EXPLORING TRADEOFFS IN ACCURACY, ENERGY AND LATENCY OF SCALE INVARIANT FEATURE TRANSFORM IN WIRELESS CAMERA NETWORKS

Teresa Ko, Zainul M. Charbiwala, Shaun Ahmadian, Mohammad Rahimi,
Mani B. Srivastava, Stefano Soatto, Deborah Estrin

Center for Embedded Networked Sensing, University of California, Los Angeles
tko@cs.ucla.edu, zainul@ee.ucla.edu, ahmadian@lecs.cs.ucla.edu, mhr@cens.ucla.edu,
mbs@ee.ucla.edu, soatto@ucla.edu, destrin@cs.ucla.edu

ABSTRACT

Advances in DSP technology create important avenues of research for embedded vision. One such avenue is the investigation of tradeoffs amongst system parameters which affect the energy, accuracy, and latency of the overall system. This paper reports work on benchmarking the performance and cost of Scale Invariant Feature Transform (SIFT) for visual classification on a Blackfin DSP processor. Through measurements and modeling of the camera sensor node, we investigate system performance (classification accuracy, latency, energy consumption) in light of image resolution, arithmetic precision, location of processing (local vs. server-side), and processor speed. A case study on counting eggs during avian nesting season is used to experimentally determine the tradeoffs of different design parameters and discuss implications to other application domains.

Index Terms— embedded vision, system tradeoffs, DSP, SIFT, object recognition

1. INTRODUCTION

While it is intuitive that, in wireless camera sensor networks, local processing of images followed by transmission of processed data is generally more efficient than direct transmission of raw images, there has been little quantitative study of the specific cost and performance tradeoffs characterizing these two approaches. This is due in part to the complexity of vision algorithms and the volume of imaging data that until very recently has been incompatible with the processing, storage and energy constraints of camera sensor nodes. Hence, the emphasis of the research in computer vision has been historically given to enhancing the performance of the algorithms with little interest in improving them under constraints of embedded environments. However, with continuous advances of Digital Signal Processing and computing technology [1], there are new avenues of research to study the application of state of the art computer vision approaches in more computationally-capable camera sensor networks.

In this paper, we empirically study a camera sensor node which uses Scale Invariant Feature Transform (SIFT) [2] to identify objects in the environment. The sensor extracts SIFT features from the images and matches them to the features of the object(s) of interest through application of a local Support Vector Machine (SVM). By using SIFT features which are invariant to scale, rotation and substantial range of affine distortion and varying illumination, the sensor node can reliably identify objects in a cluttered or occluded environment. While local SVM requires a training phase, we assume that the SVM on the camera sensor is trained initially and rather focus on the object classification problem.

To study the performance and cost of the SIFT based sensor against various system parameters, we model a camera sensor which consists of a Blackfin DSP processor [3], a low-power CMOS image sensor, an 802.15.4 CC2420 radio and acquisition and processing memory. We have chosen the Blackfin processor due to a portfolio of features including high computation performance to power consumption ratio, hybrid architecture that supports efficient computation as well as control oriented applications (processing images vs. communicating with radio or imager), multiple power states and agile transition among them, and finally, rapid frequency and voltage rescaling to adjust performance during various operation episodes [4, 5]. The CC2420 radio which is widely used in various sensor network nodes [6] is chosen due to its low power consumption and moderate transmission speed.

Our study is generally divided into two sets of experimentation to benchmark 1) local computation cost and performance and 2) overall camera sensor performance. To benchmark the computation cost, we have implemented the SIFT feature extraction and SVM classification on the Blackfin processor and benchmarked the computation in the VisualDSP++ instruction level simulator. To expedite accurate benchmarking of classification accuracy on test datasets, we ran the same code on a simulation server where many instances of the classification performance are evaluated. Finally, we feed our results to an analytical camera sensor model to evaluate overall node performance under various system parameters.

Through experimentation, we study the sensing accuracy, latency and energy consumption in light of various system pa-

1-4244-1354-0/07/$25.00

rameters and compare our results with direct transmission of the raw images. The parameters include a range of architecture-specific and application-specific parameters including input image resolution, processor frequency, arithmetic precision of computing and various combination of SIFT scale spaces. Our results illustrate that by adjusting the system parameters we can significantly reduce energy consumption of the local SIFT computation with minimum loss in classification accuracy. In addition, although local SIFT classification (*i.e.*, compared to sending the raw images) leads to significant reduction of bandwidth utility in a camera network and has an important benefit in and of itself, we find that in some instances it also leads to reduction in node energy consumption or sensing latency. For instance, the energy consumption of a camera node that performs optimized SIFT classification is less than its energy consumption when it sends raw images of the size that results in the same classification accuracy at a backend server (which runs a standard implementation of SIFT).

