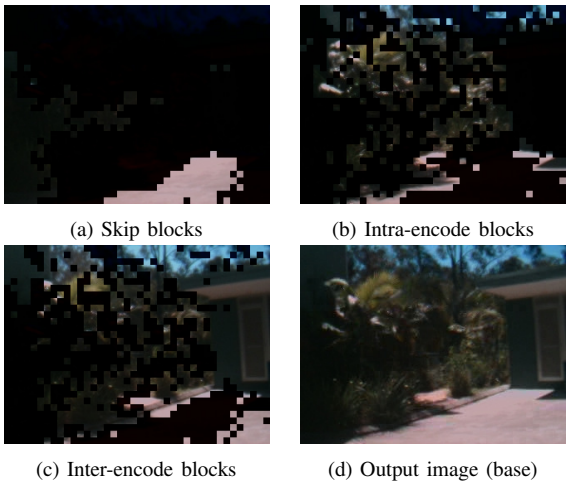


Fig. 7: Simplified block diagram of decode algorithm.



(a) Skip blocks (b) Intra-encode blocks

(c) Inter-encode blocks (d) Output image (base)

Fig. 8: Illustration of ways in which blocks are encoded to send data.

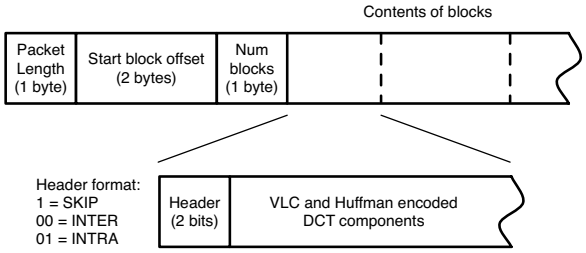


Fig. 9: Structure of packet contents used for compressed image blocks.

used, a lost packet will not only cause error in the current frame, but the error will continue to propagate into the following frames. To stop the error propagation, intra blocks must be is inserted.

Two strategies are used for deciding when to insert these intra blocks depending on whether feedback from the decoder can be provided or not. If feedback can be provided, the decoder sends information about lost blocks to the encoder and these blocks are then encoded using intra mode. If no feedback is provided, as is the case in our current implementation, the encoder will insert an intra block periodically given a maximum intra block interval. Error control at block level can be used since the lack of motion compensation in our scheme will not cause the error to propagate in the spatial domain. Ongoing work is investigating the use of feedback messages

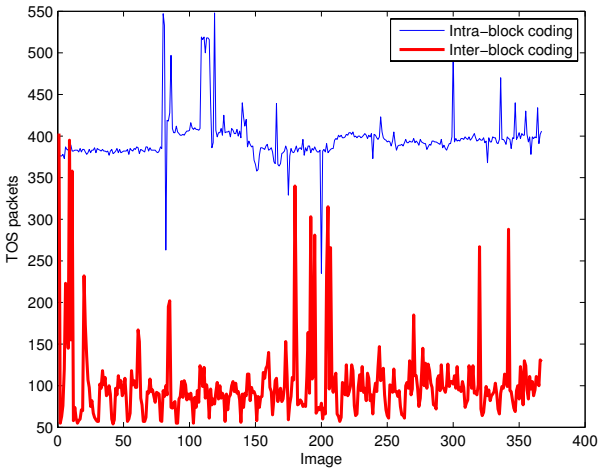


Fig. 10: The number of packets requires to transfer an image over an extended image sequence.

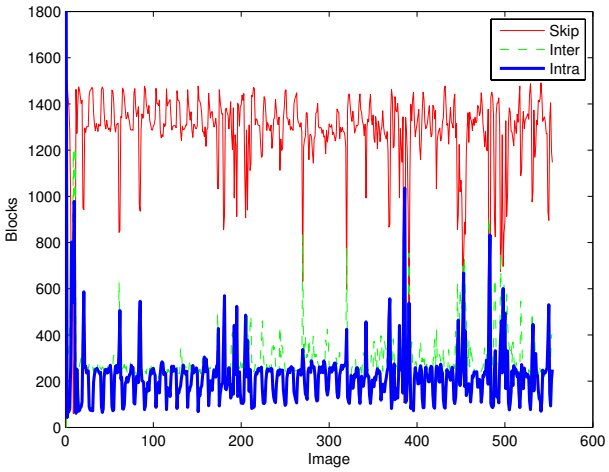


Fig. 11: The number of the different types of blocks required to transfer an image over an extended image sequence.

to the camera to determine optimal times to insert forced intra blocks.

