RECONFIGURATION IN NETWORK OF EMBEDDED SYSTEMS: CHALLENGES AND ADAPTIVE TRACKING CASE STUDY

Soheil Ghiasi, Ani Nahapetian, Hyun J. Moon, and Majid Sarrafzadeh

Abstract—**Many applications utilize deeply embedded sensors and actuators that are tightly coupled with the physical environment in order to perform their functionality. Sensor, actuators and embedded computation resources used for implementing such systems usually exhibit regular local configurations, while the global structure of the subsystems is either not fixed a priori and can change at runtime or is not known. Examples include systems that use many randomly distributed sensing boards, each one having a fixed structure of computation resources and sensing devices, to autonomously detect events and take proper actions.**

This paper discusses the requirements of the aforementioned systems, their advantages and the issues involved in developing them. Specifically we focus on dynamic adaptation of the system as a particular feature of such systems. This feature is discussed in depth in a collaborative and dynamically adaptive object tracking system that has been built in our lab as the experimental framework of this study. We exploit reconfigurable hardware devices embedded in a number of networked cameras in order to achieve our goal. We justify the need for dynamic adaptation of the system through scenarios and applications. Experimental results on a set of scenes advocate the fact that our system works effectively for different scenario of events through reconfiguration. Comparing results with non-adaptive implementations verify the fact that our approach improves system's robustness to scene variations and outperforms the traditional implementations.

Index Terms— **Adaptive Tracking, Feature Selection, Networked Embedded Systems, Reconfigurable Computing, Tiered Resource Architecture.**

I. INTRODUCTION

Many applications rely on distributed sensing of events, an example of which is a class of applications called unsupervised detection of spatio-temporal events. Instances of this class of applications include environmental monitoring, and traffic management and control. Traditionally, sensor nodes used in such applications, solely serve as data acquisition units that transfer the perceived information to processing stations. Utilizing sensor nodes with embedded computation resources allows the system to –at least partially- collocate the data acquisition and processing, which in turn improves system energy dissipation, scalability and robustness. In addition, it can enhance system performance for various implementations that exhibit non-negligible communication overhead [4].

One approach to developing networked embedded system uses similar sensor nodes for data acquisition and local processing. While having similar sensor nodes facilitates many development issues, network performance can be significantly improved by utilizing heterogeneous sensor nodes. Sensor nodes can vary in many different aspects including embedded processing power, communication overhead, power dissipation and the modality of the signal they sense. Moreover, heterogeneous sensor nodes can be deployed to form a tiered architecture, which allows intelligent utilization of proper resources for performing each of system tasks. This in turn results in significant system energy dissipation and performance improvements [6].

Figure 1 illustrates a tiered network of embedded sensors that has been built as part of this work. The system's application is to intelligently track some distinguished objects using its tiered architecture of resources. It employs many cheap and constrained acoustic sensors (micro-nodes) called motes1 [3] in its second tier of resources. Motes are deployed in an ad-hoc manner and have a short-range wireless radio that can be used for sending/receiving data to/from their close neighbors. Motes run on the battery and hence, it is essential for them to dissipate little amount of energy to save their batteries and increase their lifetime. Therefore, motes radio is restricted to communicate to close neighbors at specific points of time.

There are a few vision sensors (cameras) with more powerful computation resources, including reconfigurable hardware devices [36], mounted on panel corners. Cameras serve as first-tier network resources (macro-nodes) and their reconfigurable computation resources can be dynamically altered to better accommodate the particular task assigned to them. The location of the vision sensors on the panels is fixed. The communication media on the panel is a local area network and the resources do not run on the battery. The system exhibits a locally regular structure, since the configuration of resources on each panel (and higher tiers of the system) is

Manuscript received July 21, 2003. This work was supported in part by the U.S. Defense Advanced Research Project Agency (DARPA) under contract F33615-01-C-1906-P00001.

Soheil Ghiasi, Ani Nahapetian, Hyun J. Moon and Majid Sarrafzadeh are with the Computer Science Department, University of California (UCLA), Los Angeles, CA 90095 USA (phone: 310-794-5616; fax: 310-794-5056, e-mail: {soheil,ani,hjmoon,majid}@cs.ucla.edu).

[1] Motes also contain sensors of other modalities such as magnetometer and accelerometer, however acoustic sensors are used for this project.

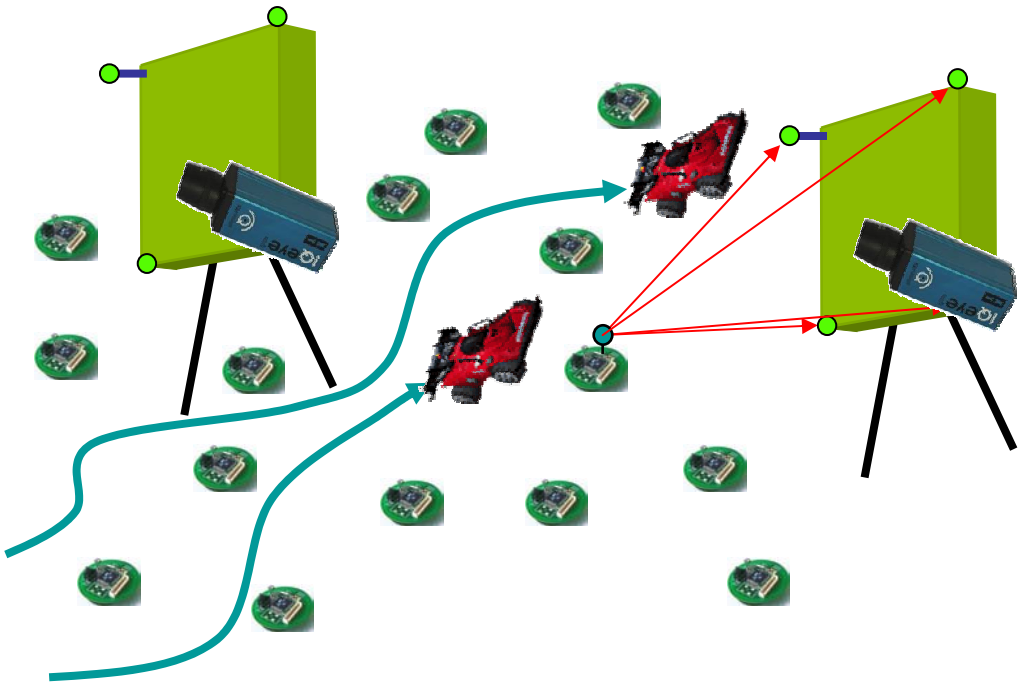

Figure 1. A tiered architecture combining engineered subsystems and ad-hoc global deployment.

determined. However, from a global point of view the structure of the entire system does not follow a predetermined pattern, due to the ad-hoc deployment of sensors in the field. The ideas and approaches presented in this paper are explained using this platform, however they are more general and applicable to other similar networked embedded systems.

Next section describes the research challenges arising in such a platform, which serves as an overview of various research topics addressed in the course of this project. A survey of other research efforts related to these challenges is presented in Section 3. We proceed to focus on a particular challenge in our system, namely dynamically adapting the system to environment changes thru hardware reconfiguration and parameterization. This issue is discussed in depth in the rest of this paper. A simplified version of the general system framework and its application is presented in Section 4. This version of the system performs collaborative target tracking thru hardware reconfiguration. In Section 5, we present the image-processing algorithms that are required for the implemented tracking application. In addition, the effect of environment changes on these algorithms and hence, the need for system adaptability is explained in this section. Section 6 discusses the issues involved in implementing these algorithms in our platform. Experimental results including algorithms implementation and their performance for some scenes, are presented in Section 7. Finally, Section 8 outlines the conclusions and future directions of this work.

II. Research Challenges

There are a number of key research challenges that arise in a tiered network of heterogeneous sensors and computation resources. This section briefly overviews some of these issues and proposed techniques with respect to the aforementioned platform; however the statements and discussions are valid for any distributed network of embedded devices with reconfigurable resources.

To meet the severe and dynamic constraints imposed on networks of embedded sensors, the nodes of the networks need to collaborate to achieve time and location information [5]. With heterogeneous and distributed systems, new challenges arise to schedule tasks among distributed resources and to preserve power of the network to prolong the life of the network. Finally, a new paradigm of reconfiguration in a network environment needs to be addressed.

Time synchronization and localization, two key services, need to be provided to the network. Synchronization of the network's clocks allows for the fusion of sensor data. It increases the effectiveness of coordinated actuation and prolongs the life of the network by allowing power efficient duty cycling. Traditional synchronization involves sending time-stamped packets to a receiver, allowing the receiver to coordinates itself with the sender. Many sources of unknown, non-deterministic latency between the timestamp and its reception can introduce error, however.

Thus reference broadcast synchronization has been proposed [15]. Nodes send reference beacons to their neighbors, whose arrival time acts as a point of reference for comparing clocks. A receiver does not coordinate with its sender, but instead multiple receivers coordinate among themselves. Hence, the latency incurred has no effect on the quality of the time synchronization. Further modifications have been made to adapt reference broadcast synchronization to multiple hop networks. The strategy is as follows. Some nodes broadcast synchronization pulses. Receivers within

range synchronize themselves using these pulses. Nodes that receive more than one pulse have the ability to relate the time in one range to the time in another range.

Providing fine-grained localization to ad-hoc deployed nodes is the next key service. Localization allows nodes to determine their position either globally or relative to the other nodes in the network. It also allows the network to act as a positioning system. Localization allows for an ad hoc deployment of systems, since the nodes can localize themselves on a scale with the node density, independent of the environment. The localization method developed in our project works as follows. An acoustic "chirp" is emitted by the sender along with an RF message stating the time the "chirp" was emitted. By comparing time of the "chirp" detection and the actual time of emission, the time of flight can be calculated and hence the distance between the two nodes [14].

Reconfigurable hardware resources are those that can be dynamically altered to execute a particular task more efficiently. These devices can be exploited to provide both runtime flexibility and real time performance for high-level application. These two features, namely runtime flexibility and real time performance, are not simultaneously achievable by traditional pure software or hardware implementations. Utilizing dynamic hardware reconfiguration in network applications is another challenge of the system and is the focus of this paper.