Clearly, the optimization of the application-specific parameters is only meaningful in the context of an application. Throughout this paper, our experiments are focused on a case study of counting eggs during avian nesting season. While the numbers are representative of this application, we discuss generalization of the basic tradeoffs to a broad range of application domains.

The primary contribution of this paper is an empirical evaluation of SIFT classification on a model of a Blackfin-based camera sensor. We identify tradeoffs among the key system parameters that affect the sensing accuracy, latency and energy consumption and explore the design space where those parameters can be adjusted to fulfill the requirements of the application. In addition, we investigate the tradeoffs of local processing of the images followed by transmission of the result data vs. transmission of the raw images. A secondary contribution of our work is the implementation of the application and architecture optimized SIFT algorithm on the embedded Blackfin processor, previously considered the exclusive realm of high-end computers.

The rest of this paper is organized as the following. Section 2 describes the camera sensor model. Section 3 outlines various parameters that affect the performance and cost of sensing. Section 4 describes the implementation of the SIFT feature extraction and SVM classification on Blackfin processor. Finally, Section 5 describes the experiments followed by the presentation of the results and discussion in Section 6.

2. SYSTEM MODEL

To evaluate the energy consumption and detection latency, we have developed a model of the camera sensor node platform. The model consists of four major components- Imager, CPU, Memory and Radio. Figure 1 shows the interconnection of the blocks in our system. The Imager component is the CMOS or CCD imaging element responsible for acquiring the image. The imager used for evaluation in this study is the Agilent

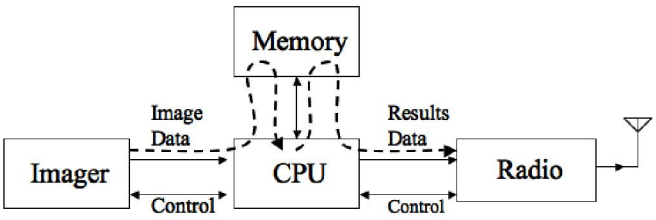

Fig. 1. System Model Block Diagram

ADCM-1700 CMOS Camera Module [7]. This module has an I2C control bus and an 8-bit parallel CCIR656-compliant data bus, both of which are connected gluelessly to the CPU, the Blackfin ADSP-BF533.

The Blackfin DSP is at the heart of the camera sensor node platform and performs algorithmic as well as control functions. The processor has a direct peripheral interface to synchronous DRAM memory, in our case, a low-power Micron part [8]. This memory is used as temporary storage for image data captured from the imager, for intermediate data generated by the SIFT algorithm and for the final results. The hashed lines in Figure 1 denote the flow of image and results data from the imager to memory to the CPU and from the CPU to the radio component respectively.

The radio component used is the popular Texas Instruments CC2420 Single-Chip 2.4 GHz RF Transceiver which has been extensively studied in [6]. The radio component receives both the control commands as well as the results from the CPU along the SPI bus. For purposes of our evaluation, we assume a lightly loaded one-hop network to form a baseline for comparing costs of local processing versus image transmission. Channel contention, lossy transmissions and hop delays in loaded and multi-hop networks favor the local processing argument.

In our model, we assume each component in the system spends time and energy in one of a set of distinct states. This approximation is similar to that in [9] and is commonly used to model hardware at the micro-architecture level [10] and full system level [11, 12]. However, our approach is slightly different from [9] by modeling the state of each component rather than the entire system. The full system model is then constructed as a combination of the states of each of the components over the course of time. This feature lends greater flexibility in modeling the operation of the system and conveniences the user evaluating the effect of changes to individual components.

As a measure of standardizing the taxonomy of states across widely heterogeneous components, we distill the various states into a set of seven basic states. We then map the basic states into component-specific states. Table 1 shows these states and power consumption figures derived from the respective datasheets:

- Sleep (S): In the sleep state, each component of the system is in a power down mode. Each component in the system has the ability to power itself down by software

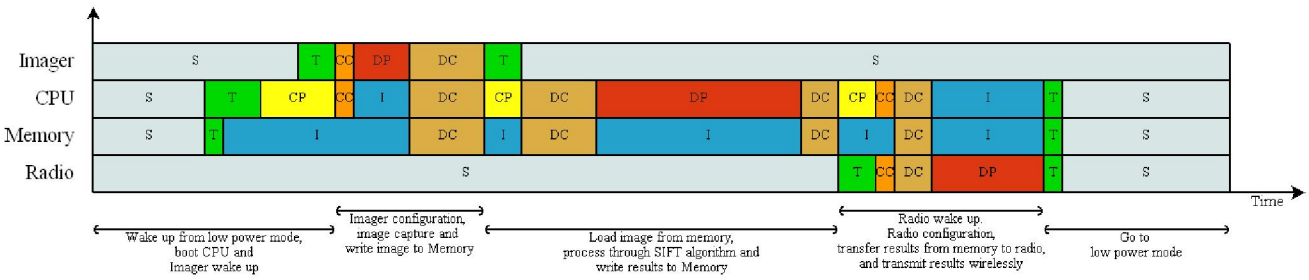

Fig. 2. System Operation and Data Flow. Each block represents time spent in a specific state for the respective component.

