In Proc. of 2nd Int. IEEE/Create-Net Conf. on Testbeds and Research Infrastructures for the Development of Networks and Communities, Mar. 2006, Barcelona, Spain.

WiSNAP: A Wireless Image Sensor Network Application Platform

Stephan Hengstler and Hamid Aghajan

***Abstract*—Wireless networks in combination with image sensors open up a multitude of previously unthinkable sensing applications. Capable tools and testbeds for these wireless image sensor networks can greatly accelerate development of complex, yet efficient algorithms that meet application requirements. In this paper, we introduce WiSNAP, a Matlab-based application development platform intended for wireless image sensor networks. It allows researchers and developers of such networks to investigate, design, and evaluate algorithms and applications using real target hardware. WiSNAP offers standardized and easy-to-use Application Program Interfaces (APIs) to control image sensors and wireless motes, which do not require detailed knowledge of the target hardware. Nonetheless, its open system architecture enables support of virtually any kind of sensor or wireless mote. Application examples are presented to illustrate the usage of WiSNAP as a powerful development tool.**

I. Introduction

Wireless sensor networks open up an entirely new field for research and development. These types of networks conceptually offer many exciting features including scalability, self-configuration, self-healing, multicast routing, and easy deployment [1]. For these reasons, they are well suited for a wide range of applications in monitoring, control, surveillance, and distributed sensing among many others [2]. In particular, image sensor-based wireless networks are deemed advantageous for many of these applications [3]. However, specific applications still need to be defined, and their implementation requires further work.

The availability of suitable tools can greatly aid investigation and development of applications for wireless sensor networks. Although several testbeds for wireless sensor networks have been proposed [4]–[6], they often lack high-level support for effective algorithm development or they are tailored to a limited set of hardware. For this purpose, Matlab [7] may serve as one appropriate rapid-prototyping tool. Its high-level programming environment would allow for easy design, implementation, and emulation of algorithms and applications for wireless image sensor networks if interfaces to wireless motes and image sensors exist. Such interfaces could be realized as standardized libraries, which provide easy-to-use functions to communicate with a variety of image sensors and wireless motes. The use of such libraries unburdens its potential user from having to deal with mote- or sensor-specific interface details. A set of such standardized libraries could form a Matlab-based framework for wireless image sensor networks, in which algorithms and applications can be developed in an efficient manner.

Wireless image sensor networks in particular often require a great deal of algorithm work since robust and reliable information extraction from image data typically results in complex, fine-tuned image processing algorithms. Use of high-level development testbeds can significantly accelerate cycle time for development and verification of image-based sensor algorithms. Following the design flow referred to, the algorithm is first developed and evaluated in a high-level development platform, which provides easier and more comprehensive insight into algorithm performance than most low-level target hardware tools. Once the desired performance criteria are met, the algorithm can be ported to the final target hardware without the need for further algorithm design. A limitation of this design flow however, which one frequently encounters, lies in the insufficient real-time capability of high-level development environments. Nonetheless, equivalent-time emulation can adequately model real-time performance of many algorithms.

Conceptually, the approach outlined here to create a development platform for wireless image sensor networks can be applied to high-level simulation and analysis environments other than Matlab. For example, Agilent VEE Pro [8] and LabVIEW [9], which both provide extensive interface capabilities to test and measurement equipment, represent possible choices although they target primarily data acquisition and analysis. In our research, we prefer the Matlab interactive, high-level language environment mostly due to its powerful features in algorithm development and graphical visualization.

The remainder of the paper is organized as follows. In Section II, we describe the overall system architecture of our proposed application development platform for wireless image sensor networks. A more detailed explanation of its application program interface layer and its device libraries follow in Sections III and IV, respectively. Application examples for

Author affiliations: S. Hengstler, Avago Technologies, San Jose, CA 95131, USA, Email: stephan.hengstler@ieee.org; H. Aghajan, Wireless Sensor Networks Lab, Department of Electrical Engineering, Stanford University, Stanford, CA 94305, USA, Email: aghajan@stanford.edu.