Utilizing heterogeneous distributed resources and numerous pieces of application computations creates a new challenge, which is often referred to as "computation and resource management". New scheduling challenges arise, when dealing with reconfigurable distributed systems that have not been addressed by the classical scheduling literature. There are two competing goals when it comes to task scheduling on distributed resources. The first is to increase the throughput of the system and the second is the preserve the life of the system by decreasing the power consumption.

Consider the model of a directed acyclic graph (DAG), where nodes represent tasks and edges represent their dependencies. Scheduling these tasks onto heterogeneous reconfigurable resources to minimize make span involves consideration of the reconfiguration cost they would incur, along with how this cost could be amortized over multiple tasks. Currently there exists literature that examines scheduling of independent tasks and tasks with dependencies on different resources, but none of which considers paying a reconfiguration cost. Most of the well-known DAG scheduling algorithms rely on the knowledge of the critical path, and hence a polynomial algorithm for finding it. This is not the case with heterogeneous resources, especially when the cost varies depending on the reconfiguration schedule of the resources.

The scheduling of tasks onto heterogeneous resources is not only an issue for distributed systems. The intrinsically parallel vision applications can be executed on both reconfigurable and general-purpose processor embedded in efficient in-network processing sensors. The scheduling of the basic blocks of the applications boils down to the same problem of scheduling onto heterogeneous resources with reconfigurable costs.

Finally, there is the challenge of minimizing power consumption. Prolonging the network life span is directly dependent on the power consumption. This issue should be addressed at different levels of the hierarchy. For example, scheduling research continues to focus on minimizing the power consumption of the reconfigurable distributed systems while still maintaining their high throughput. By taking advantage of the timing slack of the basic blocks of the application, non-critical nodes can be executed on less power consuming resources [22].

III. Related Works

The presented project combines a number of different research areas. Many research efforts have been carried out to address challenges similar to those presented in the previous section. This section summarizes some of such works, which usually focus on a particular aspect of the system issues. Therefore, related works have been implicitly divided into three main categories: sensor network services, resource management and task scheduling and finally hardware reconfiguration.

There is an enormous amount of literature discussing embedded and distributed sensor networks, their architectures, algorithms, and applications. Authors in [7], present a good survey of existing works on sensor network related topics. These topics vary from sensing task to network applications, communication architectures, protocols and algorithms. Pottie *et al*. present a motivating article for utilizing heterogeneous resources in a sensor network [8]. They argue that a layered networking and processing architecture is suitable for many applications. Furthermore, A number of other works study distributed and cooperative detection of events [10, 9], an essential component of our framework.

Accurate time synchronization and fine-grain localization are two key services required for sensor data fusion. Many previous efforts have been addressing these two issues. For example, a fine-grain localization scheme that can provide location information for a similar experimental testbed with accuracy of about a few centimeters is presented in [13]. Researchers in [14] present a localization method based on acoustic and multi-modal ranging. This technique is implemented in the course of the current project at hand. [15, 16] present a new time synchronization methodology that synchronizes receivers of a packet with each other as opposed to traditional approach of synchronizing a sender with a number of receivers. This technique is also integrated into our framework.

Another important issue that is often addressed by research community is network energy consumption, which is directly affecting system availability and lifetime. Various research efforts try to conserve system energy at different layers of

network architecture. For example, two sample works by Schurger *et al*. address energy optimization and its implications in sensor networks [11, 12].

Dynamic hardware reconfiguration is a relatively novel issue and has not been addressed extensively before. [23] is a good survey paper on different aspects of reconfigurable computing including applications. On the other hand, existing literature usually focus on reconfigurable devices dedicated to perform a particular job, while, reconfigurable resources employed in a network can be shared among different tasks and applications. The notion of sharing a reconfigurable resource thru network creates new dimensions to the problem. [21] proposes a software architecture for a networked system that utilizes vision sensors with embedded reconfigurable devices. In addition, some efforts are made to extend the virtual machine idea to reconfigurable hardware platforms. Such ideas are inspired by Java language and strive to develop the proper framework for network reconfiguration [27, 28].

Researches in [17, 18] present the idea of dynamic hardware plugins for improving network routers. Their work is considered one of the first working systems that utilize runtime hardware reconfiguration. Hardware reconfiguration incurs a delay on the order of 100 milliseconds. Since this delay is not tolerable for many applications, many research efforts have been carried out to reduce the runtime reconfiguration delay [19, 20, 24, 25, 26].

The third and last main category of related works addresses the resource management and task scheduling issue. The problem of scheduling tasks onto resources has been widely explored. The problem can be generally stated as follows. Given n tasks and m resources schedule the tasks onto the resources to minimize the makespan. Makespan is the time in which all the tasks complete their processing. The assumption is that the execution is not preemptive, that is the tasks must run on the resources to their completion. The tasks to schedule can either be independent of each other, where they can all be executed in parallel, or they may have precedence constraints that impose an ordering on the scheduling.

The area of independent task scheduling consists of two main explorations. The first is scheduling of tasks onto homogeneous resources. [29] initially proposed a 2-approximation algorithm, which is simply arbitrary list scheduling. List scheduling is the scheduling of tasks placed in an ordered list. Various improvements have been made to the algorithm, as given in [34]. [34] presents a 3/2-approximation algorithm with time complexity of O(n.log(n)), which iteratively uses Jackson's rule.

Scheduling of independent tasks onto heterogeneous resources is more closely mapped to our problem at hand. Linear programming solutions are commonly used in this area. There is a distinction between related and unrelated heterogeneous resources in this area of research. Unrelated resources have no relation between their processing times, whereas related resources can process tasks within factor of each other. In the case of unrelated resources where the number of resources is a constant, [30] proposes an ε-approximation solution with time complexity $O(nm(nm/\varepsilon)^{m-1})$ utilizing a dynamic programming approach. [35] formulates the problem as a 0-1 integer linear programming problem. After LP relaxation, he proves that at most m-1 jobs will be scheduled on more than one resource. Thus, these tasks can then be scheduled with an exhaustive search. This proves to be a 2-approximation algorithm. [33] also gives a fully polynomial-time 2-approximation scheme based on a linear programming approach, but the fractionalized jobs are scheduled using generalized assignment techniques. Recently, [31] have put forth a combination of dynamic programming of long tasks and linear programming of short tasks to achieve an ε -approximation scheme with time complexity $n(m/e)^{\wedge O(m)}$.

Along with the work on independent task scheduling, there is much research in the area of precedence constrained task scheduling. Task dependencies are represented using directed acyclic graphs (DAGs). DAG scheduling to minimize makespan is an NP-complete problem except for a few special cases. There are many issues to consider when examining the DAG scheduling literature. The first is whether the communication cost is to be considered. The second is whether the structure of the graph and the computational costs are arbitrary or restricted. The number of processors and their connectivity are also issues. An in depth survey of various heuristics for each of these variations can be found in [32].

Although scheduling of tasks, independent and precedence constrained, has been explored before, the work on scheduling of tasks that incur a reconfiguration cost, both on homogenous and heterogeneous resources, is relatively new [60].

IV. Adaptive Tracking Case Study

A sample application of the aforementioned systems is intruder detection and object tracking on which we focus in the rest of this paper. We present a simplified version of the sensor network presented in section 1, tailored to this particular application. The system has been built in our lab as part of this research effort. Furthermore, we discuss the utilization of hardware reconfigurable resources embedded in the vision sensors to provide both real time performance and dynamic adaptability to environment variations for the tracking application. The application and system presented in this section are special cases of the general framework and hence, they highlight the significance of such systems by stressing a particular research challenge, i.e., utilizing dynamic hardware reconfiguration in networks of sensors.

A. System Framework

The hardware framework for our system is comprised of several components including: IQeye3 cameras provided by IQinVision [36], pan-tilt units to enable the actuation of the cameras, a small portable computer serving as the main controller, and a network that connects the cameras and the controller and allows them to communicate and collaborate. Figure 2 illustrates a simple view of the system framework.

