Cyclops: In Situ Image Sensing and Interpretation in Wireless Sensor Networks

Mohammad Rahimi[†],Rick Baer[‡],Obimdinachi I. Iroezi[†], Juan C. Garcia[†],
Jay Warrior[‡],Deborah Estrin[†],Mani Srivastava[†]

[†]Center for Embedded Networked Sensing, UCLA, Los Angeles, CA 90095,USA
[‡]Agilent Technology Agilent Laboratories, 3500 Deer Creek Road, Palo Alto, CA 94304,USA

mhr@cens.ucla.edu,rick_baer@agilent.com,jay_warrior@agilent.com,destrin@cs.ucla.edu,mbs@ucla.edu

ABSTRACT

Despite their increasing sophistication, wireless sensor networks still do not exploit the most powerful of the human senses: vision. Indeed, vision provides humans with unmatched capabilities to distinguish objects and identify their importance. Our work seeks to provide sensor networks with similar capabilities by exploiting emerging, cheap, low-power and small form factor CMOS imaging technology. In fact, we can go beyond the stereo capabilities of human vision, and exploit the large scale of sensor networks to provide multiple, widely different perspectives of the physical phenomena.

To this end, we have developed a small camera device called Cyclops that bridges the gap between the computationally constrained wireless sensor nodes such as Motes, and CMOS imagers which, while low power and inexpensive, are nevertheless designed to mate with resource-rich hosts. Cyclops enables development of new class of vision applications that span across wireless sensor network. We describe our hardware and software architecture, its temporal and power characteristics and present some representative applications.

Categories and Subject Descriptors: B.0 [Hardware]: General; D.2.11 [Software]: Software Engineering: Software Architectures

General Terms: Performance, Design, Measurement, Experimentation

Keywords: Sensor Network, Vision, Imaging, CMOS Imaging, Power Efficiency

1. INTRODUCTION

Today as we make progress toward making sophisticated, reliable and low power sensor networks [1] [2], we still face difficulties in collecting useful data from our environment.

SenSys'05, November 2–4, 2005, San Diego, California, USA.

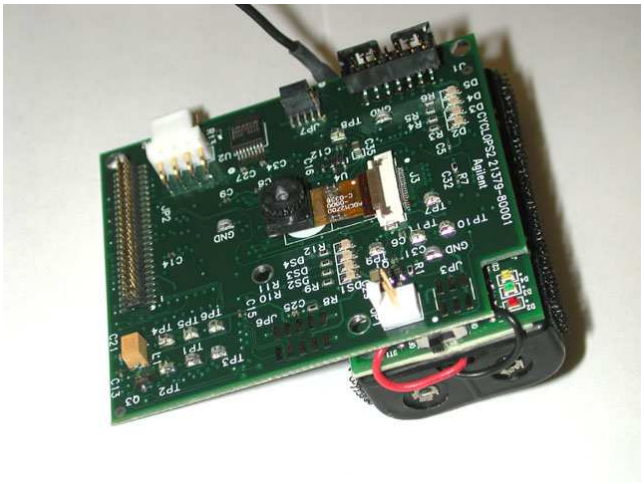


Figure 1: Cyclops is a smart vision sensor that will be controlled by a low power micro-controller host. This picture shows Cyclops with an attached MICA2 Mote [4]

Most *low power sensors* are constrained by limited coupling to their environment. While vision can reveal immense amount of information about the surrounding environment, there are serious obstacles for using vision in wireless sensor networks. Vision sensors are typically power hungry and vision algorithms are computationally expensive, with high power consumption requirements [3]. In wireless sensor networks, vision can only play a meaningful role if the power consumption of the network node is not significantly increased. Networks with a large number of vision-sensing nodes should utilize algorithms that consume less power. Scalability in number would *1) overcome occlusion effects, 2) provide multiple perspectives and 3) provide closer views of the phenomena (or objects).* This may enable the application of lightweight vision algorithms, with lossy inferences that can be reinforced through multiple observations and interpretation in the network.

On the sensor side,the emergence of integrated CMOS camera modules with low power consumption and moderate imaging quality [6] [7] brings such dreams closer to reality. These modules are widely used in electronic gadgets

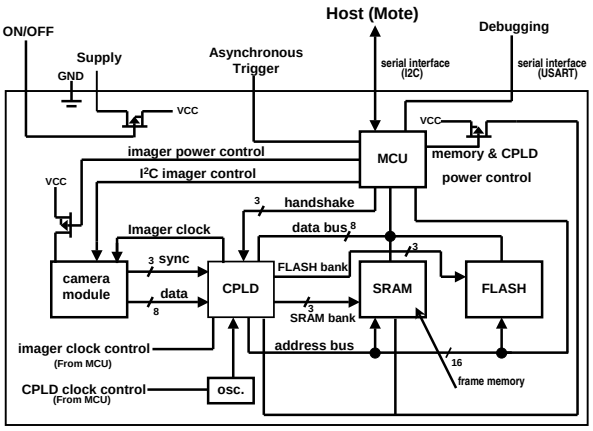


Figure 2: Diagram shows Hardware architecture of first generation of Cyclops.

such as cell phones and personal digital assistants (PDAs). They typically include a lens, an image sensor, and an image processing circuit in a small package. In spite of this high level of integration, these devices are still difficult to apply in lightweight sensor network nodes. Image data must be captured synchronously at high rates and programming is complex.

Cyclops (Fig-1) attempts to overcome these difficulties. It is an electronic interface between a camera module and a lightweight wireless host. Cyclops contains programmable logic and memory circuits for high-speed data transfer. It also contains a micro-controller (MCU) that presents a simple interface to the outside world. It isolates the camera module's requirement for high-speed data transfer from the low-speed capability of the embedded controller, providing still image frames at low rates. Since capturing and interpretation of an image is potentially time consuming, separate computational resources are used for imaging and communication. This is particularly important for network-enabled nodes which experience asynchronous events (e.g. MAC layer event) with stringent delay constraint requirements.

Cyclops power consumption is minimal to enable large scale deployment and extended lifetime. This results in extreme constraints in its computational power and imaging size. This makes Cyclops appealing for particular classes of applications. Any application that requires high speed processing or high resolution images should utilize a different imaging platform.

The rest of this paper describes the architecture of the Cyclops sensor and analyzes its characteristics. It further discuss Cyclops software framework, provides timing analysis of different operations and Cyclops energy requirements in its different states. Finally it presents two example applications that further illustrate Cyclops capability. Our future work will apply this architecture to distributed imaging applications.

2. HARDWARE

The main principles in Cyclops hardware design are:

- low power consumption, on the order of a sensor network node (e.g. Mote) [4] [5].
- simple interfacing that separates the complexity of the vision algorithm from the network node.
- adaptivity to make it a flexible sensor for applicability to a variety of sensor network problems.

To achieve this Cyclops exploits *1) Computational parallelism to isolate prolonged sensing computations 2) on-demand control of clocking resources to decrease power consumption 3) on-demand access to memory by using an external SRAM for storing image frames and 4) capability for automatic relaxation of each subsystem to its lower power state*. Next, we describe Cyclops hardware organization and its individual components.

2.1 Hardware Organization

Cyclops consists of an imager, a micro-controller (MCU), a complex programmable logic device (CPLD), an external SRAM and an external Flash (Fig-2). The MCU controls the Cyclops sensor. It can set the parameters of the imager, instruct the imager to capture a frame and run local computation on the image to produce an inference. The CPLD provides the high speed clock, synchronization and memory control that is required for image capture. The combination of the MCU and the CPLD provides the low power benefits of a typical MCU with on-demand access to high speed clocking through a CPLD. Furthermore, the CPLD can perform a limited amount of image processing such as background subtraction or frame differentiation at capture time. This results in extremely economical use of resources since the CPLD is already clocking at the capture time. When MCU does not need the CPLD services it halts its clock to minimize power consumption.