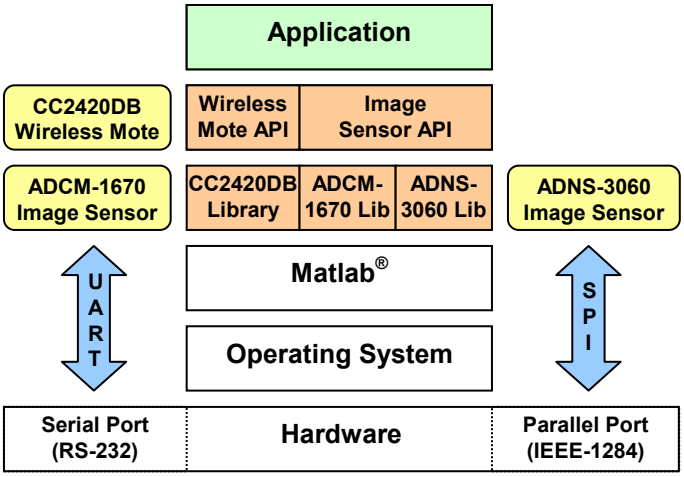

Fig. 1. Program stack of WiSNAP.

event detection and node localization are presented and discussed in Section V. Finally, Section VI summarizes the main aspects of the application development platform introduced and outlines directions for further work.

II. System Architecture

For the reasons stated in the introduction, the Wireless Image Sensor Network Application Platform (WiSNAP) introduced in this paper provides a Matlab framework for researching, developing, and investigating algorithms and applications for wireless image sensor networks. WiSNAP can be obtained from the web site of Stanford's Wireless Sensor Networks Lab [10]. Fig. 1 illustrates its program stack. WiSNAP consists of two Application Program Interfaces (APIs): an Image Sensor API and a Wireless Mote API. These APIs provide the user for application development with simple, easy-to-use functions to interface with image sensors and wireless motes. Thus, there is no need to deal with hardware- and device-specific details. These are taken care of by underlying device libraries that execute the necessary hardware-specific protocols and functions. The device libraries can either be implemented as Matlab scripts or as Matlab executables. Through functions provided by the operating system, they gain low-level access to the computer's hardware including its peripheral interfaces.

The APIs and libraries are written to facilitate easy extension to other image sensors and wireless motes or to utilize some device-specific functionality. The current implementation of WiSNAP includes device libraries for Agilent's ADCM-1670 camera module [11], Agilent's ADNS-3060 optical mouse sensor [12], and for Chipcon's CC2420DB IEEE 802.15.4 compliant demonstration board [13]. The details of each device library are briefly described in Section IV. At this point, WiSNAP does by no means represent a complete or exhaustive implementation of available image sensors or wireless motes. It rather intends to establish an open-system development structure that can be easily extended by other developers based on their particular application needs.

TABLE I
Image Sensor API Functions

Function	Description
'open'	Opens a communications session with the image sensor.
'init'	Initializes the image sensor.
'frame'	Captures the current frame from the image sensor.
'close'	Closes the communications session with the image sensor.

TABLE II
Wireless Mote API Functions

Function	Description
'open'	Opens a communications session with the wireless mote.
'init'	Initializes the wireless mote's channel and node address.
'recv'	Receives a MAC packet from a wireless mote.
'send'	Transmits a MAC packet to a wireless mote.
'close'	Closes the communications session with the wireless mote.

III. Application Program Interfaces

As mentioned before, the role of the API layer is to provide simple, hardware-independent functions to the application layer for application and algorithm development. Currently WiSNAP includes two APIs: the Image Sensor API enables frame capturing from images sensors and the Wireless Mote API provides access to wireless motes. Nonetheless, the open architecture of the WiSNAP framework allows easy integration of additional APIs. For example, a separate API for sensors providing scalar outputs like temperature, pressure, acceleration, or velocity can readily be added to the existing application platform presented in this paper.

A. Image Sensor API

The purpose of the Image Sensor API lies in allowing its user to capture frames from an attached image sensor. Hence this API requires only a very limited function set, which is summarized in Table I.

A user interested in capturing a number of frames from a particular image sensor would first open a communications session to the image sensor (*'open'*), initialize it (*'init'*), capture as many frames as desired (*'frame'*), and finally close the communications session (*'close'*).