5. Evaluation

A. Packet Transmission

Figure 10 shows the number of packets transmitted for each image transferred over a long sequence of images over a period of several hours with people moving occasionally in front of an otherwise static scene. We compare two cases, where intra-block coding is enforced, top curve, and where inter-block coding is allowed. For the enforced intra-block case the average number of TOS packets is 395 packets or 10.2kbyte of data. Assuming 1.5 bytes per pixel in YUV420 format, this represents a compression to 8.8%. The enforced intra-block method is quite robust to lost packets, those regions of the image corresponding to the lost packet will be refreshed in the next cycle. When allowing inter-block coding, see Figure 11, we achieve much greater compression to an average of if 106 packets, 2.8bytes or 2.4%. For both these cases a network of two cameras was used for evaluation.

The Fleck 3 has a typical throughput of 125 packets/second, allowing a frame rate of one image/second. The average numbers of the different types of block are 71% skip, 12% intra- and 17% inter-block coding.

Figure 12 shows 4 images sampled from a long sequence of images taken over a day once every 5 minutes. The camera position was static and different objects moved through its field of view over the day. This particular sequence clearly shows a puddle of water drying up over the day, providing new meaning to the phrase “watching the grass grow”.

Fig. 12: Four sample images out of a sequence of about 150 images over a day.

B. Energy Usage

In order to show the advantage of compressing images at the node as we have done, we have evaluated the energy required to compress then send image data over the radio compared with the energy required in order to send an uncompressed image over the radio. In order to implement this second configuration, we made another version of the camera node in which the image was stored on an compact flash memory (MMC) daughterboard. This allowed the DSP to be switched off during the extended time taken to transmit the raw image.

The average power and energy consumption during each phase of operation (for the compressed and uncompressed configurations) is given in Tables 1 and 2. Aggregating the data in the tables we can calculate the total energy required to compress an image then transmit is around 1086mJ whereas the total energy required to store then transmit an uncompressed image is around 8018mJ – around 8 times more energy! Clearly the small amount of additional energy required to compress at the node is highly beneficial.

C. Compression Performance

To assess the actual performance of the compression algorithm we ran the camera for six days in an outdoor environment, very similar to the one shown in Figure 8. The environment was chosen as it provided strong variation in shadows as the

TABLE 1: AVERAGE ENERGY CONSUMED BY FLECK CAMERA NODE WHERE COMPRESSION (INTER-FRAME) IS USED. IN THIS CONFIGURATION IMAGE IS COMPRESSED ON THE DSP AND THEN SENT OUT OVER THE RADIO.

	Camera On	DSP on	DSP → TX
Voltage	3.3V	3.3V	3.3V
Current	30mA	240mA	260mA
Power	99mW	792mW	858mW
Time	0.03s	0.5s	0.8s
Energy	3.0mJ	396mJ	687mJ

TABLE 2: AVERAGE ENERGY CONSUMED BY FLECK CAMERA NODE WHERE NO COMPRESSION IS USED. IN THIS CONFIGURATION AN IMAGE IS SENT FROM THE DSP MEMORY TO AN MMC BEFORE TRANSMITTING FROM THE MMC OVER THE RADIO.

	Camera On	DSP on	DSP → MMC	MMC → TX
Voltage	3.3V	3.3V	3.3V	3.3V
Current	30mA	240mA	270mA	50mA
Power	99mW	792mW	891mW	165mW
Time	0.03s	0.1s	1.5s	40s
Energy	3.0mJ	79.2mJ	1336mJ	6600mJ

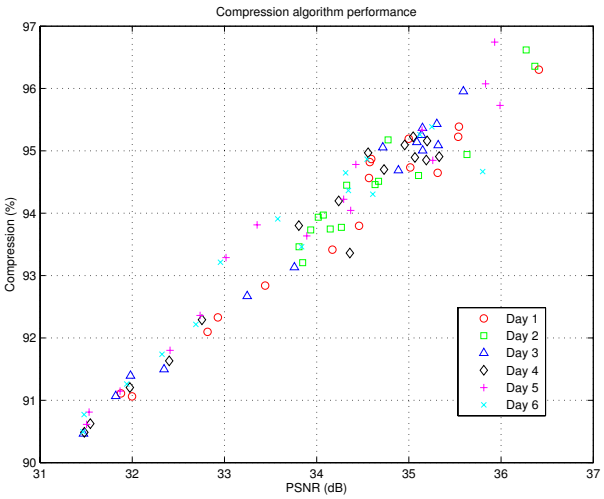


Fig. 13: Evaluation of camera compression algorithm over six days. Camera images were taken at 1 hour intervals during daylight hours. (Each point represents a single image.)