Power (mW) / State	Imager	CPU 50MHz	CPU 600MHz	Memory	Radio
Sleep	1	0.081	0.081	0.018	0.054
Transition	50	13.2	141.3	171	48.8
Data Proc	42	20.8	264	0	47
Data Comm	42	39.8	283	171	48.8
Control Proc	10	20.8	264	0	50.76
Control Comm	42	22.9	266.1	0	48.85
Idle	10	11.2	37.2	0.360	50.76

Table 1. State-wise Power Consumption for each Component

control, a feature that obviates the need for external power switching.

- Transition (T): The transition state represents the time interval taken to go between states. In our study, we have found that the only non-negligible time is the wake up delay of a component.
- Data Processing (DP): The data processing state models the component performing some algorithmic or functional task. For the imager, the DP state represents the image capture time, whereas for the radio, it represents the time spent transmitting data wirelessly. The DP time spent by the CPU is the time spent in the SIFT algorithm.
- Data Communication (DC): The data communication state represents the interval when components are exchanging data with each other. The time each component spends in this state depends on the bus interface speed between components.
- Control Processing (CP) and Control Communication (CC): Control processing is the time that each module spends during the configuration whereas control communication is the time it spends exchanging configuration information and commands.
- Idle (I): Idle time is spent in components waiting for each other to complete a task or subtask. In the case of imager and radio, since the CPU co-ordinates their operation, the idle time can be ideally made zero (by waking up at the right instant and staying up for only the time required to perform their task). CPU itself idles while the imager is capturing an image and the radio is transmitting results data.

Parameter	Symbol	Value	Unit
Image Capture Speed	$\rho_{I,cap}$	1.53	Mpix/s
Imager Comm. Speed	$\rho_{I,comm}$	13	Mpix/s
I2C Bus Speed	$\rho_{I,i2c}$	50	KB/s
Memory Access Speed	$\rho_{M,acc}$	133	MHz
Radio Transmit Speed	$\rho_{R,tx}$	31.25	KB/s
SPI Bus Speed	$\rho_{R,spi}$	100	KB/s
Sampling Interval	ρ_{samp}	-	s
CPU Clock Frequency	$\rho_{C,freq}$	-	MHz
Image Resolution	$\rho_{I,res}$	-	pixels
CPU Cycle Count	$\rho_{C,cyc}$	-	cycles
Memory Access Count	$\rho_{M,cnt}$	-	accesses
Result Size	$\rho_{R,size}$	-	bytes

Table 2. System Parameters and System Variables

In our system we assume a schedule-driven approach, where the sensor wakes up periodically to perform the image acquisition and classification. Figure 2 illustrates the sequence of operations in one round of the node's scheduled activity.

The camera sensor node is initially in the sleep state with each component powered down. Even in its low power mode, the CPU sets a real time clock timer to wake up the CPU on a specific schedule. When the real time clock expires, the CPU (and hence memory) wakes up, boots and carries out perfunctory control processing (CP) tasks. The CPU then communicates with the imager to configure image acquisition (CC block) but both CPU and memory move to the idle state while the image acquisition in progress. At the end of the acquisition period, the imager stores the image data in the memory (DC block) via the CPU's direct memory access (DMA) and returns back to sleep upon receiving appropriate configuration commands from the CPU.

The CPU then runs the SIFT feature extraction and SVM classification and writes the results back to the memory. These tasks are memory-intensive and the memory load/store operations are intermixed with arithmetic operations running on the CPU's execution units. However, to simplify the illustration we depict memory accesses (DC blocks) distinct from the main processing (DP block). The CPU then wakes up the radio and configures it to transfer the results to the radio

(DC block) to be subsequently sent over the radio (DP Block). Since the result data may not all fit into the limited buffer of the radio, this process is performed through multiple DMA operations with the CPU idling during radio transmission intervals. All components are powered down at the end of the transmission period.