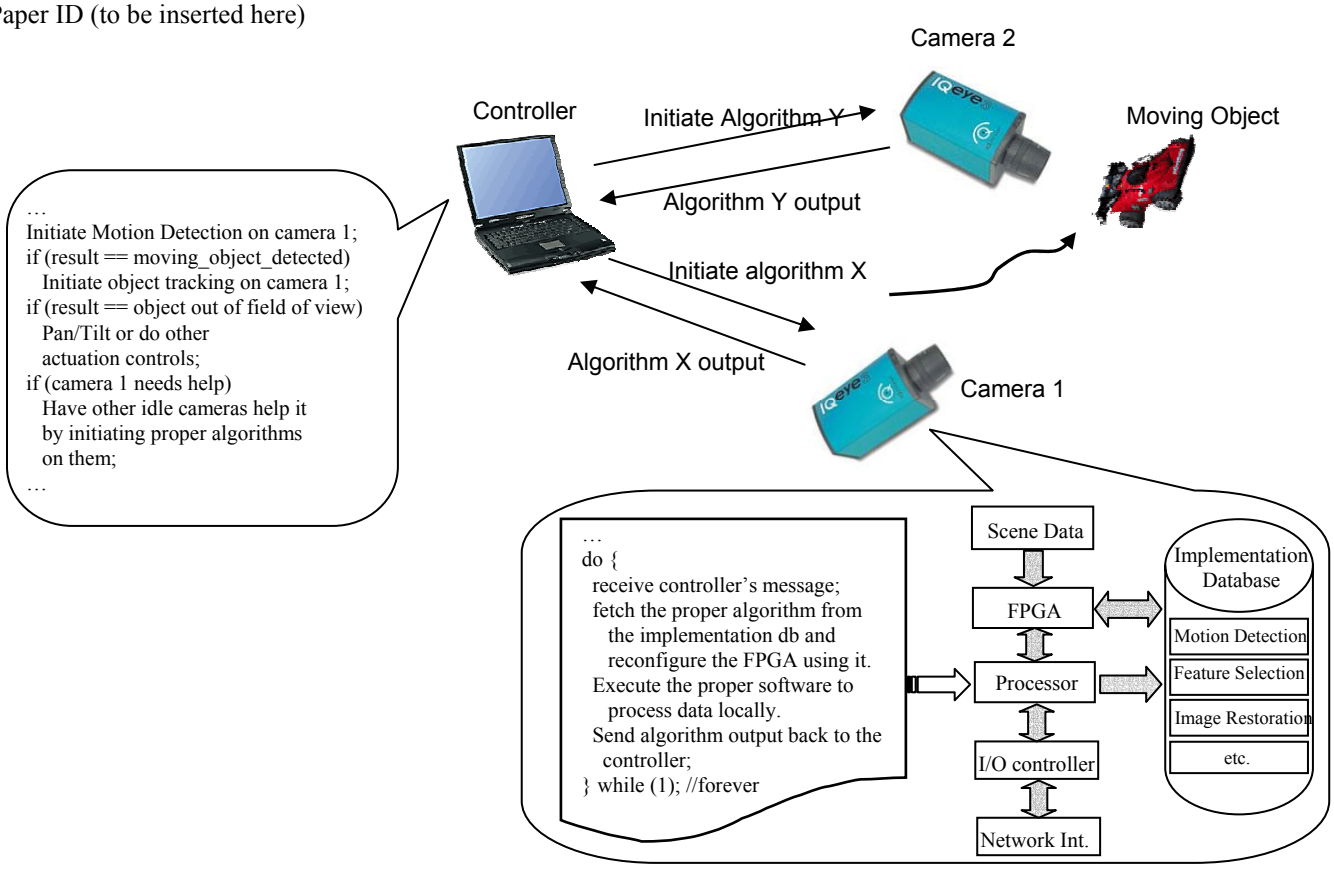

Figure 2. An overview of the tracking system architecture: Each camera has a set of the required configurations available. The controller communicates with the cameras via an implemented message passing scheme and can initiate the proper algorithm on each camera, organizing the collaboration among cameras.

An IQeye3 camera, as a "smart" vision sensor with embedded computation resources, allows input image data acquisition and processing to be collocated in the camera, which minimizes network communication overhead and facilitates scalability. The processing resources embedded in each camera include a Xilinx Virtex 1000E FPGA and a 250 MIPS PowerPC CPU. In addition, there is 4 MB of Flash RAM and 16 MB of SDRAM on each camera. Each IQeye3 camera gives full access to raw real-time image data streams and the general-purpose processor can be used for customization since a large "C" development library is available to application developers. Full networking functionality is provided by each IQeye3 camera through an Ethernet connection. It can communicate using TCP, UDP, and IP.

The IQeye3 camera can send and receive 230 Kbps over a 9-pin RS232C serial port. By supporting such communication standards, the IQeye3 cameras can be placed in various environments; while the raw and/or processed captured images can be accessed remotely. In our system, each IQeye3 camera is mounted on a pan-tilt unit, which is directly controlled by the corresponding camera via its RS232C serial interface. A pan-tilt actuation unit can be controlled using simple commands that specify the pan/tilt angle/speed/acceleration. Figure 3 illustrates the need for actuation control when an object moves out of the field of view on camera. The flow of commands from a camera to its corresponding pan-tilt unit is demonstrated.

Figure 2 demonstrates our system with two cameras and the main controller. The main supervisory controller resides on an ordinary small computer and acts as the centralized governing unit of the system by maintaining the current state, processing internal and external triggers, and coordinating the collaboration among the cameras. When the main controller receives data from one of the IQeye3 camera clients over the network, it deterministically selects the appropriate actions that should be taken by each camera (e.g. reconfiguring an embedded FPGA by swapping in a different algorithm from the database). This is performed by sending a message to the designated camera. Cameras have a database of different algorithms locally available. Therefore, they can retrieve the proper implementation according to the controller's message. The two blocks close to the main controller and the lower IQeye3 camera in Figure 2 outline the functionality of the main controller along with the idea of "implemented algorithms database" and reconfiguration at the sensor node.

B. System Application

The sample application implemented on the framework is to continuously detect and track a moving object that is within the field of view of a camera (Figure 2). We assume that the object is always moving across the camera and hence, KLT tracking scheme [37, 38, 39] can effectively track the motions. However, various parameterization and dynamic adaptations have to be performed in order to make the system robust to variations in light, objects' shape and location, etc.

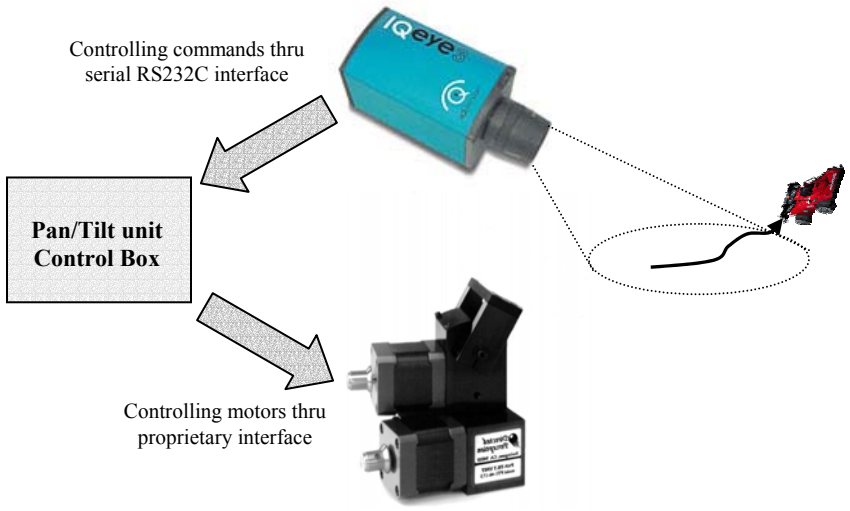

Figure 3. Each camera is mounted on a pan/tilt unit (PTU). When the object moves out of the field of view of the camera, it sends controlling commands to the PTU thru its serial interface. Consequently, PTU moves the camera in order to be able to track the object.

If the object leaves the field of view of one camera, the camera should pan or tilt to maintain the object within its field of view or it should hand off control to another camera. Depending on the light, focus and other parameters, different algorithms are used to maximize the tracking performance.

When the entire system initializes, cameras establish a connection with the main supervisory controller on the PC. First camera assumes control initially and continuously runs feature selection algorithm on its embedded FPGA. Feature selection algorithm selects points in the scene that are appropriate for tracking. Sharp corners and local intensity variations in an image usually form good features. The selected features are passed to the KLT tracking algorithm to track their motion in consequent images. The tracking algorithm has to meet the real time performance constraint.

Feature tracking has to perform some computations for each selected feature and hence, the algorithm latency increases with the number of selected features. If the number of selected features is more than a certain upper bound, the algorithm will be so slow that it cannot meet the real time performance constraint. Furthermore, accuracy will be compromised if the number of selected features is not large enough. Therefore, it is desired that the number of selected features be within a certain range.

However, as the objects in the scene, distance of the object to the camera, light conditions, lens focus and other parameters change, the number of selected features varies. For example, two runs of the algorithm on a scene with two different lighting conditions will lead to selecting less number of features for the darker scene. Our implementation can detect such conditions and can adapt itself in order to compensate the effect of variations in the scene and environment. Therefore, it is ensured that the number of selected features, and hence both latency and tracking accuracy, are kept within a certain range. This is accomplished through reconfiguration and parameterization of the algorithms running on the embedded FPGA.

Furthermore, when a moving object moves close to the edge of the image, the camera detects this situation and sends a message to the pan-tilt unit to take the appropriate action to keep the moving object within its field of view. At a certain point, the pan-tilt unit will no longer be able to pan or tilt further and the moving object will move completely out of the field of view of the camera. The camera has to surrender complete control of the scene and another camera will be forced to monitor the scene. In this situation, the camera that can no longer monitor the scene notifies the main controller by sending a message indicating the position where the moving object is located. The main controller then decides which camera should gain control and sends the proper camera a message indicating where the object is. As a result, the camera issues commands to move the pan-tilt unit so that the moving object is in the field of view of the camera. Figure 2 outlines the architecture and application of the system. A sample pseudo code running on the controller and a high-level block diagram of each camera have been demonstrated.

In such a manner, the moving object is vigilantly tracked using multiple cameras. The use of reconfigurability in our system leads to the proper tradeoff between tracking quality and latency. Moreover, it improves the system robustness to changes in the scene such as lighting and moving objects variations. Note that by use of the "hands off" approach, the cameras can collaborate in tracking an object. The object will be continuously tracked as long as the object is within the field of view of a camera.

V. Vision Algorithms Overview

In this section, we present two algorithms that are required for enhancing the image quality and tracking the motions, i.e. image restoration and feature selection. First, we outline the algorithms' underlying idea and functionality and then, we describe their sensitivity to the changes in the scene. Finally,

details of the FPGA implementation in our system will be discussed.

A. Feature Selection

In this work, we assume that the object is moving across the camera. Therefore, from camera point of view, the object in each frame is moved by a constant displacement compared to its immediately preceding frame. KLT tracking scheme [37, 38, 39], has been developed to track the objects that comply with the aforementioned motion. Note that this scheme cannot track rotations or size variations (when the object moves towards or away from the camera and its size changes from camera point of view).

KLT tracking scheme is carried out in two stages. In the first stage, called feature selection, a number of *trackable* points in the images are selected. These points, called features, show significant intensity changes compared to their neighboring pixels. Feature points are passed on to the second stage, feature tracking, in order to find their location in the consequent images. In our system, we have implemented the feature selection stage on the FPGA[2] and feature tracking is currently performed on the PowerPC embedded in the IQeye3 cameras.