day passed by, as well as passing vehicles and people. Images were taken at one hour intervals and constrained to daylight hours. For each compressed frame we calculated the total amount of compression and the Peak Signal to Noise Ratio (PSNR) between the final decoded image and the original raw image. In order to achieve this, we used an offline version of the compression algorithm which we ran over a sequence of uncompressed images which has been sent over the camera network previously.

The results shown in Figure 13 clearly show the linear relationship between PSNR and compression. As more change occurs between shots, the differences between the current frame and previous reference frame increases which reduces the amount of compression and the PSNR level proportionally. Over a wide range of conditions our compression algorithm has shown very good performance with compression levels ranging between 90% and 97% and PSNR between 31dB to 36.5dB according.

6. CONCLUSIONS AND FUTURE WORK

This paper has presented our architecture for streaming image data over a low-bandwidth camera network. The architecture allows for cameras to be dynamically added and removed from the network, as well as undertaking compression at each node to minimise energy in transmitting image data over the network.

Future work will focus on adding multi-hop functionality to the network architecture as well as integrating plug'n'play type functionality to nodes to allow various filter functions (e.g. low-pass filtering or edge-detection) to be dynamically added to a node's operations.

ACKNOWLEDGMENTS

The authors wish to thank Ben McKay, Natasha Mendes and Michael Ung for their work on low-level DSP video software, and Les Overs and Stephen Brosnan for the Fleck, DSP and camera hardware.

REFERENCES

[1] B. Krishnamachari, *Networking Wireless Sensors*. Cambridge University Press, 2005.

[2] P. Sikka, P. Corke, and L. Overs, "Wireless sensor devices for animal tracking and control," in *First IEEE Workshop on Embedded Networked Sensors*, pp. 446–454, 2004.

[3] P. Corke, "Non-berkeley platforms." Presented at TinyOS Technlogy Exchange (TTX3), Stanford, Feb 2006. http://www.eecs.berkeley.edu/čuller/tinyos/ttx/slides/Platforms/CSIRO.pdf.

[4] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, pp. 921–960, 2007.

[5] R. Puri, A. Majumdar, P. Ishwar, and K. Ramchandran, "Distributed video coding in wireless sensor networks," *IEEE Signal Processing Magazine*, vol. 23, no. 4, pp. 94–106, 2006.

[6] T. Wark, P. Corke, P. Sikka, L. Klingbeil, Y. Guo, C. Crossman, P. Valencia, D. Swain, and G. Bishop-Hurley, "Transforming agriculture through pervasive wireless sensor networks," *IEEE Pervasive Computing*, vol. 6, no. 2, pp. 50–57, 2007.

[7] A. Rowe, C. Rosenberg, and I. Nourbakhsh, "A low cost embedded color vision system," in *Proc. IROS*, 2002.

[8] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava, "Cyclops: In situ image sensing and interpretation in wireless sensor networks," in *Proc. ACM SenSys05*, (San Diego, California, USA), November 2005.

[9] W. Hu, V. N. Tran, N. Bulusu, C. T. Chou, S. Jha, and A. Taylor, "The design and evaluation of a hybrid sensor network for cane-toad monitoring," in *IPSN '05*, (Piscataway, NJ, USA), p. 71, IEEE Press, 2005.

[10] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *IPSN/SPOTS*, 2005.

[11] P. Corke, P. Valencia, P. Sikka, T. Wark, and L. Overs, "Long-duration solar-powered wireless sensor networks," in *EmNets*, 2007.

[12] P. Corke, S. Sen, P. Sikka, and T. Wark, "Wireless sensor network: two-year progress report," Tech. Rep. TR 06/249, CSIRO ICT Centre, August 2006. http://www.csiro.au/files/files/p8zh.pdf.

[13] J. Hill, R. Szeweczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for network sensors," in *Archirectual Support for Programming Languages and Operating Systems*, 2000.

[14] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc langauge: A holistic approach for network sensors.," in *Programming Language Design and Implementation*, 2003.