Cyclops uses external SRAM to increase the limited amount of internal MCU memory and provide the necessary memory for image storage and manipulation. In essence, the external memory provides us on-demand access to memory resources at the capture and computation time. The SRAM is kept in sleep state when the memory resources are not needed. In addition, Cyclops has an external Flash memory. The Flash memory provides permanent data storage for functions such as template matching or local file storage.

The MCU and CPLD and both memories share a common address and data bus. This facilitates easy data transfer between the imager, SRAM and FLASH memory but it also requires an appropriate mechanism that guarantees synchronized access to such shared resources. This will be further described in Cyclops firmware discussion.

Each module in Cyclops has several power states. Typically the lowest power consumption sleep states have the highest wake-up cost. Consequently, the application should choose the sleep level based on its amortized cost. In addition Cyclops has an *Asynchronous Trigger* input. This can be used as a *paging channel* in applications that are characterized by prolonged sleep times and quick response to asynchronous ambient variations. The paging channel can be connected to sensors of other modalities for event triggering. For example it could be connected to a passive IR detector or a microphone to trigger image capture on motion, or connected to a magnet sensor to monitor a transition (e.g. opening and closing gates). Next we describe each component more elaborately.

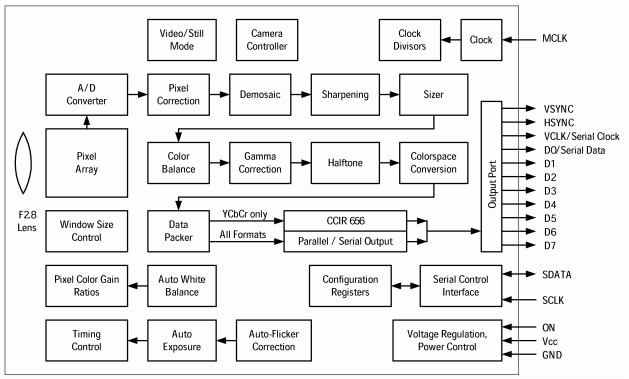


Figure 3: ADCM-1700 camera module block digram. It consist of an F2.8 lens, image sensor and digitizer, image processing units and data communication units.

2.2 Imager

The main requirement for the camera module is low power consumption. We chose a medium quality imager with moderate clock speed requirements and extremely low power consumption in its sleep state. It is an ultra compact CIF resolution CMOS camera module (ADCM-1700) from Agilent Technology. It combines an Agilent CMOS image sensor and processor with a high quality lens. The lens has a focal length of 2.10mm with practical focal lenght of almost 100mm to infinity and has 52 deg field of view. Its clock frequency can be set to anywhere from 4 to 32 MHz. We set the imager clock to 4MHz to provide the CPLD (which operates at 16MHz clock), enough time to grab an image pixel and copy it into the memory. The image sensor (Fig-3) has CIF resolution (352×288). The output of the image array is digitized by a set of ADCs to generate the raw image data. The camera module contains a complete image processing pipeline. The pipeline performs demosaicing (reconstruction), image size scaling, color correction,tone correction and color space conversion. It also implements automatic exposure control and automatic white balancing. Cyclops generally operates on images of reduced resolution because of its constrained memory and power.

The camera module has two parameters for adjusting the size of the image. Input window size determines the array of sensor pixels selected from the total number of available pixels (352×288) and output window size determines the number of pixels in the output image. When the input window size is greater than the output window size, the camera module averages the input pixels to generate the result. The image output is normally taken from the center of the field of view of the sensing array. An offset can be adjusted to pan the region of interest across the field of view.

The image processing unit is capable of generating variety of image formats. Currently Cyclops supports three image formats of 8-bit monochrome, 24-bit RGB color, and 16-bit YCbCr color. The camera module is programmable through a synchronous serial I2C port. Many low level parameters such as exposure time and color balance may be accessed. Data is output over a parallel bus consisting of eight data lines and three synchronization lines.

2.3 Micro-controller

The MCU is responsible for controlling Cyclops, communicating with the host and performing image inference. *Most notably, by providing separate processing power, it isolates sensing and networking computations from each other.* This is very important since Cyclops computations are potentially prolonged and might block highly asynchronous network operations. The Cyclops micro-controller(MCU) is an ATMEL ATmega128L [8]. It has an 8-bit RISC core processor with a 16-bit address Harvard architecture. It has an arithmetic logic unit that provides a multiplier supporting both signed/unsigned multiplication and fractional format. It provides general purpose registers, internal oscillators, timers, UARTs and other peripherals. In Cyclops, the MCU operates at $7.3728MHz$ and $3.3V$.

The system is very memory constrained: it has 128KB of Flash program memory and 4KB of SRAM data memory. The MCU is designed such that it can map 60KB of external memory into its memory space. The combination of the internal and external memory presents a continuous and cohesive memory space. External memory accesses are slower, but this is invisible from a programming perspective. The external SRAM is referred to as *extended Memory* since it extends the Cyclops memory to the maximum addressable 64KB.

The processor integrates a set of timers which can be configured to generate interrupts at regular intervals. This can be used to make Cyclops a synchronous sensor or an asynchronous sensor exploiting an underlying synchronous sampling mechanism. The MCU has two clock domains, the main clock for processor and I/O, and external timer clock for independent timer and clock operation. It has six sleep modes: idle, ADC noise reduction, power-down, power-save, standby and extended standby. Typically Cyclops is either *active*, in *power-save(PS)* or in *power-down(PD)* mode. In power-down mode, MCU shuts everything but the I2C and asynchronous interrupt logic necessary for wake up. Power-save mode is similar to the power down mode, but leaves an asynchronous timer running. This enables Cyclops to go into a deep sleep mode leaving an underlying synchronous timer operational for periodic sensing services.

2.4 CPLD

Image capture requires faster data transfer and address generation than what a lightweight processor can provide. To satisfy this requirement we use a *complex programmable logic device* (CPLD) as a lightweight frame grabber. The CPLD provides *1) on-demand access to high speed clocking at capture time and 2) (potentially) computation as image capture is underway* (e.g. calculating statistics such as histogram at capture time). The use of an FPGA was rejected because the high power required during initial configuration would have increased the amortized cost of the power-down state. Provisions have been provided to further save power by disabling the CPLD clock and by removing power from the entire memory subsystem (CPLD and memories).

A Xilinx XC2C256 CoolRunner CPLD was chosen for its low power consumption. This device consists of sixteen Function Blocks inter-connected by a low power Advanced Interconnect Matrix (AIM). The AIM feeds 40 true and complement inputs to each Function Block. The Function Blocks consist of a 40 by 56 Programmable Logic Array (PLA) and 16 macrocells which contain numerous configu-

ration bits that allow for combinational or registered modes of operation. It has 256-macrocell device and 80 user I/Os. The CPLD runs at $1.8V$, obtained from a dedicated voltage regulator, and has quiescent current of as low as $13\mu A$. It runs from a 16MHZ input clock and feeds a 4MHz clock to the imager at the capture time. We picked the CPLD's clock to be higher than the imager clock to allow a certain amount of image processing on the CPLD at image capturing time.

2.5 SRAM and FLASH

Cyclops needs memory for image buffering and local inference processing. It uses external CMOS static random access memory (SRAM) which provides 64KB [1] in 8bits format (TC55VCM208A from TOSHIBA). The device operates from a single $2.3V$ to $3.6V$ power supply to support extended battery operation. It provides both high speed and low power at an operating current of $3mA/MHz$ and a minimum cycle time of $40ns$. *The memory is automatically placed in low-power mode at* $0.7\mu A$ *standby current when it is in not in use.*

In addition, Cyclops has 512KB of CMOS Flash programmable and erasable read only memory (AT29BV040A from ATMEL) which is organized in 8bits format. This provides Cyclops with permanent storage to save information (e.g. templates). It operates from $2.7V$ to $3.6V$ for extended battery operation and *has auto-sleep functionality of* $40\mu A$ *current when it is in no use.*

2.6 Other Components

Cyclops has three MOSFET switches for power control. One switch can completely turn off the device, the second switch activates the memory subsystem (SRAM, CPLD and Flash) and the third switch activates the imager. These switches can override the normal auto-shutdown procedure of the devices to achieve further reduction in power consumption.