To give an example, the syntax of the function *'frame'* is
imager = image_sensor_api(device,'frame',handle),
where *device* denotes a string containing the device name of the attached image sensor, *handle* is a device handle provided by the function *'open'*, and *imager* returns the image array data captured by the sensor.

B. Wireless Mote API

The Wireless Mote API presents an easy-to-use application interface to wireless motes. Most importantly it provides functions for mote initialization and medium access control

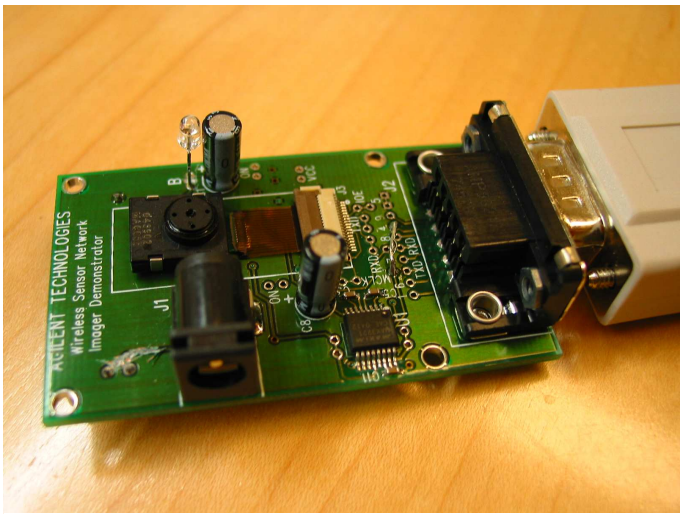

Fig. 2. Agilent's ADCM-1670 camera module mounted on serial interface board. Equipment courtesy of Agilent Technologies.

(MAC) packet transmission to other motes and reception from other motes in the network. Table II summarizes this API's function set.

A user wanting to send and receive packets within the wireless sensor network would proceed as follows. Similar to the steps described for the Image Sensor API, open a communications session to the wireless mote (*'open'*) and initialize it with the desired channel number and node address (*'init'*) before sending (*'send'*) and receiving (*'recv'*) MAC packets to and from other motes within the network.

For instance, the syntax of the function *'send'* is *status = wireless_mote_api(device,'send',handle,TxMacPckt)*, where *device* denotes a string containing the device name of the attached wireless mote, *handle* is a device handle provided by the function *'open'*, and *status* indicates whether the packet transmission completed successfully. The structure *TxMacPckt* contains at least the destination mote's address and the packet payload.

IV. Device Libraries

The device libraries reside below the API layer. For a specific image sensor or wireless mote, a device library provides a set of hardware-dependent functions, which match the corresponding set of functions of the overlaid API. Thus, the device library "knows" the interface and instruction set of the device it is written for.

A. Agilent ADCM-1670 Image Sensor

Agilent's ADCM-1670 camera module (Fig. 2) can be used as a cost effective, medium resolution image sensor for image sensing applications. Its low power dissipation of typically less than 90 mW makes it especially appealing for energy-constrained wireless sensor networks. The ADCM-1670 camera module offers a programmable resolution of up to 352x352 pixels, selectable output format between JPEG, YCbCr, RGB, or grayscale only and built-in pan and zoom capability. In addition, an internal frame buffer can store image data up to 48 Kbytes.

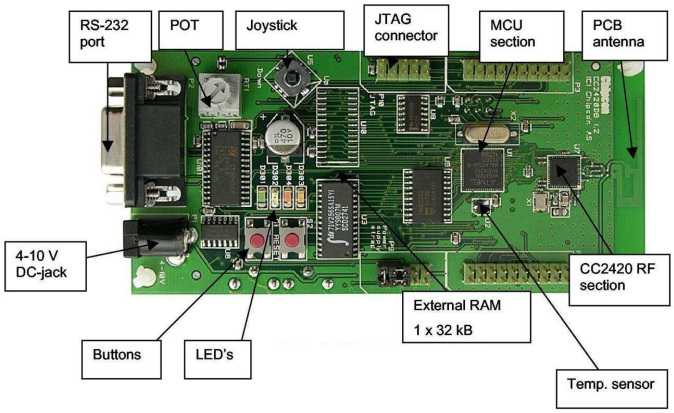

Fig. 3. Chipcon's CC2420DB demonstration board. Photo and equipment courtesy of Chipcon AS.