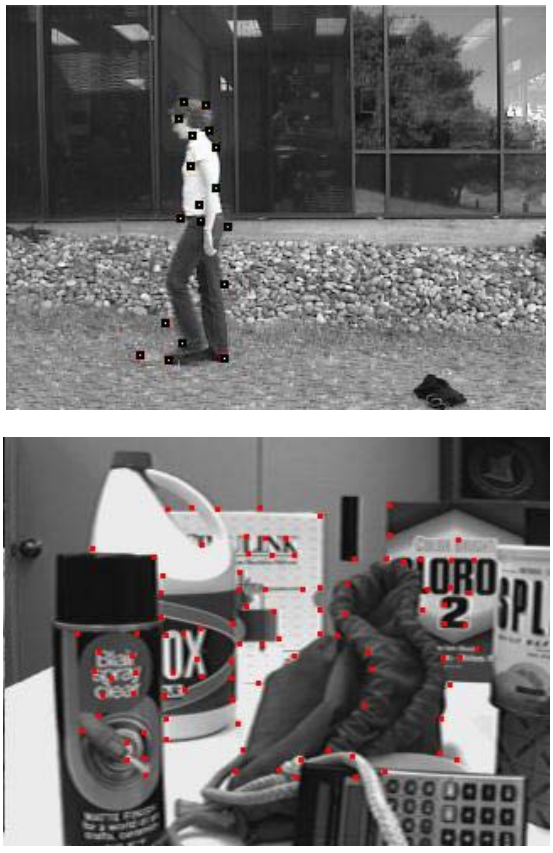

Figure 4. Sample outputs of feature selection algorithm run on a selected portion of the images. Features are denoted by black squares with white centers in the left image, and by filled dark squares in the right image.

[2] Our implementation is based on [50].

Feature selection algorithm consists of carefully choosing the points in the image, which can be easily tracked throughout a series of images. Corner points of an object, where intensity changes noticeably, are considered as good feature points. The tracking stage looks in a small patch around the location of a feature in the preceding frame, in order to find its new location after possible motion. This process is repeated for all selected features. Therefore, the latency of tracking phase linearly grows with the number of selected features. On the other hand, due to various factors including variations in the intensity of two consecutive frames and noise, some features might be lost during tracking. Therefore, a minimum number of features are required to guarantee an accurate tracking. Hence, despite ever-changing parameters of the scene, controlling the number of selected features is required.

In summary, the feature selection algorithm performs the following operations for all of image pixels [40]:

1. Calculate g_x and g_y, the intensity gradients in the x and y directions for all pixels of the image. This is done by computing the Gaussian and Gaussian derivative kernel as well as convolving these kernels in the horizontal and vertical directions.
2. Sum the gradients in the surrounding window of each pixel in order to compute the Z matrix, where
$$Z = \iint_W \begin{bmatrix} g_x^2 & g_x g_y \\ g_x g_y & g_y^2 \end{bmatrix} dx$$
3. Compute λ_1 and λ_2, the eigenvalues of the Z matrix. Let $\lambda_1 = \min(\lambda_1, \lambda_2)$. λ_1 represents the trackability of the pixel.
4. Given λ as the threshold value, If $\lambda_1 > \lambda$ then declare the pixel as a feature.

Figure 4 demonstrates the output of feature selection algorithm executed on a selected region of sample images. For example in the left image, a rectangular region around the walking girl has been chosen for selecting features. Note that the choice of two different threshold values has lead to selecting different number of features in two images. Features are denoted by black squares with white centers in the left image, and by red squares in the right image.

The number of selected features reduces with the increase of λ and vice versa. Therefore, points that are selected with higher values of λ are considered *better* features. Note that such features are also selected with small values of λ. These points are usually easier to track in consequent images. They exhibit significant intensity variation compared to their neighboring pixels.

Based on the main steps of the algorithm, it is easy to observe the effect of the changes in the scene on the number of selected features. Intuitively, increasing/decreasing the intensity value of the image pixels should increase/decrease the number of selected features with a constant λ. In reality,

brighter/darker lighting can create such a case. Therefore, different number of features will be selected for a particular scene under different lighting conditions. Furthermore, the number of selected features heavily depends on the objects in the scene. A particular threshold value will select less number of features on a round object with a few sharp corners compared to a complex object with many sharp corners and intensity variations. In addition, other parameters such as lens focus and the number of objects in the scene can affect the number of selected features.

The feature tracking stage of the KLT tracking method locks onto the selected features and strives to locate them in the next upcoming frame. Note that this is performed with the assumption that the two consecutive images differ only by a small displacement factor. The tracked features will be tracked again in the future upcoming frames. Therefore, the displacement, motion direction, velocity and other information about the motion can be inferred.

B. Image Restoration

Image restoration is a commonly used algorithm in image acquisition or processing for recovery of degraded images. Atmospheric turbulence, defocusing or motion of objects can be reasons of degradation. Restoration process recovers lost information of images by such degradation [53, 54, 55]. The following degradation model holds in a large number of applications [49]:

$$y(i,j) = d(i,j) ** x(i,j)$$

where x(i, j) and y(i, j) denote the original and observed degraded image respectively. d(i, j) represents the impulse response of the degradation system, and ** stands for two-dimensional (2D) discrete linear convolution. The goal of image restoration is to estimate x(i, j) given y(i, j) and d(i, j), however one of the main difficulties in performing an ideal image restoration is that the degradation model is not completely known. In other words, d(i, j) is not exactly defined/known at the receiver. Therefore, it might not be able to completely reconstruct the image.

Noise signal injected into the image usually exhibits quick variations and hence, is considered high frequency signal. Therefore, common realizations of noise-removal filters implement a low-pass filter, which allows the image signal to pass and filters out the high-frequency noise. A low-pass filter has no effect on low frequency image data (pixels with small variations compared to neighboring pixels) and removes the high frequency elements of the signal. As a result, the sharp edges of an image passed through a low-pass filter become blurred while the solid textures remain intact. On the other hand, blurred and defocused images have to be passed through a high pass filter in order to be restored. The high pass filter restores such images by sharpening and/or preserving their edges. Figure 5 demonstrates a simple image and the result after applying a low pass and a high pass filter on it. Note that the high pass filter preserves the sharp edges, while the low pass filter blurs them out [57].

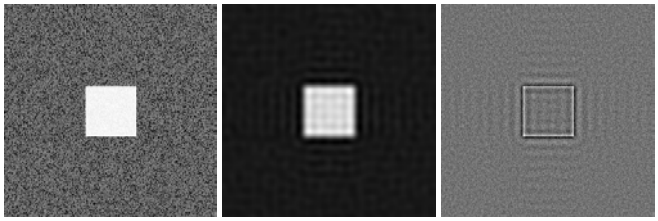

Figure 5. A sample image, its low pass, and high pass filtered versions are shown, respectively[3]. Note that the low pass filter removes quick variations in intensity and blurs out sharp edges, while the high pass filter preserves these elements.

A common implementation of image filters places an imaginary 3x3 window with filter coefficients, over a pixel in the original image and calculates the new value of this pixel in the filtered image using its old intensity value and those of the neighboring pixels. Figure 6 demonstrates the idea of such an implementation. The coefficients used in the window, specify the type of filtering operation that the filter performs. Intuitively, positive coefficients take average of close pixels to calculate the new value of a pixel, and therefore blur sharp edges. Therefore, they make low-pass filters while negative coefficients for neighboring pixels highlight the difference of the center pixel with its adjacent pixels and create a high pass filter (Figure 6). Usually, the total value of all nine coefficients is one, in order to keep the total intensity of the image intact.

The process of applying a filter on a pixel is repeated for all of the pixels in the image. Moreover, for some applications, the image is filtered many times until the residual value (the normalized amount of change between two consecutive images) is less than a given threshold. Experiments have shown that a certain number of iterations on the image, exhibit satisfactory quality for most of the scenes [49].

In our system, applying image restoration (or any other proper filter) before feature selection can enhance the image quality by sharpening the edges, and improve the quality of the selected features. Iterative application of the filter on the image requires the entire image to be accessible throughout the process. Conventional hardware implementations constantly retrieve the image from an attached memory unit and store the result back, however this is not possible in our constraint platform. In our system, the entire image is not available to the restoration module due to real time incoming stream of the scene data, which is not flow controllable. Therefore, we had to adapt the functionality of image restoration to our constrained platform. This will be thoroughly discussed in the next section.

VI. HARDWARE IMPLEMENTATIONS

In this section, we describe our system constraints and the modifications we had to make to the original algorithms in order to fit them to our platform. Moreover, we discuss the system adaptability issue and discuss its implications on

[3] http://astronomy.swin.edu.au/~pbourke/analysis/imagefilter/

hardware implementation. Throughout the paper, we assume that IQeye3 cameras, as discussed in Section 2, are the experimental platform of our system.

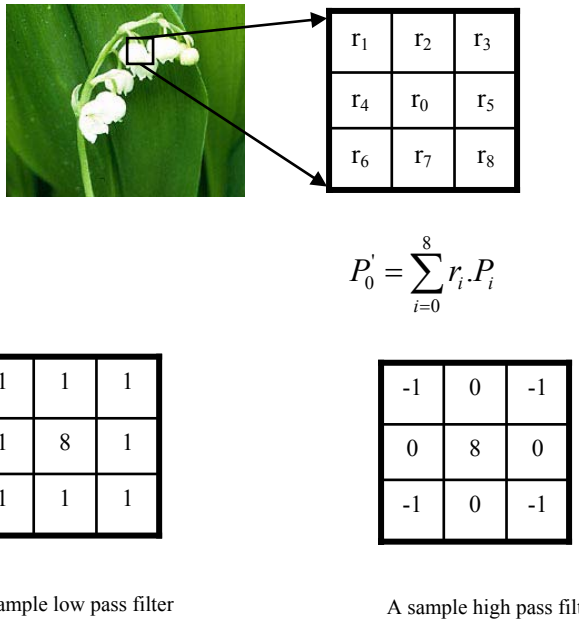

Figure 6. A filter is applied on a pixel by replacing its value with a weighted combination of its old value and its neighboring pixels. A low pass filter typically has positive coefficients, while a high pass filter has negative coefficients for neighboring pixels. Coefficients can be normalized to keep the total intensity of the image intact.

A. Platform Constraints

As described in Section 2, IQeye3 camera is the vision sensor used in our platform. Three major components of IQeye3 are the imager, embedded FPGA chip and PowerPC. The imager continuously captures scenes and injects a real-time stream of image pixels into FPGA. The incoming stream of information is not flow controllable and runs at 24 MHz. The design residing on the FPGA (called the image processing pipeline in Figure 8.a) performs several operations on the incoming stream such as image correction, windowing and down sampling. Finally, a DMA unit residing on the FPGA stores the processed scene data in the main memory. Any program running on the PowerPC can access the memory and scene data through regular software function calls. For example, a sample application running on the processor embedded in the camera implements an embedded web server that compresses the image data into jpeg format and exports the jpeg file through HTTP connection. Figure 8.a visualizes the path that each pixel goes through in order to become available to software programs running on the processor.