3. SOFTWARE

The motivation for our Cyclops firmware design is:

- transparency in using resources (such as the imager or SRAM) while supporting their automatic relaxation to the lowest possible power state.
- supporting the long computations that are needed for image operations.
- supporting synchronized access by both the MCU and the CPLD to shared resources such as SRAM and the imager.

This implies that, *unlike a network-centeric approach that calls for handling a high degree of network asynchronous events, the Cyclops architecture should target sequential image capture and processing.* Thus following a frame capture, Cyclops calls a series of potentially long synchronous operations with little concurrency.

[1]Since it is difficult to obtain memory chips with less than 512KB of capacity, we used SRAM and FLASH memories which are 512KB and we have organized each of the memories as eight 64kB banks, with the bank address selected by the CPLD. We are currently using only the lowest bank in each device. This obviously has additional power cost due to lack of the off-the-shelf 64KB memory.

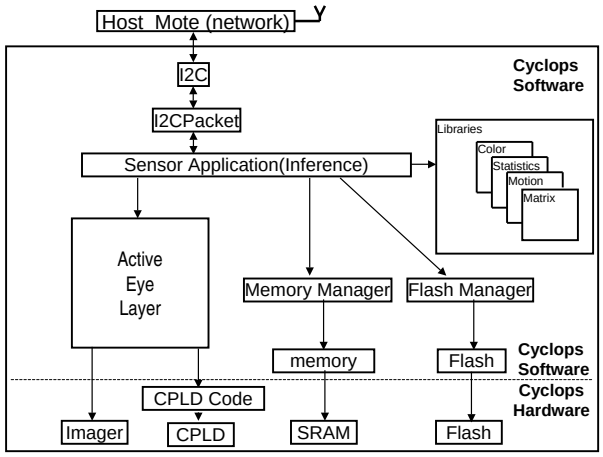


Figure 4: Component layouts in Cyclops. The components consist of the main sensing application, the libraries that encapsulate image manipulation logic and the devices that enable communication with hardware components.

To efficiently utilize Cyclops constrained resources, Cyclops firmware is written in the nesC [9] language and runs in TinyOS [10] operating system environment. This helps the Cyclops software architecture to abstract functionalities in the form of components with clear interfaces and exploit standard TinyOS scheduler and services. Next, we describe Cyclops software organization, its components and its image capturing and control stack (i.e. Active Eye).

3.1 Software Organization

There are three sets of components in Cyclops (Fig-4): drivers, libraries and the sensor Application. Drivers are the set of components that enable the MCU to communicate with peripherals. Examples of drivers are the imager setting and capturing drivers, CPLD drivers, Flash memory drivers and drivers that enable communication with the host Mote. There are two different class of libraries: primitive structural analysis libraries and high level algorithmic libraries. Among the very important structural libraries are the variety of matrix operation, statistics and histogram operation. These libraries are key to processing raw images. Other libraries are closer to application (algorithm) such as background subtraction, object detection or motion vector recognition libraries that are designed to do specific high level tasks. Finally the sensor Application is the main sensing component on Cyclops. It receives commands from the host and is responsible for performing necessary sensing operations as requested by the host. It is the sensing application that makes Cyclops a smart sensor.

In Cyclops, external address and data bus are shared among many entities including MCU, SRAM, FLASH and CPLD. Consequently devices may contend for bus access. The SRAM and Flash are slave devices, hence a mechanism is only necessary for the synchronization of MCU and CPLD bus access. We modified TinyOS scheduler to support Direct Memory Access for such synchronized bus usage. When CPLD needs access to the bus (e.g. capturing an image) the CPLD driver requests the scheduler a permit to perform a

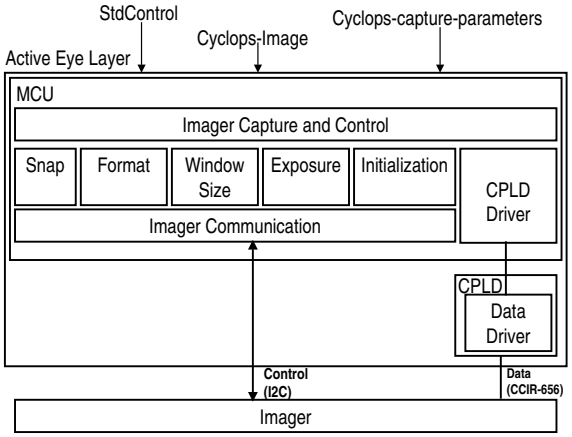


Figure 5: The Active Eye layer contains the image capture and control stack. It consists of components that control the imager and the CPLD. The CPLD modules consist of the CPLD driver inside the MCU and the CPLD data driver in the CPLD.

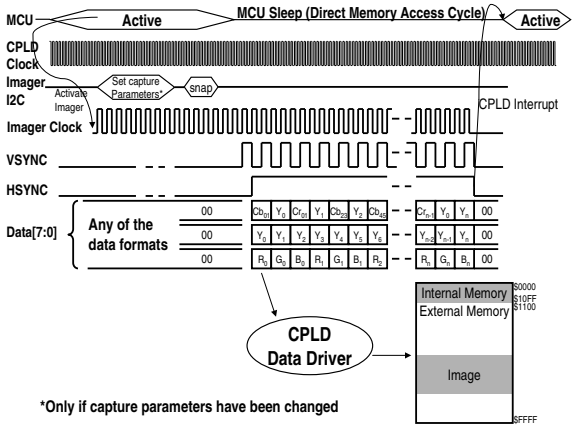


Figure 6: This picture shows the flow of data at capture time. Active Eye enables the camera module clock, sets capture parameters and snaps the image. It then requests a DMA transaction from the scheduler and the MCU goes to sleep when the transaction is granted. The camera module provides data via a parallel bus that includes vertical and horizontal synchronization signals. The CPLD receives the image data stream and saves it into memory. Upon capture of a complete frame, the CPLD notifies the MCU and a normal MCU operation resumes.

Direct Memory Access (DMA) transaction. On completion of the current active task such a permit is granted by the scheduler and CPU goes to power-save mode (power-down if there is no active timer). At the end of the transaction, the CPLD will notify the scheduler that it may resume normal operation. In essence there are two mechanisms that guarantees safe access of each component to memory: *1) the default mode in which the MCU controls the SRAM memory and 2) the DMA mode in which the scheduler sleeps.* This combination means that any external memory access in the TinyOS task routines happens while the MCU has access to the SRAM. Libraries should be implemented in task context and this leverages access to the external memory without any additional synchronization requirement.

3.2 Active Eye: Image Capturing and Control Layer

The Active Eye Layer (AEL) enables Cyclops to see the external world . It is the Cyclops image capture and control stack and is responsible for communicating with the imager, setting its parameters, keeping it in proper state, and finally capturing the image. It consists of different components that exist in the MCU and CPLD (Fig-5). The rest of this section describes Active Eye components and their interactions.

The main component in the AEL is the image capture and control module. It uses subordinate control modules for controlling the imager. These modules are responsible for changing different aspects of the image. Examples of these modules are a window size module that is responsible for setting the input and output size window; a format module which is responsible for setting image format (i.e. monochrome, RGB color, etc); and an exposure module which sets capture exposure parameters. All these modules use the imager communication component to access the camera over the imager I2C bus.