Both sensor control and image output take place over a simple Universal Asynchronous Receiver-Transmitter (UART) interface. Utilizing a RS-232 level-shifter, this sensor can be directly connected to a computer's serial port. This allows the ADCM-1670 device library to operate the image sensor by transferring character data through an available serial port.

B. Agilent ADNS-3060 Image Sensor

Agilent's ADNS-3060 optical mouse sensor is actually intended for tracking applications in high-performance optical computer mice. However, its programming capabilities allow for reading out the raw image captured by the sensor's pixel array. Operated this way, the ADNS-3060 makes a suitable low resolution image sensor for image sensor networks as it provides fixed 30x30 pixel, 6-bit grayscale images. Its extraordinary frame rates of up to almost 6500 frames per second may especially appeal to applications that require target or object detection of high acceleration or velocity.

The ADNS-3060 image sensor employs a four-wire Serial Peripheral Interface (SPI) for programming and image acquisition. The ADNS-3060 device library can directly communicate with the sensor's interface by bit-banging the SPI protocol onto the computer's IEEE-1284 parallel port. Our WiSNAP implementation uses a Logitech MX310 Optical Mouse [14], which has an ADNS-3060 optical mouse sensor built in, but the SPI interface is intercepted by the computer's parallel port.

C. Chipcon CC2420DB Wireless Mote

The Chipcon CC2420DB IEEE 802.15.4 compliant demonstration board (Fig. 3) pairs an Atmel 8-bit AVR ATmega128L microcontroller [15] with a Chipcon CC2420 2.4 GHz IEEE 802.15.4 RF transceiver [16], which makes it a powerful wireless mote. Furthermore, the demonstration board comes with a micro-strip antenna, RS-232 and SPI interfaces, and several general-purpose input-output pins. Hence it can

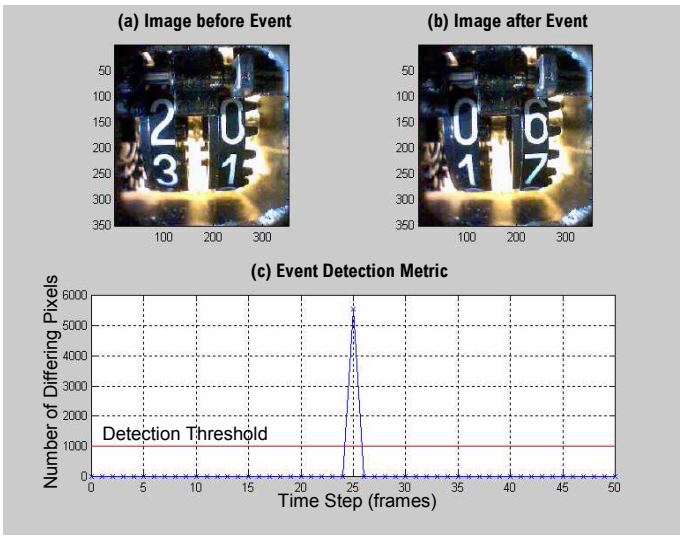

Fig. 4. Screenshot of event detection example: (a) image before event, (b) image after event, and (c) event detection metric with detection threshold.