To compute the overall energy consumption during each operating episode, we determine the time spent in different states by each component and combine them with the amount of power they spend in that state (from Table 1). Combining these times with the power figures in provides the final energy consumption as:

$$E_{total} = \sum_{\lambda \in \Lambda} \sum_{\omega \in \Omega} (\tau_{\lambda,\omega} * P_{\lambda,\omega})$$

Here, Λ represents the set of components *{Imager, CPU, Memory, Radio}* and Ω is the set of states *{S, T, DP, DC, CP, CC, I}*. $\tau_{\lambda,\omega}, P_{\lambda,\omega}$ are time and power consumed by component λ in state ω. To compute the detection latency of the system, we combine the data processing, data communication, control processing and control communication time for each component and term it as the active time of the system. The combination is performed using:

$$\tau_{active} = \left[\sum_{\lambda \in \{I,C,M,R\}} (\tau_{\lambda,DP} + \tau_{\lambda,CP}) \right] + (\tau_{C,DC} + \tau_{C,CC})$$

The first term corresponds to processing times for each component. The second term covers communication times between components. Since all communication is via the CPU, it is sufficient to consider just the DC and CC times for the CPU.

To determine the time spent in each state of a round, we derive first-order analytical relationships for each term with respect to hardware specific system parameters and algorithm dependent system variables. The top half of Table 2 lists the parameters derived from respective data sheets and the bottom half represents system variables that are varied (as discussed in Section 3) for system evaluation.

Table 3 provides the simplified expressions for significant time intervals used to compute energy consumption and latency. Note that the key arguments in this computation are the CPU cycle count and memory access count, which depend on the algorithmic load as well as the data. To accurately model the system, these values are evaluated from real execution traces using the cycle accurate simulator provided within VisualDSP++. Using this approach, we compare the costs of local classification to image transmission with server-side processing across a number of design variables. Counts derived from execution traces are used to determine times for local computation whereas server-side computation implies $\rho_{C,cyc} = 0$ and $\rho_{R,size} = \rho_{I,res}$. Simplified analytical models have the drawback of not capturing subtle higher-order effects on evaluation metrics. However, based on experimentally derived results using the ADSP-BF533 EZ-KIT Lite evaluation platform, we believe this model is adequately representative of the node and reckon it to be a tool useful for a general audience.

Time	Relationship
$\tau_{\lambda,S}$	$\rho_{samp} - \sum_{\omega \in \Omega} \tau_{\lambda,\omega}$
$\tau_{I,DP}$	$\rho_{I,res}/\rho_{I,cap}$
$\tau_{C,DP}$	$\rho_{C,cyc}/\rho_{C,freq}$
$\tau_{R,DP}$	$\rho_{R,size}/\rho_{R,tx}$
$\tau_{I,DC}$	$\rho_{I,res}/\rho_{I,comm}$
$\tau_{M,DC}$	$\rho_{M,cnt}/\rho_{M,acc}$
$\tau_{R,DC}$	$\rho_{R,size}/\rho_{R,spi}$
$\tau_{C,DC}$	$\tau_{I,DC} + \tau_{M,DC} + \tau_{R,DC}$
$\tau_{C,I}$	$\tau_{active} - \tau_{C,DP} - \tau_{C,DC} - \tau_{C,CP} - \tau_{C,CC}$
$\tau_{M,I}$	$\tau_{active} - \tau_{M,DC}$

Table 3. State-wise Time Relationships

3. SYSTEM VARIABLES

There are several places in the system design in which trade-off can be made in terms of energy, latency, and accuracy. The following is a listing of the ones evaluated in this paper.

3.1. Architecture designs

Arithmetic precision: Floating point, 16-bit fixed point. Floating point is the default arithmetic precision used in the SIFT algorithm, but incurs significant cycle counts due to the DSP lacking a floating point unit. Using 16-bit fixed point allows us to take advantage of architectural optimizations designed for the native arithmetic precision of the Blackfin, such the MAC unit.

CPU freq: 50 Mhz, 63 Mhz, 124 Mhz, 400 Mhz, 500 Mhz, and 600 Mhz. By reducing the frequency of the Blackfin processor, power consumption is reduced, but may ultimately result in increases in the overall latency and energy consumption due to idling components in the system.

3.2. Application designs

Number of Octaves: 1, 2, 3, 4, 5, ..., N. Invariance to scale is achieved by consistently finding feature descriptors at the same scale regardless of the scale the object is captured. Detection of these features are guaranteed by repeatedly searching at different scales. Depending on the application, it may be possible to reduce the number of octaves (set of scales) searched. Note that the search space is implicitly reduced by the selection of the capture resolution.

Scale space sampling: Direct or Inferred. To broaden the scale space search, the generic SIFT algorithm upsamples the original image and detects features at that octave. We distinguish between the SIFT descriptors extracted through upsampling because the resulting histogram is different than if the scene was captured at the higher resolution and the computational cost is substantially different.