The AEL uses the CPLD for image data capture and manipulation. The CPLD modules consist of the CPLD driver inside the MCU and data driver inside the CPLD. The CPLD data driver module is written in Verilog Hardware Descriptive Language. It is essentially a state machine with different states trajectories that are selected by micro-controller commands. The micro-controller places the CPLD on the proper state by programming the CPLD's internal registers and by asserting control over CPLD handshake lines. To capture an image, the AEL first enables the imager module clock and updates the capture parameters (if they have been modified). It further places the camera module in video output mode and sets the CPLD state to capture mode. Upon completion, the CPLD notifies the MCU that it may resume operation and relaxes to standby mode. Fig-6 illustrates this process.

There are three interfaces that facilitate access to AEL: "StdControl", "snap" and "captureParameters". The "StdControl" enables initialization, starting and stopping of the AEL stack. At starting routine AEL turns ON imager power supply, waits for a predetermined time for the imager to be accessible (i.e. 200ms) and sets the initialization parameters in the imager via its I2C bus. Starting the AEL should happen only once, AEL then always relaxes to its lowest possible state by disabling imager clock when it is not needed. Still, application can stop Active Eye Layer in cases with extremely low duty cycling to further reduce power consumption. In such cases they should start AEL before using it again.

The second interface is "captureParameters" which enables the user to provide AEL with new capture conditions. Capture parameters should only set once and will be preserved afterward. This means that in constant capture conditions this interface would be called once. Cyclops capture parameters structure is as follows:

```
typedef struct CYCLOPS_Capture_Parameters
{
     wpos_t      offset;
     wsize_t     inputSize;
     uint8_t     testMode;
     uint16_t    exposurePeriod;
     color8_t    analogGain;
     color16_t   digitalGain;
     uint16_t    runTime;
}CYCLOPS_Capture_Parameters;
```

The input size (a two element structure) determines the size of of the input region of interest (ROI). In practice these values can be between 24-288 for image height and 24-352 for its width. The offset (another two element structure) determines the position of the ROI within the field of view (FOV), relative to the center of the array. Modification of this parameter enables the user to pan across the image sensor window. The test mode parameter can be used to obtain a synthetic test image. The exposure period controls the duration of the exposure. The analog and digital gains control the amount of gain that is applied to each color channel. The camera module can also determine the exposure and gain values automatically. The run time parameter causes the camera to run for an additional period before capture begins in order to allow the auto functions to converge.

The third interface is "snap" which should be called each time an image capture is required. To snap the image the desired snap condition such as image type, its height and width, and its memory location should be set in the image data structure. The snap interface then returns the image, filling the desired memory location with the image content. Cyclops image data structure is as follows:

```
typedef struct CYCLOPS_Image
{
    uint8_t  type;
    uint16_t size;
    uint8_t nFrames;
    uint8_t *imageData;
    bufferPtr imageHandle;
}CYCLOPS_Image;
```

The type specifies the image format (i.e. monochrome, RGB, etc). The size is the output image size. "nFrames" specifies the number of sequential frames to capture in memory. The "imageData" is a pointer to the image location in memory. The imageHandle is a memory handler (pointer to memory pointer) when dynamic memory is available. Currently we allocate memory statically.

3.3 Libraries

TinyOS does not provide a preemptive mechanism in its synchronous execution model (i.e., tasks cannot preempt tasks). This means that a synchronous task will run to completion when not preempted by an interrupt handler. Consequently, long running tasks can block access to computational resources, thereby starving other tasks. A basic design principle of TinyOS is that long-running tasks should be broken down into smaller tasks that are posted sequentially. In this manner, a large block of processing can explicitly yield the processor at predetermined safe points during its execution to promote fairness and prevent starvation. In network-enabled sensor nodes, task-yielding is particularly important to cope with asynchronous requirements of the network

Image processing operations are typically long-running and not suitable for sequential decomposition. The consequence of loading a sensor network node with processing the image is a significant increase in implementation complexity to break long image processing operations into pieces. This in turn results in reduction of performance of the system. In our case, due to existence of dedicated processor we build an execution model that supports continent serialization of image processing operations. When an image is captured, one expects a series of potentially long computational blocks with limited concurrency requirements. This gives Cyclops the liberty of using prolonged computations in the TinyOS synchronous model. Cyclops libraries are implemented in long tasks with potential delays of hundreds of milliseconds. In addition since the libraries are not pending for any event they do not use the TinyOS *split-phase* call-back model. This means that the result of their computation is readily available at the completion of their execution. This greatly simplifies implementation of any algorithm using Cyclops libraries.

Library	Operation
Matrix	Image to Matrix Conversion
Matrix	Matrix to Image Conversion
Matrix	Extracting a particular plane in an image e.g. red plane in RGB or Cb in YCbCr
Matrix	Extract Row
Matrix	Extract Column
Matrix	Threshold
Matrix	Clone Matrix
Logic	matrix AND operation
Logic	matrix OR operation
Logic	matrix XOR operation
Logic	matrix-wide shift right operation
Logic	matrix-wide shift left operation
Arithmetic	matrix Addition
Arithmetic	matrix Subtraction
Arithmetic	matrix Scaling
Statistics	minimum,max,summation
Histogram	histogram generation
Sobel	Matrix derivative vs. hight and width
Template	Provides hollow or solid rectangles of different sizes

Table 1: This table shows different primitive libraries and the operations that they currently support. These operations are supported on matrices of different depth such as one, eight or sixteen, when applicable.

Libraries are divided in two categories: primitive structural libraries such as matrix operation libraries or histogram libraries and advanced libraries such as coordinate conversion and background subtraction. We next briefly describe some of the Cyclops libraries.

3.3.1 Primitive Structural Libraries

An important data structure in any image analysis is matrix operation. Cyclops supports matrices of different depth of one, eight or sixteen. A matrix is defined as:

```
typedef struct{
    uint8_t depth;
    union
    {
        uint8_t* ptr8;
        uint16_t* ptr16;
```

```
    }data;
    uint16_t rows;
    uint16_t cols;
} CYCLOPS_Matrix;
```

where depth determines the size of elements of the matrix (i.e. 1,8,16), data is the location of the memory that matrix resides and rows and cols are the number of rows and columns in the matrix. Any library that implements a matrix operation typically supports matrices of different depth. For example MatrixArithmetic library supports addition of matrices of depth 8 and 16 (for depth 1 it is not defined) and MatrixLogic supports matrices of depth 1,8 and 16. In addition, the default assumption is that values are clipped when they overflow the depth of the output matrix.

Table-1 shows the current set of supported libraries and some of their associated operations. The Matrix library enables primitive matrix operations such as conversion of image to a matrix (and reverse). In addition, it enables extracting of particular planes of an image such as individual RGB or YCbCr in a form of a matrix and has primitives for extraction of particular rows or columns of a matrix. Histogram library generates histogram of the input matrix. To adapt histogram operation to constrained Cyclops environment, it can generate only histograms in 4,8,16 or 32 breaking points. This results in efficient implementation of histogram library. The Sobel library provide derivative of a matrix vs. its rows or columns. It is implemented using Sobel algorithm. This library is useful in primitive edge detection applications. Template library provides matrices with determined patterns. Currently, only rectangular templates are implemented.

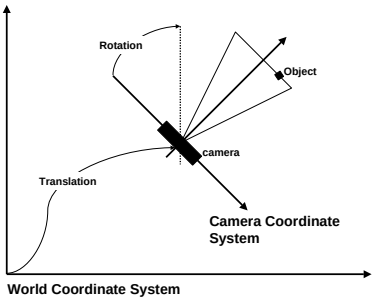
3.3.2 Advanced Libraries

- Background Subtraction: In many applications, objects appear on a largely stable background. Background subtraction is important in many image analysis and manipulation [3]. The main issue is obtaining a good estimate of the background in the face of changes in illumination, moving objects and gradual environmental variation. One method that usually works fairly well is to estimate the value of background pixels using a *moving average* filter. In this approach we estimate the value of the background image as a weighted average of the previous background and the instantaneous image. More precisely:

  $$B_n = \lambda \times B_{n-1} + (1 - \lambda) \times Img$$

  Where B_n is the background image at nth iteration of the algorithm, B_{n-1} is previous background image and Img is the instantaneous image. In our case,to further enhance the performance of the library we pick $\lambda = 0.25$ to be able to implement that efficiently with shift and addition operations.