Within this environment and platform, applications implemented on the FPGA need to meet a number of constraints. The most important issue is the timing constraint of the design, because the imager continuously generates real-time stream of image pixels and injects the flow into the FPGA. The applications implemented on the FPGA have to process the input stream and generate the corresponding output at the same rate to avoid congestion. This forces many designs to perform their intended computations with the small on-chip memory, because using the off-chip memory units will impose additional latency, which might not be tolerable for some designs. Consequently, we have implemented a modified version of the required algorithms that work with the limited available on-chip memory.

Furthermore, there is a basic design running on the FPGA at all times. This design performs basic necessary image manipulation functions such as windowing and packetizing. Any application being mapped onto the FPGA has to integrate with this design and has to cope with its communication standards and data formats. Therefore, the algorithms cannot be used in their original form and have to be adapted to our constrained platform.

For example, the aforementioned basic FPGA design processes the image stream in Bayer pattern [56]. Therefore, any other application has to comply with this constraint and perform its computation using Bayer pattern; or convert the Bayer pattern to any other desired format, perform the computation and convert the stream back to Bayer pattern. These two major constraints, namely limited amount of on-chip memory and complying with system existing format/standard conventions, impose significant overhead in implementing new designs on the system.

B. Implementations

In this subsection, we discuss the issues involved in implementing the required algorithms, i.e., feature selection and image restoration, on our constrained platform. In general, implementing an application on the IQeye3 camera is composed of hardware and software development. Each of these two portions of the design, require a particular

development style and tool chain in order to be able to run the application on the camera. Figure 7 illustrates the block diagram and the required tool chain for developing an executable application for the camera. Software development process, which is shown on the right column of the Figure 7, is similar to an ordinary software development flow except that the compiler and linker are tailored to the particular camera platform. Similarly, for hardware design development the process shown on the left column of Figure 7 has to be followed.

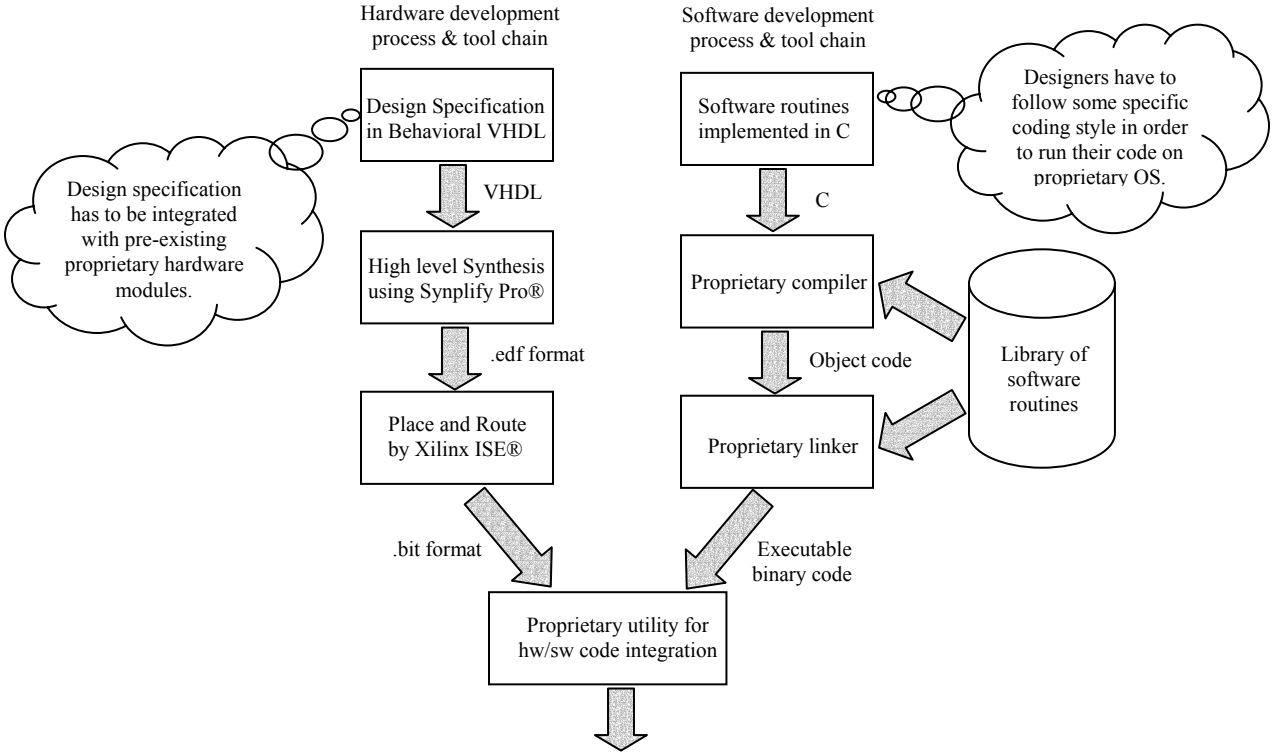

Figure 7. The process of developing a software and/or a hardware application for executing on IQeye3 camera, and the corresponding tool chain are illustrated.

For all of our hardware implementations, design specifications have been done using a combination of RTL and behavioral VHDL. ModelSim VHDL simulator [58] has been used for simulation and debugging of designs. Architectural Synthesis has been carried out using Synplify Pro from Synplicity [59], which is one of the popular FPGA synthesis tools. The result of synthesis has been saved as an EDF file.

The generated EDF format has been passed to physical synthesis stage. The physical synthesis stage, including clustering, mapping, placement, and routing has been done using Xilinx ISE package. All tools have been targeted for our embedded FPGA devices (Xilinx Virtex1000E). Finally, The FPGA chips embedded in the cameras have been programmed using the generated configuration files. Therefore, all of the designs are physically implemented on our platform and experiments are performed with actual scenes to verify the designs functionality and performance in action.

1) Feature Selection

Feature selection algorithm has been implemented on the same platform in a previous work [40, 50]. This implementation only needs to store two rows of the image data on-chip before deciding whether a pixel is a feature or not. The algorithm performs local computations in a 3x3 window around a pixel and compares the result with a *fixed* threshold for determining features. The value of threshold used in this implementation has to be specified at design time. Then, the design undergoes conventional architectural and physical synthesis phases and the resulting FPGA configuration stream is mapped onto the FPGA embedded in the camera.

While this implementation works well in practice, it does not have any control on the number of selected features. Moreover, the value of threshold cannot be altered easily. The feature selection's threshold has been implemented as a constant, which should be specified at design time. Therefore, altering the threshold forces the designer to repeat the entire design flow, which can take up to 30 minutes and is not tolerable for real time applications.

Various parameters such as objects' shape, scene light and lens focus can affect the number of selected features. As mentioned before, the selected features are passed to the tracking phase. The latency of the tracking grows, while its accuracy drops, with the increase of feature count. Therefore, the number of selected features has to be controlled in order to

maintain a proper tradeoff between tracking latency and its accuracy.

We have started from the implementation in [50] and have modified the original design such that the threshold value can be controlled by a program running on camera PowerPC at runtime. Specifically we have developed registers that can be read/written by a software program running on the PowerPC. The hardware design has also been modified to read its threshold value from the register, without losing its synchronous operation with other parts of the basic design. Note that, the software program can alter the register contents at any time during processing of a frame and therefore, the design has to be able to handle asynchronous incoming events.

Our implementation can dynamically tune the feature selection algorithm running on the FPGA. According to the algorithm, if the threshold used in feature selection is too low for a particular scene, we get too many features and if the threshold is too high, we get too few features. Therefore, given a target number of features desired, we increase the threshold if we get features more than the target and decrease if we get less.

Note that the actual feature selection performs its computations on the FPGA and exhibits real time performance. The threshold controlling entity is a small program running on the camera PowerPC, which counts the number of selected features and controls the threshold value accordingly.

2) Image Restoration

Image restoration has a variety of implementations and iterative method is a widely used one. The purpose is to estimate the original image given the degraded image. Common restoration methods perform operations on the entire image iteratively. Following each iteration, the normalized difference between current and immediately preceding image, called residual value, is calculated. Iterations are stopped when the restored image converges with insignificant residual ε [49].

As discussed in Subsection 4.1, our constrained platform does not allow the entire image to be stored on the FPGA. On the other hand, accessing the off-chip memory iteratively will impose additional latency on the algorithm, which is not affordable because of the real time performance constraint of system applications.

We have made several modifications to adapt the original method to our environment. Firstly, instead of globally iterating over the entire image, we iterate over local windows, where the size of window can be from 3x3 to the entire image. As the window gets smaller, the restoration quality drops since the center pixel does not have any information about pixels out of the restoration window. However, this enables processing of image stream using a small-sized storage.

Figure 8.a illustrates the path that each image pixel goes through to be processed in our system cameras. Image sensor converts the scene into a non flow-controllable stream of pixels flowing into the FPGA. The proprietary image processing pipeline implemented on the FPGA performs various computations on the incoming flow of pixels and finally stores the result in the system memory, where the software applications running on the camera processor (PowerPC) can access it.

The image restoration algorithm has been implemented as one of the stages in the pipeline (Figure 6.b). Therefore, it does not have access to the entire image pixels at any point of time (assuming no off-chip data communication). Note that image pixels are revealed to the system starting from upper left corner of the image flowing to the right. When a row is finished, the flow of pixels moves down a row and again start from left to right. As Figure 6.b visualizes, the amount of memory required for implementing a 3x3 restoration window is a bit larger than two rows of the image. The FPGA devices embedded in the system cameras have enough BlockRAMs available to store two rows of the image on-chip. Therefore, the restoration algorithm can be performed without any off-chip communication.