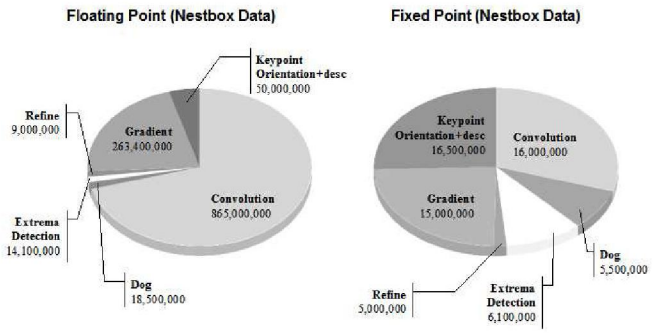

Fig. 3. SIFT breakdown on a 160x160 image for a single octave at two types of precision, fixed-point and floating-point. Here fixed point includes using native bit-width data types, compiler optimizations, and fixed-point implementation of particular functions.

4. BLACKFIN PORT

The compiler influences the amount of effort required to port code to a particular platform; if the compiler is unable to make appropriate platform-specific optimizations, the programmer must re-implement those sections of the code so that they do. But before re-writing parts of the code, the application should be profiled to identify key bottlenecks in the implementation.

In our study, a C++ implementation [13] of SIFT was used. Having tuned the parameters of SIFT for our particular application, the code was compiled and executed on the EZ-Kit Development Board using Visual DSP++. As can be seen on the left side of Figure 3, this compiler-optimized floating-point version spends most of its time in the convolution and gradient phases.

A first cut optimization converted the variables to the DSP's native word length (16-bits). Since the input to these functions is a pixel value in the range of (0,1), they are first mapped to 16-bit integer values (16.0 format) thereby allowing us to use regular integer operations. In intermediate steps where larger dynamic range was needed, 32-bit integers were used for temporary storage. Both convolution and gradient functions contain simple operations making it easy to verify the dynamic range at each step. Additionally, code within `for` loops was re-organized to make them more compiler-friendly as discussed in [14]. Together these allow the compiler to make more efficient use of dedicated hardware resources such MAC units.

A second optimization replaced math related function calls with fixed-point alternatives. Two functions, the `arctan()` and `sqrt()` were replaced with a well-known function approximation and LUT techniques, respectively, with errors in the order of 0.06 radians and 5% respectively.

The effects of our changes on algorithmic accuracy are empirically evaluated during experimentation on real data sets (see Section 5). An important point to note is that for robustness, SIFT quantizes keypoint orientations and descriptors by binning them at certain granularities; consequently certain errors introduced by the conversion to fixed-point have no effect on accuracy. A more in-depth analysis similar to [15] could be done to determine how much precision could be forgone while the quantization inherent in SIFT remains the most dominant source of error. Overall the effect of our optimizations can be seen on the right side of Figure 3. Based on cycle counts from our simulation and EZ-Kit, we were able to improve from an average of 50K cycles/pixel to approximately 2K cycles/pixel in the fixed-point version depicted in the figure.

In conclusion, we see that given reasonable compiler support such as the one found in VisualDSP++, C++ code can be relatively easy to optimize without having to deal with assembly. Also, since the number of features is application dependent, it might be the case that optimizations should be more focused at the descriptor stage. For this reason it is important to profile the application in order to guide further optimization that could further increase the benefits of local computation.

5. RESULTS

In this section, we describe the experiments used to evaluate the tradeoffs between accuracy, energy, and latency. These findings will be used to propose pairings between system design and application domain in the following section. The energy consumption of running SIFT on the node is modeled as:

$$E_{total} = E_{fixed} + E_{variable}$$

E_{fixed} represents a fixed cost given input resolution of image and $E_{variable}$ can vary depending on application specific parameters such as thresholds that increase computation on the device by varying amounts. In the case of SIFT, the fixed cost includes the creation of the scale space, DoG, extrema detection, and pixel gradients; the variable cost includes refinement of the extrema points along with the determination of keypoint orientation and descriptors. In choosing between local computation and transmission of raw image, at minimum, E_{fixed} of local computation needs to be below the cost of transmitting the image.

Energy/Latency under Arithmetic Optimizations Figure 4 shows that the traditional SIFT algorithm which scans the entire scale space (all octaves) consumes more energy than transmitting the image for both the floating point and fixed point implementations, and that this difference is exacerbated as images get larger. Additional energy consumption will be consumed by searching for and computing the SIFT features. *If the generic SIFT algorithm is needed, transmitting the image and processing at the sever-side is better in terms of accuracy, latency, and power.*

Accuracy under Fixed-Point As described in Section 4, fixed-point arithmetic would allow the Blackfin DSP to utilize processor specific optimizations, reducing computation by an order of magnitude. However, this energy/latency reduction comes at the cost of accuracy. In Table 4, the effect

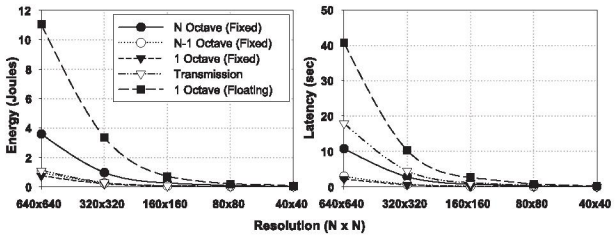

Fig. 4. Energy/Latency vs image resolution for different local computation schemes, for the fixed cost portion of SIFT. Regions where tradeoffs may exist can be identified by looking under the Transmission curve.