- Co-ordinate Conversion: In any network based application that uses multiple numbers of Cyclops, the result of individual Cyclops inference should be interchanged across the network. In such cases these results should be interpreted in a unified world coordinate system. This library facilitates changing coordinate system from the Cyclops coordinate system to a world coordinate system (and reverse). The library currently support 2-dimensional coordinate conversion for translation and rotational adjustment.



4. PERFORMANCE ANALYSIS

Our analysis of the Cyclops platform focuses on the platform's power consumption and its temporal characteristics. Table-2 shows different Cyclops power states, their associated active clock domains, and the sources that can affect Cyclops transition in that state. It also shows *measured* power consumption of Cyclops in each state. Cyclops normal states are 1)Image capture, 2)Extended memory access, 3)Permanent memory access, 4)Microcontroller internal and 5)Sleep. During normal operation, based on I/O requests (i.e. transition sources), Cyclops transparently makes the resources available and upon releasing those resources,it automatically relaxes to the lowest possible state. This process lets the Cyclops be in the sleep state when no resources are in use.

In addition, Cyclops has two other sleep states, OFF and Shutdown. In the OFF state Cyclops' main MOSFET switch is off and the device is completely turned off. In this state Cyclops can not be programmed until it is turned ON by the host. The Shutdown state is the same as sleep state except that Cyclops shuts down its external memory subsystem. Since external peripherals are OFF, the contents of the SRAM are not preserved. Further reduction in power (i.e. shutdown and OFF states) requires an explicit instruction from application since the contents of the memory will be lost in those states.

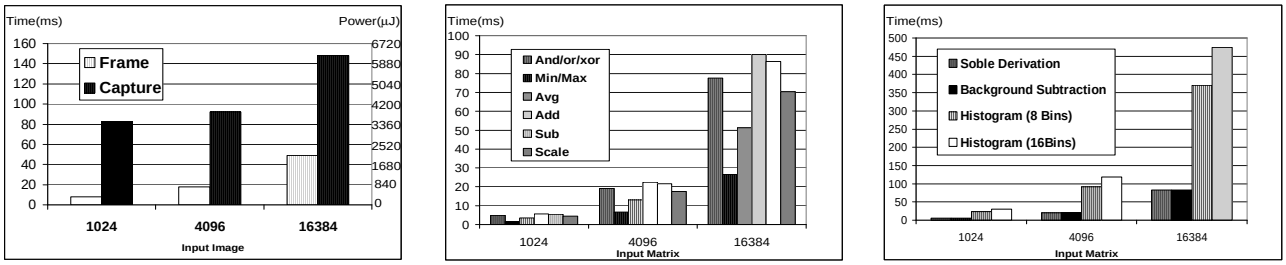
In practice, Cyclops energy consumption depends on the power consumption of the different states and their time duration. In the case of the imager, the image capturing time depends mainly on two parameters: *1) the input image size and 2) the ambient light intensity.* Input image size determines the size of sensor array and the subsequent number of necessary operations to convert each pixel to a digital value. The ambient light intensity determines the necessary exposure time of the sensor array. Fig-7(a) shows the effect of input image size on capture operation for room light intensity. In all these cases we used the imager's automatic exposure feature. In our implementation the CPLD needs two video frames to synchronize with the imager video stream. This increases the cost of capture operation significantly.

Fig-7(b) shows Cyclops timing for some structural libraries and Fig-7(c) shows Cyclops timing for some of our advanced libraries for different input matrix sizes. In our case, the timing of these libraries are dominantly determined (almost linearly) by input matrix size. Most of these operations happen in Cyclops extended memory access mode.

Table-3 shows power consumption of some known host nodes [11]. It shows that power consumption of Cyclops in different states is comparable to these host nodes. In

Mode	Components State					Active Clock Domains				Transition Source					Power
	MCU	SRAM	FLASH	Image	CPLD	MCU	ASY.	CPLD	Imager	MCU	I2C	Timer	CPLD	Trig.	
Capture Image	PD/PS†	√	S‡	√	√		√	√	√		√	√††	√	√	42mW
Extended Memory Access	√	√	S	S	S	√	√	√		√	√	√		√	W=53mW R=50mW
Permanent Memory Access	√	S	√	S	√	√	√	√		√	√	√		√	W=64.8mW R=28mW
MCU Internal	√	S	S	S	S	√	√			√	√	√		√	23mW
Sleep	PD/PS	S	S	S	S		√				√	√††		√	0.7mW
Shutdown	PD/PS	OFF	OFF	OFF	OFF		√				√	√††		√	60μW
OFF	OFF	OFF	OFF	OFF	OFF										$\leq$ 1μW

Table 2: This table shows Cyclops' different states, their associated active clock domain, the sources that can cause a transition out of that state, and the measured power consumption of the device. †) "PD" is power-down and "PS" is power-save mode. The measurements reported are for "PD" mode.‡"S" stands for Sleep †† Timer interrupt only is in effect if Cyclops is in PS mode.



(a) (b) (c)

Figure 7: This figure shows the timing cost of different operations in Cyclops, a) frame time and capture time of images of different sizes, where frame time is imager frame time and capture time is Cyclops capture time (Active Eye delay). In all the three cases the input and output window sizes are the same, b) structural library operations c) performing more advanced operations.

Operation	Telos	Mica2	MicaZ
Mote Standby (RTC on)	15.3μW	57.0μW	81.0μW
MCU Idle	163.5μW	9.6mW	9.6mW
MCU Active	5.4mW	24.0mW	24.0mW
MCU + Radio RX	65.4mW	45.3mW	69.9mW
MCU + Radio TX (0dBm)	58.3mW	76.2mW	63.0mW

Table 3: Power consumption of some of the of well known sensor network host nodes. These numbers are cited from [11] for a 3V battery

particular, Cyclops sleep power is comparable to a Mote's Standby power and its image capture power is comparable to the host's power in radio operation.

There are various improvement that we are considering for the Cyclops imaging stack. Improving the CPLD's synchronization with the imager will improve the timing (and energy cost) of the capture significantly. In addition using more parallelism in the CPLD logic will reduce the the number of the CPLD's clock cycles for performing a pixel transfer to the SRAM. This would let us increase the imager clock speed and facilitate faster image capture operation.

5. PUTTING IT ALL TOGETHER

Now, that we have shown a few sample components, we will examine their composition and interaction within an application.

5.1 Design Principles

In designing an algorithm for Cyclops, we try to achieve a *low data profile* to be able to use its limited computational and memory resources efficiently. In our experience, the key issues in the development of a Cyclops application are:

- *Data reduction:* we keep the image size at minimum size to the extent that application warrants it. This can be achieved by selecting the proper image aspect ratio and output resolution.

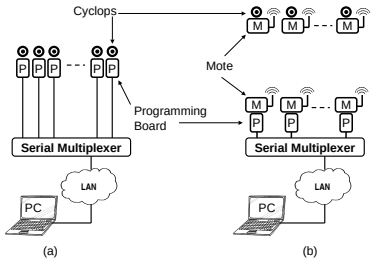
- *Efficiency in Computational Arrangements:* Applications may have different computational arrangements for performing a particular inference on an image. We designed our algorithm such that the arrangement of its computations, results in rapid reduction of the data size along the flow of its execution.

- *Static Computation*: When possible, we perform a computation and reuse its result(s) multiple times. For example results of trigonometric functions for coordinate conversion is saved for later reuse. In addition many libraries embed computation in the form of static lookup tables to trade program memory space with computational speed.

We now discuss our test environment and two specific applications. While these applications do not present a comprehensive algorithm and analysis study, they illustrate the capability and applicability of Cyclops.