In general, for a restoration window of size nxn, ((n-1).rows + n) pixels need to be stored on the chip. Each FPGA device contains a certain number of logic blocks and BlockRAMs. Hence, the window size cannot grow beyond physical limitations of the target FPGA. For example, our system's embedded FPGA (Xilinx Virtex1000E) allows the window size to grow up to 15 for processing the widest images. The

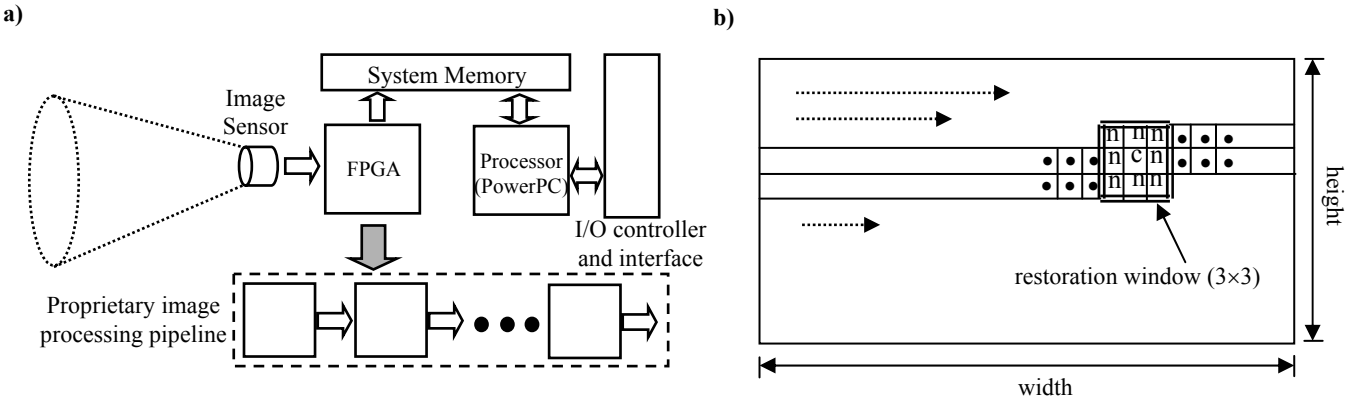

Figure 8. a) Block diagram of the camera illustrating the path each image pixel goes through in order to be processed. The image processing pipeline residing on the FPGA is not disclosed due to copyright issues. b) Restoration window implemented as one of the blocks in image processing pipeline. Pixels stream in starting from the upper left corner of the image.

maximum width of images in our system is 1280 pixels.

In addition, we have unrolled local iterations of the algorithm on a 3x3 window a priori, and therefore, the current implementation performs an equivalent but more efficient computation for restoration of each pixel. Current implementation performs a single step evaluation of each window in order to calculate the new value of the center pixel, as opposed to iterating over the window. Table 1 summarizes the area improvement of the unrolled implementation compared to the original implementation, which iterates 40 times over each pixel.

Researchers in [49] have studied tradeoffs of restoration performance and quality with changes in restoration window size. According to their work, 3x3 restoration windows reflect reasonable restoration quality for many applications. The restoration algorithm used in this work, implements a high pass filter with 2 and -0.125 as coefficients for center and neighboring pixels, respectively.

Varying the restoration window size, leads to accuracy-memory requirement tradeoff. Small restoration windows need smaller on-chip storage, however their quality is not as good as larger restoration windows. On the other hand, larger windows improve the restoration quality at the price of higher memory requirement. Note that memory requirement inversely correlates with the system performance.

Implemented Design	Block RAMs	CLBs
Basic design + Feature selection	51 out of 96 (53%)	9170 out of 24576 (37%)
Basic design + Feature selection + Original image restoration	66 out of 96 (68%)	12278 out of 24576 (49%)
Basic design + Feature selection + Unrolled image restoration	64 out of 96 (66%)	9454 out of 24576 (38%)

Table 1. The breakdown of hardware resources used by different portions of the designs. Note that the unrolled version of image restoration frees up 2% of block RAMS and 11% of CLBs for Xilinx1000E device.

VII. Experiments

In this section, we present the framework and results of our experiments. First, we describe the platform and designs used in conducting the experiments. We address the issue of dynamic system adaptation to the environment variations in this section. Then, we present the results of our approach for a number of scenes and compare them with a traditional non-adaptive system results.

A. Experimental Setup

We have implemented the feature selection and image restoration algorithms (discussed in Sections 3 and 4) on IQeye3 cameras. The threshold value in the feature selection algorithm can be dynamically adjusted through a software program running on the PowerPC of the camera.

Furthermore, the implemented image restoration algorithm can be dynamically disabled or enabled through system reconfiguration. If the quality of the image is not good enough, then the FPGA will be reconfigured to enable the image restoration before feature selection. The quality of images can be determined by examining the value of the threshold in feature selection for selecting a certain number of features. Lower threshold values correspond to lower quality features and blurred corners. On the other hand, image restoration can alter the original image if it is not degraded to some degree. Therefore, we need to disable it for cases that the image quality is reasonable.

B. Experimental Results

In the following sets of experiments, we examine the effect of our proposed techniques. The first two sets of experiments demonstrate the quality of automatically adjusted threshold compared to the original fixed threshold feature selection. The third experiment shows how image restoration can affect the performance of feature selection. In all experiments, automatically adjusted threshold targets for 150 features with 10% tolerance range, i.e. the number of selected features should be in the (135-165) range.

One example, where dynamically adaptive feature selection finds its use, is in the environments with variations in lighting. This applies to outdoor places where the natural lighting changes throughout the time. Another example is indoor scenes under various lighting conditions. For the first set of experiments, we varied the lighting condition in the laboratory and observed the results of the feature selection application.

Figures 9.a, 10.a and 11.a show the result of feature selection with fixed threshold, called FS-FIX, for an object under three different lighting conditions. Figure 9.b, 10.b and 11.b show the results of feature selection with automatically adjusted threshold, called FS-AUTO, for the same object and lighting conditions.

Figures 9.a and 9.b show the result of both FS-FIX and FS-AUTO under normal lighting. Both implementations select about 150 features (with 10% tolerance). Figure 10.a and 10.b illustrate the same object under similar lighting, which is brighter than the previous settings used in Figure 9. Extra brightness causes edges and corners to have greater intensity difference from their adjacent pixels, therefore a larger number of points are chosen as features. Figure 10.a shows many unnecessary features chosen whose count is 1150. This is too many compared to the target feature count, 150. FS-AUTO increases the threshold value from 512 to 1552 and chooses 150 features in Figure 10.b. It selects features at almost same locations as in Figure 9.b even after the significant change in brightness.

Figures 11.a and 11.b are taken under dark lighting. The object is observable by eyes, but FS-FIX is unable to find any features, since the intensity variations are not large enough for the fixed threshold value. However, FS-AUTO successfully decreases the threshold value from 512 to 160 and finds 156

features. Locations of features are almost same as Figures 9.b and 10.b.

The aforementioned set of experiment verifies the efficiency of our approach in implementing a system robust to lighting variations through dynamic adaptation of the system. However, the advantage of our implementation is not limited to handling lighting variations. We have carried out another set of experiments to show that this technique can assist in handling other realistic scenarios, such as object's shape variations and multiple object cases.

Figures 12.a and 13.a show two different objects that have been processed by FS-FIX to select some features on them. As expected, FS-FIX has no control over the number of selected features. Therefore, the number of selected features on a round object, such as a computer mouse shown in Figure 12.a, is not large enough, while this number on a complex object with many sharp corners is too large. In fact, FS-FIX chooses 42 features in Figure 12.a, which is far less than our target, 150. Similarly, it selects 572 features in Figure 13.a, which is almost 4 times more the desired number of features.

Figures 12.b and 13.b illustrate the same objects shown in Figures 12.a and 13.a, however these objects are processed by FS-AUTO. The object in Figure 12.b is round and does not have enough sharp corners, however, FS-AUTO successfully decreases the threshold value until it selects 154 features with a new threshold value of 300. Extra features are observed at the left end of the object. Feature tracking algorithm can utilize this additional information for better tracking. The object in Figure 13.b is a toy car that has many colorful parts and sharp edges, which are potentially good candidates for features. As presented earlier, FS-FIX uses a fixed threshold value for selecting features and it selects 572 features. Unnecessarily many features are observed around the wheel and wire part of the object in Figure 13.a. FS-AUTO adjusts the threshold value to select fewer features. It selects 152 features with a new threshold value of 912 (Figure 13.b).

As discussed above, FS-AUTO is able to select proper number of features for any type or number of objects. It certainly is a better solution than FS-FIX, which works only for limited type or number of objects, but it cannot solely handle all possible cases. One example is where multiple objects are present in a single scene. Therefore, the camera lens can be focused on only one of them. Under this situation, most of the features will be placed on one well-focused object and the rest of the objects will not be tracked.

Figure 14.a demonstrates such a situation where the puppy doll that is close to the camera is better focused than the mouse located farther from the camera. FS-AUTO cannot select any features on the mouse. This is generally a hard problem to solve. However, by employing image restoration, the problem is alleviated to some degree. In Figure 14.b, FS-AUTO selects features on the same scene as Figure 14.a, however the image is first restored using the implemented image restoration algorithm. Restoration enhances the clarity of the edges and corners of both objects. After applying the image restoration algorithm, features are selected on the mouse as well as the puppy. Moreover, the number of features is balanced on the two objects. Note that the choice of enabling or disabling the image restoration algorithm is made on the fly and the system dynamically adapts itself to environment changes.

Figures 15.a and 15.b clearly demonstrate the effect of image restoration on feature selection results. In Figure 15.a, the lens is not well focused on the object. Although FS-AUTO can adjust its threshold to select the required number of features, features do not show satisfactory quality. The low threshold value used for selecting the features highlights this fact. Figure 15.b shows the result of the same algorithm after dynamically enabling the image restoration before selecting the features in the image. Image restoration enhances the image quality by sharpening the edges. Therefore, the threshold value for selecting the same number of features on the restored image is larger. Hence, the features' quality has been enhanced and features with larger intensity difference compared to their adjacent pixels have been selected.

Note that sharp and clear images do not need to be restored before being passed to feature selection algorithm. Failure to do so might degrade the image quality by adding noise to the image and can create fake features in the image. Therefore, the system should be reconfigured to enable or disable image restoration based on the requirements. In our system, we can dynamically enable or disable this module before selecting the features.

Figure Number	Threshold	Feature count
9.a	512 (Fixed)	152
9.b	465	148
10.a	512 (Fixed)	1150
10.b	1552	150
11.a	512 (Fixed)	0
11.b	160	156
12.a	512 (Fixed)	42
12.b	300	154
13.a	512 (Fixed)	572
13.b	912	152
14.a	290	164
14.b	664	162
15.a	1279	146
15.b	2083	148

Table 2. Feature selection threshold value and feature count for figures presented in experimental results section.