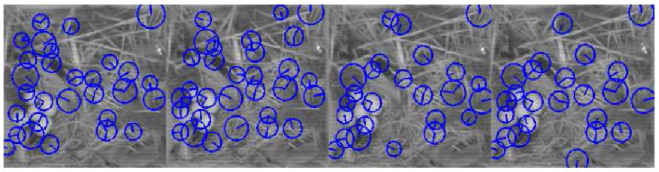

Fig. 5. Sample SIFT frames from (floating point computation, fixed inferred, floating, and inferred fixed (from left to right). While most SIFT features closely match, there are unmatched features as well.

on SIFT features is shown when using fixed-point arithmetic precision.

5376 images were used for experimentation. SIFT features are defined as matching if the position in scale space are within the scale at which it is found. The table also describes the number of extra and missed features relative to the floating point implementation. L2-norm distance in position, scale, orientation and descriptors are shown to be relatively minor. The same image of a nest with several eggs are shown in Figure 5 under and the same descriptor is shown in Figure 6, under different computational situations. From the images, it is possible to see that many of the SIFT features remain the same. Also, the resulting descriptors are still quite similar to each other as compared to another random SIFT descriptor. More over, the error in moving to fixed point is approximately the same as using the inferred method (upsampling and capturing descriptors at that octave).

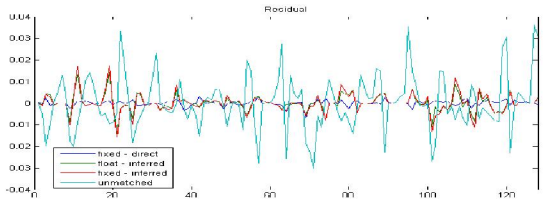

Fig. 6. Residuals of SIFT features, illustrating the magnitude of the error for the fixed point and/or inferred as compared to features that do not match.

	Direct	Inferred	
	Fixed	Float	Fixed
% Matches	76.29%	64.72%	58.85%
% Misses	23.71%	35.28%	41.15%
% Extras	17.52%	41.51%	37.51%
Ave. position error	4.3306	12.4732	12.7448
Ave. orientation error	2.8680°	3.4028°	4.0723°
Ave. descriptor error	0.0407	0.0501	0.0528

Table 4. Statistics comparing the difference between features extract from under different implementations. Direct sampling in floating point is considered to be ground truth. As more optimizations are used, error increases.

Energy/Latency under Varying No. of Octaves and Features Figure 4 also illustrates that by reducing the scale space search, it is possible for fixed point local computation to consume less energy than transmission. Computing the base octave (single) alone to computing all octaves including the base octave and below (N-1) requires less energy than transmission.

More details can be seen in Figure 7 which normalizes the energy consumption and latency relative to the most optimized version of transmission of that image for clarity. Up to now we have been considering only the fixed cost of each scheme; to evaluate its energy efficiency we must also determine $E_{variable}$. As stated previously this factor includes the refinement as well as keypoint orientation and descriptor determination. Based on our test results (as seen in Figure 3) for fixed point implementation, the most computationally expensive of the two is the keypoint orientation and descriptor, which require typically 150K cycles compared to 15K cycles required for refinement. Hence, we vary the number of feature points to see how that affects the overall energy consumption.

Feature values are given in terms of features/pixels (# of features found per pixel), which were found to be relatively constant in our experiments, and equal to 0.003. Energy consumption is given for 0.010 as well, due to results found in [2] which reported a higher number of features. The general trend is that as images get larger, the more cost effective it is to process images locally. Optimizations can be done on the number of features evaluated to further reduce computation cost. We calculated the energy and latency for both the single octave case as well as the (N-1) octaves (Figure 7). From these graphs we see that if the search space is limited to a single octave, the application can extract a large number of feature points and still be more efficient than transmitting the image. On the other hand, if the application needs to search a larger window of the scale space, it must filter out more features so that it does not surpass transmission energy.