5.2 Test Environment

Cyclops has a serial port that connects it directly to PC. We have developed a TinyOS component (called serialDump) that enables an application to dump particular locations of memory into serial port. For example an application that tests a particular image processing library may dump the input image as well as output results into a PC. This can be used for verification of performance of the algorithms on Cyclops. In cases that the algorithm includes more than one Cyclops, one can use a serial expansion module to connect multiple Cyclops directly to a PC. In addition there is a wireless version of serial debugging component (called radioDump) that enables Cyclops to dump its memory content via an RF radio link. In this case one Mote is attached to Cyclops and another Mote is attached to a MIB510 programming and serial communication board (from xbow technology [4]). In this case several wireless Mote pairs can be operated simultaneously on different frequencies.



In our experience, it is often more convenient to develop image processing routines within a high level signal processing language. In such cases we save a series of image frames from Cyclops on a host PC. Once the algorithms are developed, they can be ported to Cyclops. In practice, we found that these debugging capabilities provide us with *1) a rich set of data to design an algorithm in high level languages (i.e. matlab) 2) a high degree of visibility into different stages of an algorithm when it is ported into Cyclops*. We now describe the two applications that we have explored.

5.3 Object detection

Identifying moving objects from a sequence of images is a fundamental and critical task in many sensing applications. A common approach is to perform background subtraction, which identifies moving objects from the portion of an image frame that differs significantly from a background model. There are many challenges in developing a good background subtraction algorithm. First, it must be robust against changes in illumination. Second, it should avoid detecting non-stationary background objects such as moving leaves, rain, snow, and shadows cast by moving objects. Finally, its internal background model should react quickly to changes in background such as starting and stopping of vehicles.

There are many ways to approach this problem and we chose the basic method [12] to minimize computation requirements. In our case (Fig-8) Cyclops periodically captures images from the outside world and constructs the stationary background using the moving average of the series of

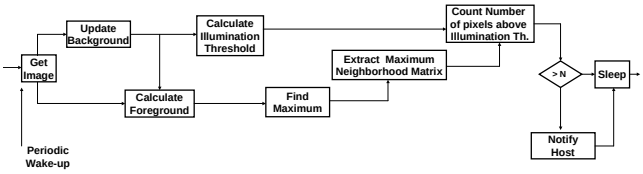


Figure 8: Diagram of Cyclops object detection algorithm.

the images. It further constructs a model of the foreground based on the absolute difference of the instantaneous image and the constructed background. Here we consider the absolute value to accommodate the fact that objects may be darker or brighter than the background. When there is no object in the scene, such differences are minimal but presence of an object leads to substantial variation in the foreground. Using a threshold filter we further construct the object map. In practice, the absolute value of these variation depends on the field illumination. We constantly estimate the intensity of illumination field and adjust the threshold value as a factor of background illumination. Finally we apply a blob filter by counting the number of pixels that exceed a threshold count. In our scheme, we initially find the location of maximum value of the object map and extract a sub-matrix of that neighborhood. This reduces the data size significantly with the expense of losing other objects in the foreground. This means that we only detect the object with the biggest impact on the foreground image.

To test performance of our algorithm we deployed Cyclops outside, in UCLA campus. In our case Cyclops periodically sleeps for 15Sec intervals and executes the algorithm after each wake-up. Cyclops then constantly updates an attached computer with its instantaneous image as well as its instantaneous background and foreground images. In addition it notifies the attached computer if it has detected an object in the scene. Fig-9 shows our result for six consecutive observations. Here the top row is instantaneous Cyclops images, following by its updated background and foreground image. The bottom row shows Cyclops foreground map in a perspective view. In our experiments Cyclops was 78.4% right and in 21.6% of cases it reported a false/positive or false negative. We also measured the execution time of our object detection algorithm and it runs in $240ms$,$60.8ms$ and $16.8ms$ for image sizes of 128×128, 64×64 and 32×32 respectively.

While these experiments are far from comprehensive, they showcase the potential capabilities of Cyclops. In our experience in many cases the background model was polluted with the foreground objects in the scene. To remedy this problem we can drop the consecutive frames with the same object locations. In addition we can add a feedback from the object detection algorithm to the background updating model to further reduce such pollution effect. In our experiments the image sizes are 128×128 pixels. In practice the aspect ratio of the scene can be proportional to the expected object trajectory. For example if the goal is detecting humans, we do not expect presence of a human in the upper side of an image and we can use images with modified aspect ratio to further reduce data size and the execution time of the algorithm. Finally our blob filter(s) can be proportional to the aspect ratio of potential objects (i.e. human body).

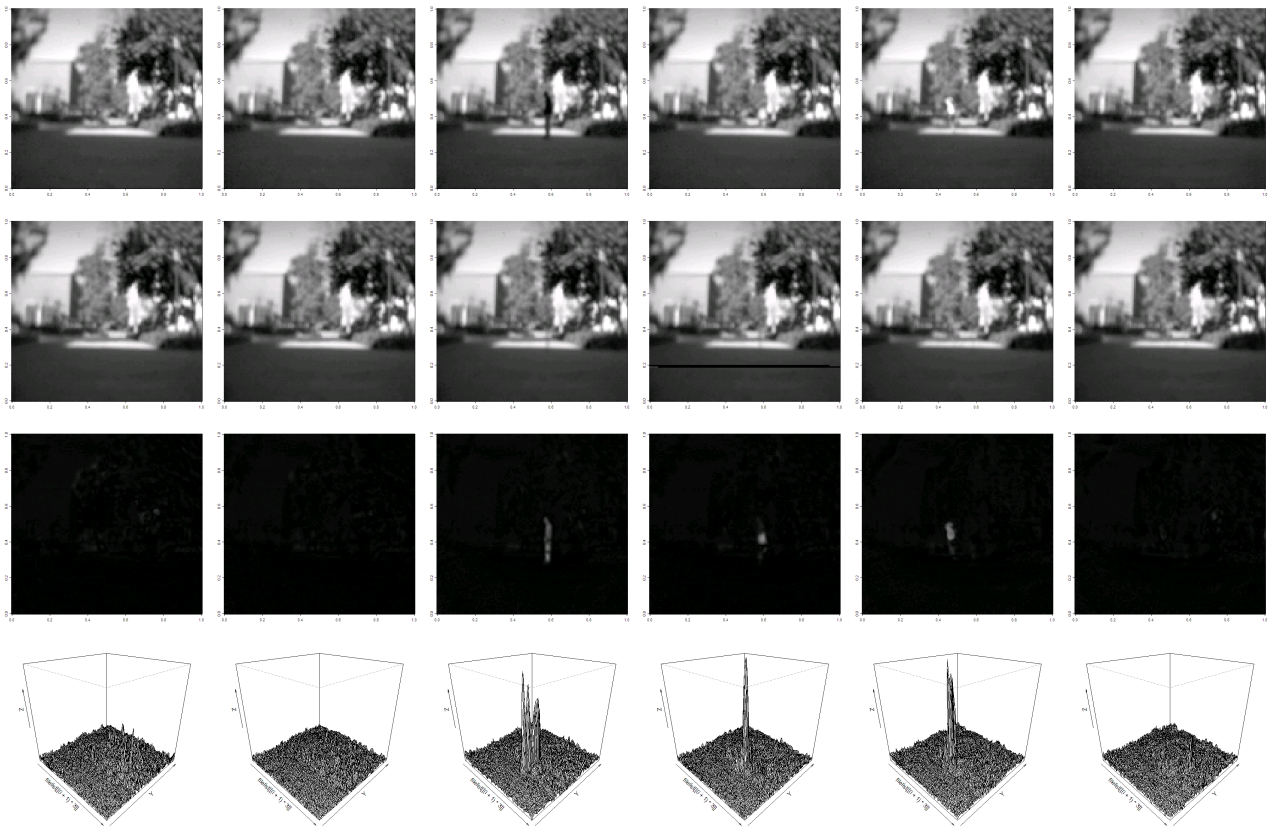


Figure 9: In this figure the rows from the top are: I) Cyclops instantaneous images, II) its constructed background, III) its constructed foreground and IV) a perspective plot of the foreground. This data has been generated by constantly dumping the three images from Cyclops to an attached logging computer.