Table 2 summarizes the number of selected features and the utilized threshold value for selecting those features for images

presented in this section. The enhanced performance of FS-AUTO compared to FS-FIXED in terms of number of selected features is evident. Furthermore, the effect of image restoration on the threshold value used in FS-AUTO can be observed. Note that applying image restoration on the blurry image shown in Figure 15.a sharpens its edges and corners (see Figure 15.b) and increases the required threshold in FS-AUTO. This in turn corresponds to features that are easier to track in the feature tracking stage. This has been highlighted in the last two rows of Table 2.

VIII. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we presented the idea of dynamic system reconfiguration in order to be able to adapt to the external events. A collaborative tracking system has been built and presented as the experimental framework for verifying the idea. Experimental results show that the idea is effective in practice and the system can function in a wide range of working conditions.

Particularly, we have implemented automatic adjustment of threshold value in feature selection algorithm, and dynamic enabling of image restoration for enhancing the image quality. These techniques have been integrated into our system framework. It has been shown that our approach is effective for dynamically adapting to various lighting and lens focus conditions in practice.

Future works include the integration of tracking phase of the KLT feature-tracking method into our current system, enhancing the collaboration schemes and applying the system reconfiguration idea to other applications or application domains.

REFERENCES

[1] D. Tennenhouse, "Proactive Computing," *Communications of the ACM*, May 2000, vol. 43, no. 5, pp. 59–66.

[2] M. Weiser, “The Computer for the 21st Century”, *Scientific American*, Sept. 1991, vol. 265, no. 3, pp. 94–104.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, "System Architecture Directions for Networked Sensors", *in Proc. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 93-104, 2000.

[4] D. Estrin *et. al.*, "Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers," Committee on Networked Systems of Embedded Computers, Computer Science and Telecommunications Board, National Research Council, Washington, DC, 2001.

[5] D. Estrin, R. Govindan, J.S. Heidemann, S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks", Mobile Computing and Networking, pp. 263-270, 1999.

[6] H. Wang, D. Estrin, L. Girod, "Preprocessing in a Tiered Sensor Network for Habitat Monitoring", *in EURASIP JASP special issue of sensor networks*, Vol. 2003, No. 4, pp. 392-401, March 15, 2003.

[7] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "Wireless sensor networks: a survey", *Computer Networks*, Vol. 38, No. 4, pp. 393-422, 2002.

[8] G. J. Pottie, W. J. Kaiser, "Wireless integrated network sensors", *Communications of the ACM*, Vol. 43, No. 5, pp. 51-58, 2000.

[9] R. Viswanathan, P. K. Varshney, "Distributed Detection with Multiple Sensors, Part I:Fundamentals", *Proceedings of the IEEE*, Vol. 85, No. 1, pp. 54-63, 1997.

[10] J. Agre, L. Clare, "An Integrated Architecture for Cooperative Sensing Networks", *IEEE Computer Magazine*, Vol. 33, No. 5, pp.106-108, May 2000.

[11] Curt Schurgers, Gautam Kulkarni, Mani B. Srivastava, "Distributed On-Demand Address Assignment in Wireless Sensor Networks", *IEEE Transactions on Parallel and Distributed Systems*, Vol.13, No.10, pp. 1056-1065, Oct. 2002.

[12] Curt Schurgers, Vlasios Tsiatsis, Saurabh Ganeriwal, Mani B. Srivastava, "Optimizing Sensor Networks in the Energy-Latency-Density Design Space," *IEEE Transactions on Mobile Computing*, Vol.1, No.1, pp. 70-80, Jan.-March 2002.

[13] A. Savvides, C.-C. Han, M. B. Srivastava, "Dynamic fine-grained localization in ad-hoc networks of sensors", *in Proc. 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '01)*, pp. 166-179, July 2001.

[14] L. Girod, D. Estrin, "Robust range estimation using acoustic and multimodal sensing", *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, October 2001.

[15] J. Elson, L. Girod, D. Estrin, "Fine-grained network time synchronization using reference broadcasts", *in Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI 2002)*, December 2002.

[16] J. Elson, D. Estrin, "Time Synchronization for Wireless Sensor Networks", *Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS): Workshop on Parallel and Distributed Computing Issues in Wireless and Mobile Computing*, April 2001.

[17] E.L. Horta, J.W. Lockwood, D.E. Taylor, D. Parlour, "Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfiguration", *Design Automation Conference (DAC)*, June 2002.

[18] D.E. Taylor, J.S. Turner, J.W. Lockwood, "Dynamic Hardware Plugins (DHP): Exploiting Reconfigurable Hardware for High-Performance Programmable Routers", *IEEE OPENARCH 2001: 4th IEEE Conference on Open Architectures and Network Programming*, 2001.

[19] S. Ghiasi, M. Sarrafzadeh, "Optimal Reconfiguration Sequence Management", *Asia South Pacific Design Automation Conference (ASPDAC)*, January 2003.

[20] Z. Li, K. Compton, S. Hauck, "Configuration Cache Management Techniques for FPGAs", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 22-36, 2000.

[21] R. Kumar, S. Ghiasi, M. Srivastava, "Dynamic Adaptation of Networked Reconfigurable Systems", *Workshop on Software Support for Reconfigurable Systems (SSRS)*, February 2003.

[22] S. Ghiasi, K. Nguyen, E. Bozorgzadeh, M. Sarrafzadeh, "On Computation and Resource Management in an FPGA-based Computing Environment", *A poster in International Symposium on Field-Programmable Gate Arrays (FPGA)*, February 2003.

[23] K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", *ACM Computing Surveys*, Vol. 34, No. 2. pp. 171-210. June 2002.

[24] K. Compton, Z. Li, J. Cooley, S. Knol, S. Hauck, "Configuration Relocation and Defragmentation for Run-time Reconfigurable Computing", *IEEE Transactions on VLSI Systems*, Vol. 10, No. 3, pp. 209-220, June 2002.

[25] Z. Li, S. Hauck, “Configuration Compression for Virtex FPGAs”, *IEEE Symposium on FPGAs for Custom Computing Machines*, 2001.

[26] Z. Li, S. Hauck, "Configuration Prefetching Techniques for Partial Reconfigurable Coprocessor with Relocation and Defragmentation", *ACM/SIGDA Symposium on Field-Programmable Gate Arrays*, pp. 187-195, 2002.

[27] Y. Ha, P. Schaumont, L. Rijnders, S.Vernalde, F. Potargent, M. Engels, and H. De Man, "A scalable Architecture to Support Networked Reconfiguration", *Proceedings of IEEE ProRISC*, pp. 677-683, November 1999.

[28] Y. Ha, P. Schaumont, M. Engels, S. Vernalde, F. Potargent, L. Rijnders, H. De Man, "A Hardware Virtual Machine for the Networked Reconfiguration", *11th IEEE International Workshop on Rapid System Prototyping(RSP)*, 2000.

[29] R.L. Graham, “Bounds on Multiprocessing Timing Anomalies”, *SIAM J. Applied Math.,* Vol. 17, pp. 416-426, 1969.

[30] E. Horowitz, S. Sahni, "Exact and Approximate Algorithms for Scheduling Nonidentical Processors", *Journal of the Assosiation for Computing Machinery,* Vol 23, pp. 317-327, 1976.
[31] K. Jansen, L. Porkolab, "Improved Approximation Schemes for Scheduling Unrelated Parallel Machines", *Proceeding of the 31st Annual ACM Symposium on the Theory of Computing (STOC 99),* pp. 408-417, 1999.
[32] Y. K. Kwok, I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Tasks Graphs to Multiprocessors", *ACM Computing Surveys,* Vol. 31, pp. 406-471, 1999.
[33] J.K. Lenstra, D.B. Shmoys, E. Tardos, "Approximation Algorithms for Scheduling Unrelated Parallel Machines", *Mathematical Programming*, Vol. 46, pp. 259-271, 1990.
[34] E. Nowicki, c. Smutnicki, "An Approximation Algorithm for a Single-Machine Scheduling Problem with Release Times and Delivery Times", *Discrete Applied Math*., Vol. 48, pp. 69-79, 1994.
[35] C. N. Potts, "Analysis of a Linear Programming Heuristic for Scheduling Unrelated Parallel Machines", *Discrete Applied Math.*, Vol. 10, pp. 155-164, 1985.
[36] IQinVision Online Documentations, IQinVision Inc., http://www.iqinvision.com.
[37] B. Lucas, T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision", *International Joint Conference on Artificial Intelligence*, pp. 674-679, 1981
[38] C. Tomasi, T. Kanade, "Detection and Tracking of Point Features", *Carnegie Mellon University Technical Report CMU-CS-91-132*, April 1991.
[39] J. Shi, C. Tomasi, "Good Features to Track", *IEEE Conference on Computer Vision and Pattern Recognition*, pages 593-600, 1994
[40] A. Benedetti, P. Perona, "Real-time 2-D Feature Detection on a Reconfigurable Computer", *IEEE Conference on Computer Vision and Pattern Recognition*, June 1998, Santa Barbara, CA.
[41] P. Athanas, L. Abbott, "Addressing the Computational Requirements of Image Processing with a Custom Computing Machine: An Overview", *in Proceedings of the 2nd Workshop on Reconfigurable Architectures*, April 1995, Santa Barbara, CA.
[42] X. Feng, P. Perona, "Real Time Motion Detection System and Scene Segmentation", *CDS TR CDS98-004*, Caltech, 1998
[43] M. Sarrafzadeh, A.K. Katsaggelos, S.P. Kumar, in "Parallel Architectures for Iterative Image Restoration", *Kluwer Academic*, M. Bayoumi editor, 1991.
[44] K. Melhorn, F. Preparata, "Area-Time Optimal VLSI Integer Multiplier with Minimum Computation Time", *Information and Control*, Vol. 58, pp. 137-156, 1983.
[45] G. Bilardi, M. Sarrafzadeh, "Optimal VLSI Circuits for Discrete Fourier Transform", *Advances in Computing Research*, Vol. 4, pp. 87-101, 1987.
[46] D.J. Li, L. Jiang, T. Isshiki, H. Kunieda, "New VLSI Array Processor Design for Image Window Operations", *IEEE Transactions on Circuits and Systems*, Vol. 46, No. 5, pp. 635-640, May 1999.
[47] S. Ghiasi, H.J. Moon, M. Sarrafzadeh, "Collaborative and Reconfigurable Object Tracking", *Engineering of Reconfigurable Systems and Algorithms*, 2003
[48] F. Cuzzolin, A. Bissacco, R. Frezza, S. Soatto, "Towards Unsupervised Detection of Actions in Clutter", *Proc. of the Asilomar Conference on Signals, Systems and Computers*, 2002.
[49] S. Ogrenci Memik, A. K. Katsaggelos, M. Sarrafzadeh, "FPGA Implementation and Analysis of an Iterative Image Restoration Algorithm". *IEEE Transactions on Computers,* vol. 52, no.3, March 2003.
[50] M. Maire, "Design and Implementation of a Realtime Visual Feature Tracking System on a Programmable Video Camera", Technical Report, California Institute of Technology, 2002.
[51] Xilinx Online Documentations, Xilinx Inc., http://www.xilinx.com.
[52] H.C. Andrews, B.R. Hunt, "Digital Image Restoration". *Prentice Hall*, 1977.
[53] H.J. Trussel, B.R. Hunt, "Improved Methods of Maximum A Posteriori Restoration", *IEEE Transactions On Computers*, Vol. 28, 1979.
[54] J. Biemond, J. Rieske, J.J. Gerbrands, "A Fast Kalman Filter for Images Degraded by Both Blur and Noise", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1983.
[55] A.K. Katsaggelos, "Iterative Image Restoration Algorithms", *Optical Eng*., Vol. 28, pp. 735-748, July 1989.
[56] B. Fortner, T.E. Meyer, T. Meyer, "Number by Colors: A Guide to Using Color to Understand Technical Data", *Springer Verlag*, 1997.
[57] J. C. Russ, "The Image Processing Handbook", *CRC Press*, 1999.
[58] ModelSim® product manual, Model Technology Inc. , http://www.model.com.
[59] Synplify Pro® product manual, Synplicity Inc., http://www.sinplicity.com.
[60] Ani Nahapetian, Soheil Ghiasi, Majid Sarrafzadeh, "Task Scheduling on Heterogeneous Resources with Heterogeneous Reconfiguration Costs", International Conference on Parallel and Distributed Computing and Systems, November 2003.