Energy/Latency under varying CPU frequencies The previous results used the optimal CPU frequency for each task. A closer look into the effects of CPU frequency on consumption and latency is discussed here. In some cases, local process-

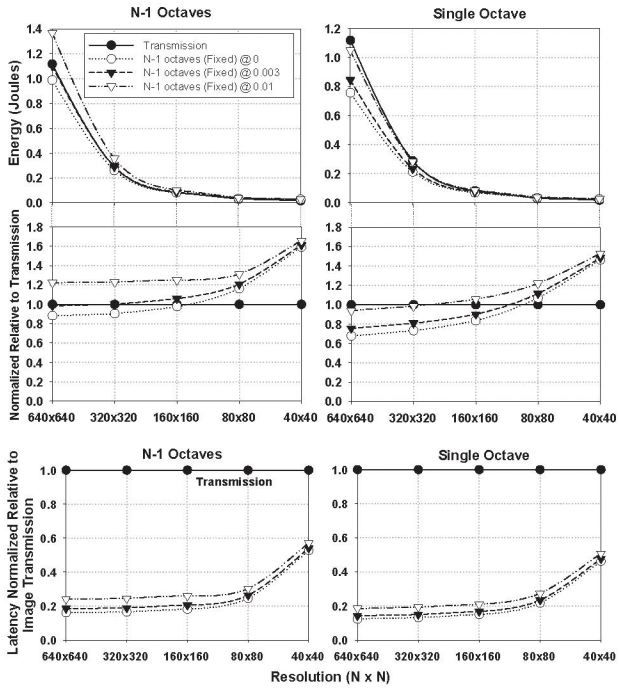

Fig. 7. Comparing energy/latency with varying number of features found for N-1 octaves and single octave.

ing consumes less energy than transmission, shown in Figure 8. The best energy consumption and latency for transmitting an image is at the lowest CPU frequency. This is due to the fact that the radio dominates in this situation. On the other hand, local processing achieves the best results at the highest frequencies. Under any resolution, there exists a CPU frequency in which it is more efficient in terms of latency to process locally. The energy graphs tell a different story; variation in CPU frequency could potentially save energy compared to transmission at higher resolutions since the dynamic power consumption of the node dominates the static power, which is not the case at lower resolutions.

Accuracy under Inferred Sampling Reducing the image capture resolution and using inferred sampling has greater performance impact to fixed point arithmetic precision. Fewer matches occur (64.72%), with greater number of extras (41.51 %) as well as larger errors in the position, orientation, and descriptor of the SIFT feature. Using fixed point and inferred sampling accumulates error from both operations, resulting in the worst performance over all. The only place where fixed point inferred sampling beats floating point inferred sampling is in the percentage of extra features.

Energy/Latency using inferred sampling To maintain the same scale space range, a lower resolution image can be sent through the network for server-side processing. The cost of this computation is compared in Figure 9. In this case, transmission wins out over local processing in terms of energy consumption. There is still a benefit in terms on latency though.

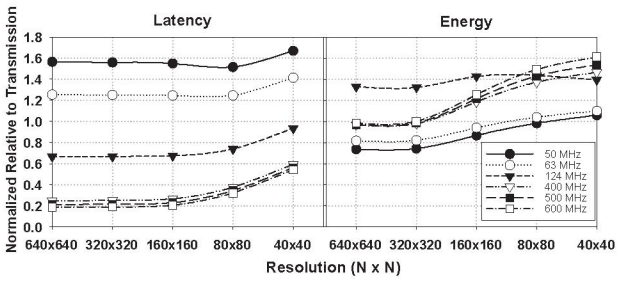

Fig. 8. Energy/Latency for nestbox application normalized to image transmission at base octave. Local processing computes N-1 octaves.

Case Study: Eggs The effects of inaccuracies incurred by the optimizations discussed above are best seen by comparing performance on a realistic recognition task. In this case, we look at detecting and counting eggs during a nesting cycle. To do this, we train a Support Vector Machine (SVM) to classify a SIFT feature as an egg or not. SIFT features from one nest box are used to train the SVM and another is used to test it. Table 5 shows the respective recall and precision of the SIFT features. As expected, both the precision and recall decrease by moving to a fixed point implementation.

However, the ultimate task is to determine the number of eggs in an image. Task accuracy in this case is measured by distance of the estimated count from the true count. The images are subdivided by the number of eggs in the image. *The average error is shown in Table* 5 *showing that on average error increases imperceptibly when fixed-point arithmetic is used.* It shows that while the inferred sampling has higher recall than direct sampling, it also has lower precision. While more SIFT features are classified as eggs, more non-eggs are classified as eggs than actual eggs classified as eggs. Using the inferred sampling method, greater loss in accuracy occurs.

6. DISCUSSION

In Section 5, we illustrated that there are many places where it may be possible to trade off one parameter to achieve better overall performance. In this section, we explore the relationship between application types and the design tradeoffs investigated in this paper.