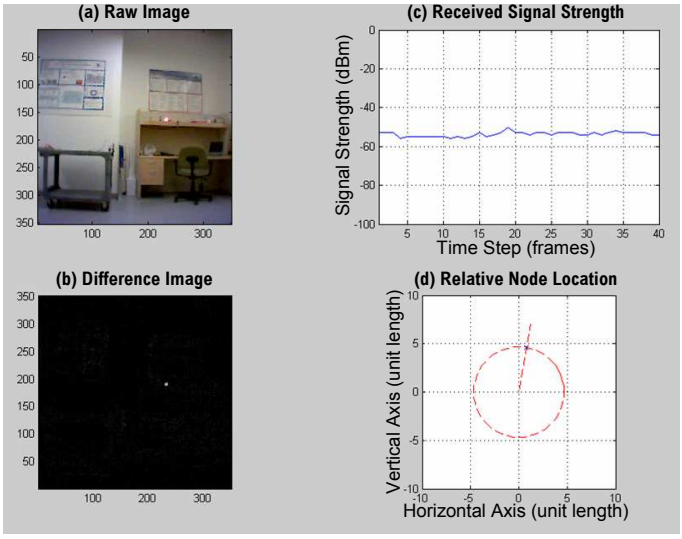

Fig. 5. Screenshot of node localization example: (a) raw image, (b) difference image, (c) received signal strength, and (d) relative node location.

easily interface to a multitude of additional input or output devices like sensors or indicators.

To readily interface to the CC2420DB device library, the microcontroller executes a custom-designed terminal application firmware that receives commands through one of the on-board serial RS-232 ports. Commands supported by the terminal application include microcontroller register read/write, transceiver register read/write, transceiver initialization, and IEEE 802.15.4 MAC packet reception and transmission. In addition, the terminal application also accepts commands in the form of MAC packets. Thus, the CC2420DB device library can completely control the wireless mote either through the serial interface or through IEEE 802.15.4 MAC communication. Conceptually, one instance of WiSNAP may therefore remotely configure and operate an entire network of CC2420DB wireless motes.

D. *Extension to Other Devices*

It should be straightforward to develop device libraries for virtually any sensor or wireless mote following the concept of the library examples presented in this paper. In short, additional device libraries first need to establish an interface from the computer to the device of interest, and subsequently implement the device-dependent functions required by the overlaid API for this type of device.

V. Application Examples

The following two application examples demonstrate the use of WiSNAP for algorithm and application implementation and emulation. The key sections of the examples' Matlab source code are listed in Appendix A and B, respectively. The application examples presented here are primarily intended to introduce the potential of WiSNAP in the development of wireless image sensor networks. More innovative, complex and demanding applications can be easily envisioned and are currently under development.

A. *Event Detection*

The first application example illustrates the usage of WiSNAP's image sensor API for visual event detection. More specifically, the computer running WiSNAP has an Agilent ADCM-1670 image sensor attached to a serial port. The sensor looks at two digits of a numeric counter dial. The application attempts to detect the event of a changing counter reading. Our straightforward approach accomplishes this detection by tracking the number of pixels that differ substantially between successive images. Once this number exceeds a threshold preset above the camera's noise level, the event has occurred and the algorithm can issue a notification for further action. For instance, this case occurs in applications for automatic meter reading or, more generally, in applications that require event detection in an image sensor's field of view.

A screenshot of this event detection example is presented in Fig. 4. Subplots (a) and (b) show the meter reading before and after the event detection along with subplot (c) graphing the number of pixels differing from frame to frame and the preset detection threshold. The detected event is clearly visible in the latter plot at time step 25 frames.

B. *Node Localization*

In the second application example, we describe how our development platform can be applied to a node localization problem. The setup of the wireless image sensor networks is as follows. A central node with attached image sensor looks at its environment to detect the relative location of a neighboring node. The central node consists of a CC2420DB wireless mote and an ADCM-1670 camera module; each of them connected to a computer's serial port. The node neighbor operates as a stand-alone CC2420DB mote, i.e. it has no connection to a computer. To signal its location, it can toggle a red light-

emitting diode (LED) upon remote request from the central node. Assuming the adjacent node is within communication range and field of view of the central node, this network configuration enables the central WiSNAP node to determine distance and direction of the node neighbor. The central node knows the fixed transmit power and can calculate the receive power from the received signal strength indicator (RSSI) embedded in received MAC packets. Hence its distance to the neighboring node can be approximated from the free-space loss model. Note that this represents an idealized, but convenient assumption; in fact, inferring distance from RSSI measurements is highly involved in indoor and even outdoor network deployments. To determine the direction of the neighboring node, the central node extracts the relative angle of the continuously blinking LED from its captured images according to the pinhole camera model. The position of this LED within the camera's image array can be easily determined through frame differencing of consecutive frames. Lastly, the central node estimates the neighboring node's two-dimensional location relative to its own as the intersection of the circle specified by the distance and the secant defined by the relative angle of the neighboring node.