5.4 Hand Posture Recognition

Visual interpretation of hand gesture is a convenient method for Human-computer-interaction (HCI). It can be exploited for contact-less command and control of devices and systems. There are two categories of such interactions: static and dynamic. A static gesture is a particular hand configuration and pose represented by a single image. A dynamic gesture is a moving gesture, represented by a sequence of images. Here, we focus on applicability of Cyclops for recognition of static gestures.

Gesture recognition is a very hard problem particularly when the hand occupies only a small portion of the image. In most cases, the hand must dominate the image for simple statistical techniques to work. In our case, if Cyclops is being used in a network, we can assume that due to exsitance of a netowrk link it can be kept at close vicinity of the human hand. Even at such close distances designing a vision algorithm that performs reliably for different people, with different illumination conditions, and with translational changes is challenging. In addition the algorithm should run fast enough so that the user senses no appreciable delay between making a gesture and the response of the system. While there are variants of algorithms in the literature, we have chosen a technique based on the orientation histogram of the image [13] [14]. The main incentive for choosing this technique is achieving high speed performance with reasonable quality (i.e. limited variant of pose). In our case, Cyclops performs pattern recognition using a transform that converts an image into a feature vector, which will then be compared with the feature vectors of a training set of gestures. We use the Euclidean distance of the features of the input gesture vs. a set of trained features and find the closest candidate. Our transform is orientation histogram:

$$hist(arctan(\frac{dImg/dx}{dImg/dy}))$$

where the orientation term gives robustness to illumination variation by calculating dominant orientation texture of the image and histogram term provides translational invariance. Fig-10 shows block diagram of our system. In our case we implemented derivative components using Sobel derivative mask and to further speed up the calculation we used the absolute value of the Sobel transform. In addition we used a look-up table for "arctan" operation for faster response time. We used Cyclops histogram library with 32 braking point for calculating orientation histogram. To avoid any floating point operation we kept all intermediate matrices in 8bit depth integer format. In addition to increase the performance of our computation we kept the result of each computation within maximum dynamic range of the following block of computation. For example since the value of "arctan" operation is changing between 0 to 1.570696, in the implementation of its look-up table, we calibrate its value to be between 0-255. This improves the performance of the

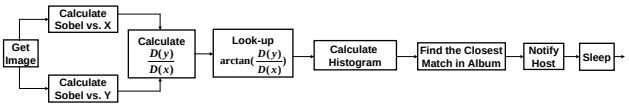


Figure 10: Diagram of Cyclops gesture recognition algorithm.

histogram operation which has an input dynamic range of 0-255.

To test the performance of our algorithm, we applied Cyclops to an album of 10 static hand gestures from a subset of American Sign Language (ASL). We further divided that album into training and test data sets. We calculated the orientation histogram of the training images, using their average as representative of that family. We then compared the test-sample recognition performance and picked a subset of best performing gestures as the Cyclops gesture vocabulary. Finally we implemented the algorithm on the Cyclops and performed a test of recognition performance of the gestures in real-time. Fig-11 shows Cyclops observed gesture and its associated orientation histogram. This data has been generated by constantly dumping the Cyclops observing image as well as its orientation histogram and its detected gesture. In practice, for image sizes of 64×64 pixels and for a vocabulary of 5 gestures, in constant illumination we had 92% of successful and 8% of failure recognition. In these cases the execution time of computation is 448ms. We leave further improvement and analysis of the performance of our algorithm as our continuing research.

6. RELATED WORK

Recently, low power CMOS imagers have been subject of tremendous attention [6] [7] [15] [16] [17]. CMOS imagers provide a low power alternative to charged-coupled device (CCD) imagers which have dominated the high quality imaging technology for the past decades. Large amount of research in improving the quality of CMOS imagers and reducing its size and power consumption has lead to significant growth of CMOS technology in commercial devices and in miniature imaging platforms. The commercialization process of these products however, has been dominated by the requirement of the market. This has lead to the class of devices that are designed to provide human pleasing images [18] [19] [20] [21] and integrate with powerful processors in cell phones and personal data assistant devices (PDAs) that have sufficient memory and clock speed. This is a serious obstacle in using them with less capable devices. In addition these devices require a complex programming interface. While Cyclops uses a CMOS imager, it hides the complexity of the device from the host processor by providing a simple interface. In addition it performs internal computation to present a form of inference from the image to the host.

In terms of system integration, low power CMOS video chips for portable multimedia has been investigated in literatures. In this research the goal is provision of video stream data at lowest possible power by using high degree of compression and maximum integration of the imagers, signal processing and radio transmission components. This often leads to extremely low power data transmission rate for video streaming in portable devices [22] [23]. The result of this high level of integration however is minimal access to the lower layers and lack of flexibility. This is in contrast with the goal of Cyclops to provide access to the user to design a sensor for the required imaging operation. In addition these highly integrated devices are link layer components that can not be used in a network with on demand image capture and interpretation requirements.

Finally distributed image processing systems have recently gained new level of attention due to the ubiquitous availability of imagers such as webcam. Due to the high power consumption of these imaging devices and their complex programming requirements, many of these systems exploit computationally capable hosts. These hosts are often some sort of light weight embedded computer with permanent power requirements and high cost of individual node. This fundamentally limits the scale of the experiments. Example of these cases are distributed attention, surveillance and monitoring systems which often combined with some form of actuation of the imagers [25] [26] [24]. In these researches, the goal of the system is detecting interesting objects or events with maximum efficiency (i.e. coverage, minimum false-positive). Other examples of these system are animal surveillance systems where the goal of the system is monitoring the health and condition of animals particularly in the context of biological experiments [27]. In all these cases the dimension of the deployment, the complexity of the distributed algorithm and the power availability assumption were based on the available classes of imagers and accompanied computers. Cyclops on the other hand enables much larger scale experiments but with much limited amount of available computation. This is entirely a new regime of operation that has not been explored before.

7. CONCLUSION AND FUTURE WORK

In this paper we presented a vision sensor for wireless sensor networks that performs local image capture and analysis. We introduced Cyclops and explained its hardware and software architecture and provided its power characteristics and temporal profile. We further demonstrated its applicability to object detection and gesture recognition. Through this paper we used the dedicated logic (CPLD) only for frame capturing. Such dedicated logic resources can be used for further reduction in processing time and power, particularly for operations that can be pipelined during image capture operation.

Cyclops makes it possible to create new vision applications. It can be used in conjunction with other sensors to disambiguate ambient conditions or detect motion in security or surveillance applications. It can detect changes in size or shape of the objects for applications such as system monitoring and diagnostics or for phenology and habitat monitoring.

While our initial experiments in the application domain are encouraging, they are far from comprehensive and it will be the subject of our ongoing research. In particular, future work will focus on collaborative in-network applications such as multi-Cyclops object detection in different coverage scenarios, hierarchical vision applications and more complex multi-modal sensing scenarios.