In many environmental monitoring applications, low latency is not needed. A desired property may be a large scale deployment. Accuracy demands could be different for different tasks though. For example, being able to accurately count eggs in each image is much less important than being able to arrive at the correct number of eggs laid in the entire season. Noisy results can be aggregated to derive the correct answer at the end. On the other hand, precision in bird presence and absence during a nesting cycle is a statistic that is of great interest to avian scientist, and requires accuracy at each image level. These two different objectives suggest different system design decisions. Bird detection may require transmitting an

	Float Direct	Fixed Direct	Float Inferred	Fixed Inferred
Precision	31.43%	29.23%	24.81%	24.77%
Recall	46.15%	43.51%	51.59%	53.54%
Distance	1.25	1.26	1.31	1.36

Table 5. Recall/Precision on egg detection using different implements. Excepted difference from true egg count over a nesting cycle is also given.

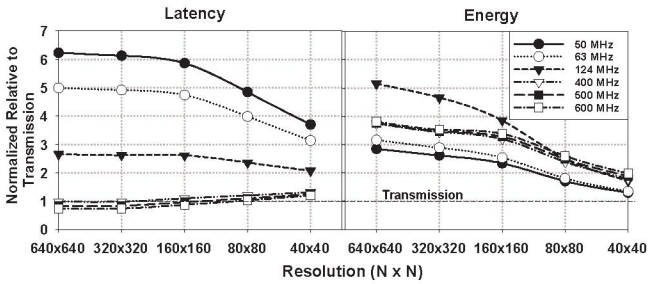

Fig. 9. Energy/Latency for nestbox application normalized to image transmission at 'inferred' octave (one octave below input image resolution)

image, even though it is a more costly operation in terms of energy and latency, to achieve the desired accuracy. Because egg counting does not require precision, a fixed point implementation on the sensor node might be preferable to preserve the lifetime and allow for a more scalable system.

Surveillance applications such as security monitoring or smart environments would have strict requirements on latency. An indoor deployment could afford to spend more energy if wired power was accessible, which argues for server side processing. Smart environments could prefer low latency over high accuracy, because real-time interaction is critical and user in the loop processing is possible. This would suggest that transmission of images with inferred sampling would be the most preferable system design. Some remote surveillance applications require accuracy over all else due to intolerable false alarms. In this case, transmission of the full image would be a better choice.

7. CONCLUSION

The empirical study presented in this paper illustrates the complexity of the design space for embedding vision algorithms onto a camera sensor node, even through the analysis of relatively few parameters. We show that by adjusting these system parameters, it is possible to run SIFT more efficiently on the local processor than on a centralized server due to transmission cost. The hope is that these results would be enlightening for future embedded vision work and better motivate the design choices made.

8. ACKNOWLEDGMENTS

This material is based upon work supported by the Center for Embedded Networked Sensing (CENS) under the National Science Foundation (NSF) Cooperative Agreement CCR-012-0778 and #CNS-0614853 and by the ONR under award #N00-014-06-1-0253. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or the ONR.

References

[1] J. Eyre and J. Bier, "The evolution of DSP processors," *Signal Processing Magazine, IEEE*, vol. 17, no. 2, pp. 43–51, 2000.

[2] D.G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[3] Analog Devices, "ADSP-BF533 Blackfin Processor Hardware Reference," 2003.

[4] BDTI, "BDTI's Buyer's Guide to DSP Processors," 2004 Edition.

[5] J. Donovan, "High Performance DSPs for Portable Applications," *Portable Design Magazine*, May 2006.

[6] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, Piscataway, NJ, USA, 2005, p. 48, IEEE Press.

[7] Agilent Technologies, "Agilent ADCM-1700 CIF CMOS Camera Module," *Technical Specification*.

[8] Micron Technology, "MT48H32M16LF 512Mb Mobile SDRAM," 2005.

[9] D. Jung, T. Teixeira, A. Barton-Sweeney, and A. Savvides, "Model-Based Design Exploration of Wireless Sensor Node Lifetimes," in *EWSN*, 2007, pp. 277–292.

[10] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *Proceedings of the 27th annual international symposium on Computer architecture*, pp. 83–94, 2000.

[11] S. Hengstler and H. Aghajan, "A Smart Camera Mote Architecture for Distributed Intelligent Surveillance," Boulder, CO, 2006.

[12] V. Shnayder, M. Hempstead, B. Chen, G.W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 188–200, 2004.

[13] A. Vedaldi, "SIFT++: A lightweight C++ implementation of SIFT," http://vision.ucla.edu/ vedaldi/code/siftpp/siftpp.html.

[14] Analog Devices, "Tuning C Source Code for the Blackfin Processor Compiler," 2003.

[15] D. Lee, H. Kim, S. Tu, M. Rahimi, D. Estrin, and J. D. Villasenor, "Energy-optimized image communication on resource-constrained sensor platforms," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, New York, NY, USA, 2007, pp. 216–225, ACM Press.