Fig. 5 gives a screenshot of this node localization example. Subplots (a) and (b) respectively display the raw and the difference image, which clearly indicate the image coordinates of the neighboring node's LED position. Furthermore, subplot (c) shows the received signal strength over time. Finally, the estimated topology is visualized in subplot (d) with the central node in the center and the neighboring node at the intersection of the aforementioned circle and secant.

Obviously, it is straightforward to extend this example application towards a network deployment of more than just two nodes, in which the nodes discover their neighbors within radio and visual range and possibly even work out the global network topology map. Such topology information can assist network deployment and even operation to determine its coverage area for instance. As another example, geographic routing requires knowledge of node positions, which could be obtained following the approach described here.

VI. Conclusion

In this paper, we have introduced WiSNAP, a novel, Matlab-based application platform for wireless image sensor networks. Its standardized application program interface layer and underlying device libraries facilitate high-level algorithm and application development on real image sensor and wireless mote hardware devices. Furthermore, WiSNAP's open system architecture can readily accommodate virtually any type of sensor or mote device. The two application examples presented, event detection and node localization, demonstrate the easy deployment of WiSNAP for efficient application development and emulation of wireless image sensor networks. Future work on WiSNAP will focus on implementing applications in larger deployments of wireless sensor networks and on extending the device libraries to additional image sensors and wireless motes.

Appendix

A. Source Code of Event Detection

```
% open communications session
shandle = image_sensor_api(SENSOR,'open',SPORT);
% initialize image sensor
status  = image_sensor_api(SENSOR,'init',shandle);
% initialize frame counter
frame   = 1;
% initialize memory ping-pong
pong    = 0;

% repeat until break
while(1)
  % capture current frame
  imager = image_sensor_api(SENSOR,'frame',shandle);
  % store grayscale image in correlation memory
  cmem(:,:,pong+1) = sum(imager,3);

  % perform event detection task
  if (frame > 1)
    % compute difference image
    cmem_dif = abs(cmem(:,:,not(pong)+1) ...
      - cmem(:,:,pong+1));
    % determine number of pixels above threshold
    N_E(frame) = sum(sum(cmem_dif > Th));
  end

  % increment frame counter
  frame = frame + 1;
  % ping-pong correlation memories
  pong = not(pong);
end

% close communications session
status  = image_sensor_api(SENSOR,'close',shandle);
```

B. Source Code of Node Localization

```
% open communications sessions
mhandle = wireless_mote_api(MOTE,'open',MPORT);
shandle = image_sensor_api(SENSOR,'open',SPORT);
% initialize wirless mote
status  = ...
  wireless_mote_api(MOTE,'init',mhandle,PhyInfo);
% initialize image sensor
status  = image_sensor_api(SENSOR,'init',shandle);
% initialize frame counter
frame   = 1;
% initialize memory ping-pong
pong    = 0;

% repeat until break
while(1)
  % capture current frame
  imager = image_sensor_api(SENSOR,'frame',shandle);
  % remotely toggle neighboring node's location LED
  TxMacPckt.pPayload = sprintf('t\n');
  TxMacPckt.length = length(TxMacPckt.pPayload);
  % send transmit packet
  status = ...
    wireless_mote_api(MOTE,'send',mhandle,TxMacPckt)
  % check for received packet
  RxMacPckt = ...
    wireless_mote_api(MOTE,'recv',mhandle);
  % determine received signal strength indicator
```

```
  rssi(frame) = RxMacPckt.rssi;
  % calculate neighboring node's distance
  distance(frame) = sqrt(1/10^(rssi(frame)/10+4.5));
  % store green image array in correlation memory
  cmem(:,:,pong+1) = flipud(imagerimager(:,:,2));

  % perform node localization task
  if (frame > 1)
    % compute difference image
    cmem_dif = abs(cmem(:,:,not(pong)+1) ...
      - cmem(:,:,pong+1));
    % determine neighboring node's LED position
    threshold = max(mean(mean(cmem_dif)) ...
      + (std(std(cmem_dif))), ...
      0.5*max(max(cmem_dif)));
    [x,y] = find(cmem_dif > threshold);
    d_i(frame) = mean(y) + j*mean(x);
    % compute neighboring node's angular orientation
    D = size(imager,1);
    phi_i(frame) = -atan(2*(real(d_i(frame)) ...
      - D/2)/D*tan(psi/2));
  end

  % increment frame counter
  frame = frame + 1;
  % ping-pong correlation memories
  pong = not(pong);
end

% close communications sessions
status  = image_sensor_api(SENSOR,'close',shandle);
status  = wireless_mote_api(MOTE,'close',mhandle);
```