Figure 9.a. 152 features are selected by fixed threshold (FS-FIX) under default lighting conditions.

Figure 9.b. 148 features are selected by automatic threshold adjustment (FS-AUTO) under normal lighting.

Figure 10.a. 1150 features are selected by fixed threshold (FS-FIX) under bright lighting.

Figure 10.b. 150 features are selected by automatic threshold adjustment (FS-AUTO) under bright lighting.

Figure 11.a. No feature is selected by fixed threshold (FS-FIX) under dark lighting.

Figure 11.b. 156 features are selected by automatic threshold adjustment (FS-AUTO) under dark lighting.

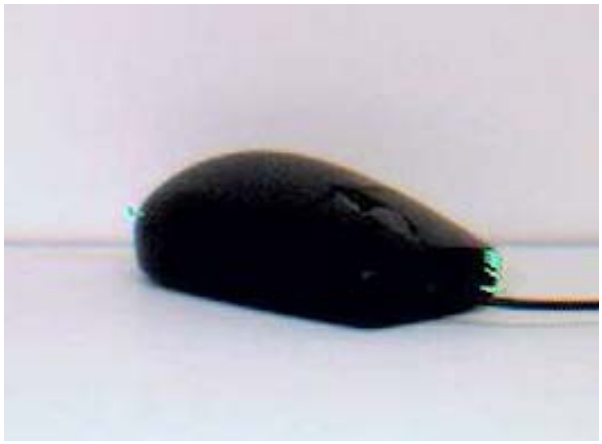

Figure 12.a. 42 features are selected on a simple object with FS-FIX.

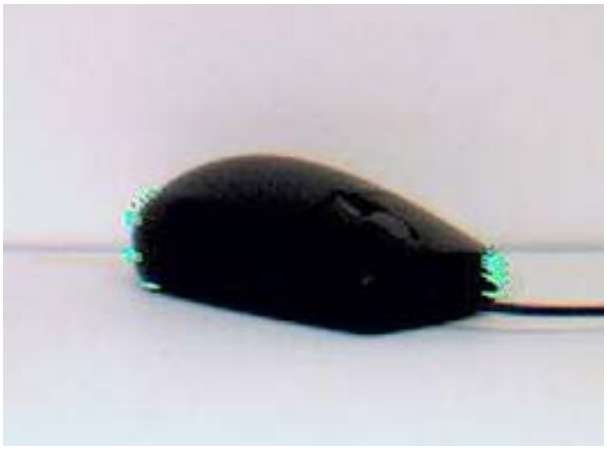

Figure 12.b. FS-AUTO selects 152 features on the object shown in Figure 10.a.

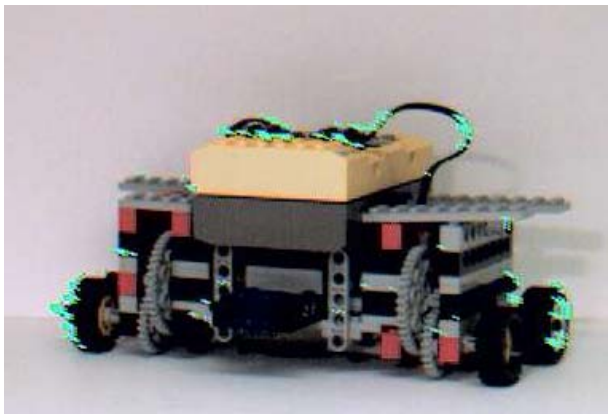

Figure 13.a. 572 features are selected on an object with sharp edges, using FS-FIX.

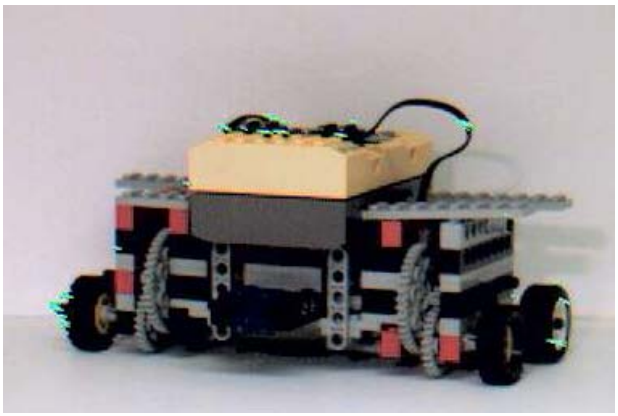

Figure 13.b. FS-AUTO selects 152 features on the object shown in Figure 11.a.

Figure 14.a. FS-AUTO selects all of features on one of the objects (puppy in this example).

Figure 14.b. Applying image restoration before feature selection enhances the image quality by sharpening the edges and distributes the selected features on both objects.

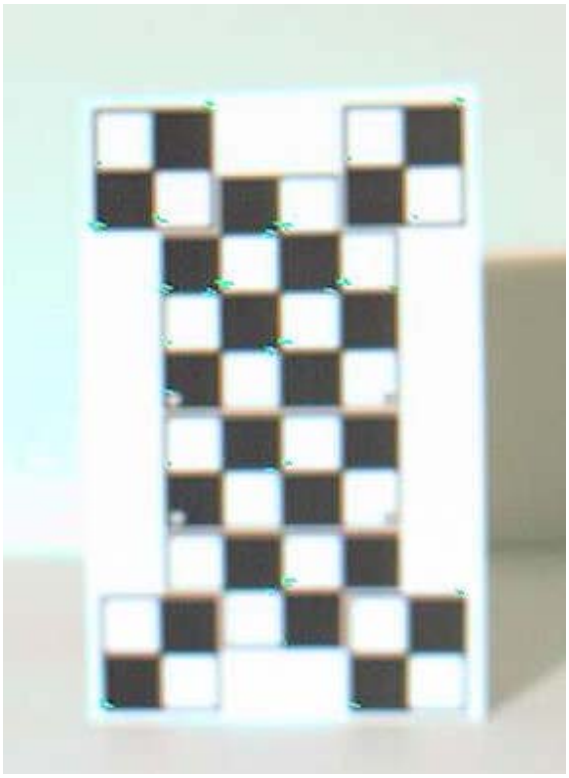

Figure 15.a. Selected features using automatically threshold adjustment (FS-AUTO) without image restoration

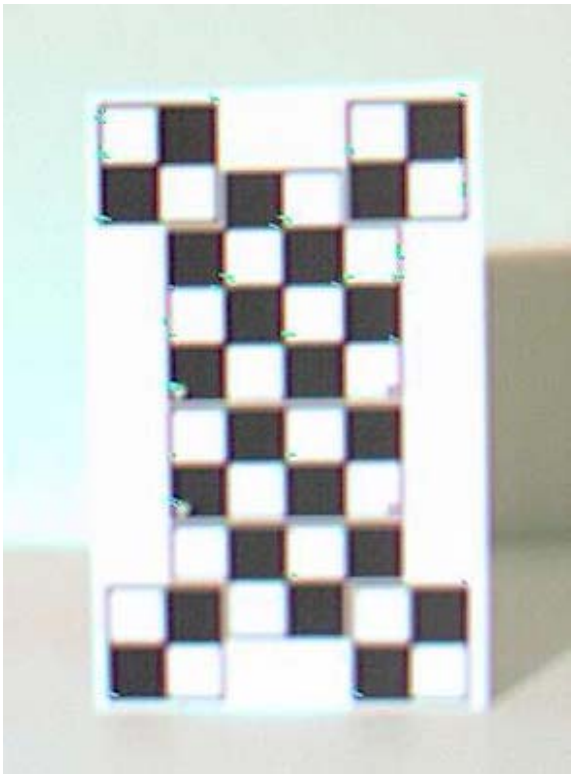

Figure 15.b. Image restoration sharpens the edges and corners. Therefore, FS-AUTO selects better (selected with larger threshold) features.