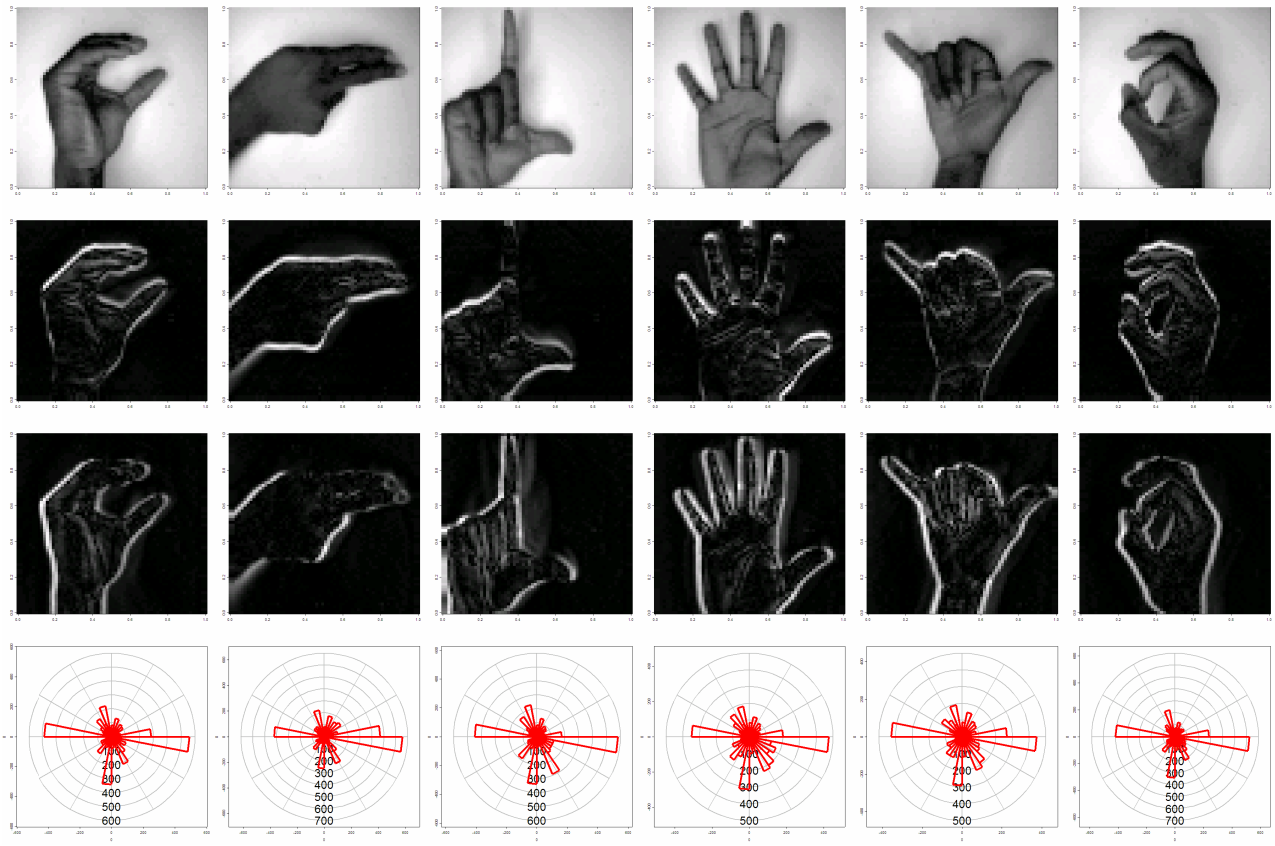


Figure 11: In this figure the rows from the top are: I) Cyclops instantaneous images, II) its constructed Sobel derivatives in and vertical III) horizontal directions and IV) the polar view of its orientation histogram. This data has been generated by constantly dumping the images, their derivatives and their orientation histograms from Cyclops to an attached logging computer.

8. ACKNOWLEDGEMENT

This work was made possible with support from The Center for Embedded Networked Sensing (CENS) under the National Science Foundation (NSF) Cooperative Agreement CCR-0120778 and support from Agilent Technology.

9. REFERENCES

[1] Estrin D., Govindan R., Heidemann J. and Kumar S, "Next century challenges: Scalable coordination in sensor networks", Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), 1999.

[2] Cerpa A., Elson J., Estrin D., Girod L., Hamilton M. and Zhao J., "Habitat monitoring: Application driver for wireless communications technology" ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, Costa Rica, 2001.

[3] Forsyth D. and Ponce J., "Computer Vision : A Mordern Approach", Prentice-Hall, 2001.

[4] Crossbow Technology, "http://www.xbow.com"

[5] Ember Corporation, "http://www.ember.com/index.html"

[6] El Gamal A., "Trends in CMOS Image Sensor Technology and Design", International Electron Devices Meeting Digest of Technical Papers, 2002.

[7] Hurwitz J., Smith S.G., Murray A.A., Denyer P.B., Thomson J., Anderson S., "miniature imaging module for mobile applications", ISSCC Digest of Technical Papers, 2001.

[8] Atmel Corporation, "http://www.atmel.com/products/avr"

[9] Gay D., Levis P., Behren R.V., Welsh M., Brewer E. and Culler D., "The nesC Language: A Holistic Approach to Networked Embedded Systems", Programming Language Design and Implementation (PLDI), 2003.

[10] Tinyos: An operating system for networked sensors, "http://tinyos.millennium.berkeley.edu".

[11] Polastre J., Szewczyk R., Culler D., "Telos: Enabling Ultra-Low Power Wireless Research", The Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors, 2005.

[12] Cheung S.C. and Kamath C., "Robust techniques for background subtraction in urban traffic video", Video

Communications and Image Processing, SPIE Electronic Imaging, San Jose, 2004.

[13] William T. Freeman, Michal Roth "Orientation Histograms for Hand Gesture Recognition" IEEE Intl. Workshop on Automatic Face and Gesture Recognition, Zurich, 1995.

[14] Freeman W.T., Anderson D.B., Beardsley P.A., Dodge C.N., Roth M., Weissman C.D., Yerazunis W.S., Kage H., Kyuma K.,Miyake Y. and Tanaka K., "Computer Vision for Interactive Computer Graphics", IEEE Computer Graphics and Applications, Vol. 18, 1998.

[15] Rhodes H., Agranov G., Hong C., Boettiger U., Mauritzson R., Ladd J., Karasev I., McKee J., Jenkins E., Quinlin W., Patrick I., Li J., Fan X., Panicacci R., Smith S., Mouli C., and Bruce J., "CMOS Imager Technology Shrinks and Image Performance", Micron Technology, Inc., 8000 S. Federal Way, Boise, Idaho.

[16] Fossum E.R., "Digital Camera System on a Chip", IEEE Computer Society Press ,Los Alamitos, CA, USA, V. 18 ,I. 3, 1998.

[17] Findlater K.M., "A CMOS camera employing a double junction active pixel", Thesis submitted for the degree of Doctor of Philosophy, University of Edinburgh, 2001.

[18] STMicroelectronics, "http://www.st.com/"

[19] Micron, "http://www.micron.com/"

[20] Agilent Technologies, "http://www.agilent.com/"

[21] OmniVision Technologies Inc., "http://www.ovt.com/index.html"

[22] Chandrakasan A., Burstein A. and Robert Brodersen W., "A low power chipset for portable multimedia applications", IEEE Int. Solid-State Circuits Conference, 1994.

[23] Simon T. , Chandrakasan A.P., "An Ultra Low Power Adaptive Wavelet Video Encoder with Integrated Memory", IEEE Journal of Solid-State Circuits, 2000.

[24] Collins, Lipton and Kanade, "A System for Video Surveillance and Monitoring", American Nuclear Society (ANS) Eighth International Topical Meeting on Robotics and Remote Systems, Pittsburgh, April 1999.

[25] Chu M., Reich J.E., Zhao F., "Distributed Attention for Large Video Sensor Networks", Intelligent Distributed Surveillance Systems seminar, London, UK. 2004.

[26] Kansal A., Yuen E., Kaiser W., Pottie G. and Srivastava M., "Sensing Uncertainty Reduction Using Low Complexity Actuation", ACM Third International Symposium on Information Processing in Sensor Networks, 2004.

[27] Belongie S., Branson K., Dollr P. and Rabaud V. "Monitoring Animal Behavior in the Smart Vivarium", Measuring Behavior, Wageningen, The Netherlands, 2005.