ACKNOWLEDGMENT

The authors wish to thank the students of the Wireless Sensor Networks Laboratory (WSNL) at Stanford University for testing and debugging WiSNAP. The authors also would like to credit Agilent Technologies—in particular Bob Black, Pete Mahowald, and Jack Wenstrand—and Chipcon AS for providing hardware samples and technical support.

REFERENCES

[1] I. F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, August 2002.

[2] D. Culler, D. Estrin, and M. Srivastava, "Guest editors' introduction: Overview of sensor networks," *IEEE Computer Magazine*, vol. 37, no. 8, pp. 41–49, August 2004.

[3] P. V. Pahalawatta, T. N. Pappas, and A. K. Katsaggelos, "Optimal sensor selection for video-based target tracking in a wireless sensor network," in *Proc. International Conference on Image Processing (ICIP '04)* , October 2004, vol. 5, pp. 3073–3076.

[4] J. Elson, L. Girod, and D. Estrin, "EmStar: Development with high system visibility," *IEEE Wireless Communications Magazine*, vol. 11, no. 6, pp. 41–49, December 2004.

[5] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A wireless sensor network testbed," in *Proc. 4th International Conference on Information Processing in Sensor Networks (IPSN '05)*, April 2005, pp. 483–488.

[6] E. Welsh, W. Fish, and J. P. Frantz, "GNOMES: A testbed for low power heterogeneous wireless sensor networks," in *Proc. International Symposium on Circuits and Systems*, May 2003, vol. 4, pp. IV-836–VI-839.

[7] The MathWorks, "MATLAB® 7 - The Language of Technical Computing," *Product Information*. Available: https://tagteamdbserver.mathworks.com/ttserverroot/Download/18842_ML_91199v00.pdf

[8] Agilent Technologies, "Agilent VEE Pro 7.5," *Product Information*, 2005. Available: http://www.agilent.com/find/vee

[9] National Instruments, "LabVIEW™ 8," *Product Information*, 2005. Available: http://www.ni.com/labview/

[10] Wireless Sensor Networks Laboratory, Stanford University, "WiSNAP — Wireless Image Sensor Network Application Platform," *Web Site*. Available: http://wsnl.stanford.edu/projects.php

[11] Agilent Technologies, "Agilent ADCM-1670 CIF CMOS Camera Module," *Technical Specification*, Preliminary Draft 0.10, August 2002.

[12] Agilent Technologies, "Agilent ADNS-3060 High-Performance Optical Mouse Sensor," *Data Sheet*, October 2004. Available: http://cp.literature.agilent.com/litweb/pdf/5989-3421EN.pdf

[13] Chipcon AS, "SmartRF® CC2420DBK Demonstration Board Kit," *User Manual*, Revision 1.3, November 2004. Available: http://www.chipcon.com/files/CC2420DBK_User_Manual_1_3.pdf

[14] Logitech, "Logitech® MX™310 Optical Mouse," *Product Information*, 2005. Available: http://www.logitech.com/index.cfm/products/details/US/EN,CRID=2142,CONTENTID=6952

[15] Atmel Corporation, "ATmega128(L) 8-bit AVR® Microcontroller," *Data Sheet*, Revision 2467M-AVR-11/04, November 2004. Available: http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

[16] Chipcon AS, "SmartRF® CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver," *Preliminary Data Sheet*, Revision 1.2, June 2004. Available: http://www.chipcon.com/files/CC2420_Data_Sheet_1_2.pdf