Collaborative and Reconfigurable Object Tracking

(Invited Paper)
Soheil Ghiasi, Hyun J. Moon, Majid Sarrafzadeh
Computer Science Department
University of California, Los Angeles
{soheil, hjmoon, majid}@cs.ucla.edu

Abstract

Many vision applications perform intensive computations and demand hardware implementations in order to exhibit real time performance. Furthermore, these applications are hard to pre-characterize and can take different paths according to events happening in the scene at runtime. Hence, reconfigurable hardware devices are envisioned as the proper platform to implement such applications, providing both real time performance and dynamic adaptability for the system. In this paper, we present a collaborative and reconfigurable object tracking system that has been built as part of our work. We justify the need for dynamic adaptation of the system through scenarios and examples. Experimental results on a set of scenes advocate the fact that our system works effectively for different scenes and scenario of events through reconfiguration.

1. Introduction

Unsupervised detection of events is widely used in many different applications. Such applications require the system to automatically detect the events happening in its surrounding area and take proper actions according to these events. Moreover, real time response to external events is usually another requirement, because of the nature of the applications; examples of which include traffic management and intelligent intruder detection. Various image-processing algorithms have been developed for this class of applications. These algorithms usually perform intensive computations and hence, require powerful computational resources in order to comply with the real time performance requirement.

Image processing algorithms are generally very intensive. Therefore, many embedded processors dedicated to image data collection and/or processing cannot meet the real time performance constraint. However, most of image-processing algorithms perform similar computations for all of the pixels of an image that only requires local information. Therefore, they are considered as *highly parallel* computations that exhibit substantial speedup when implemented on a dedicated hardware unit. Hence, hardware implementation is the only viable solution for most of the real time image-processing systems. Researchers have reported many efficient hardware implementations of such algorithms [8, 9, 10, 11, 12].

Moreover, the image-processing algorithms implemented in a system work based on some assumptions. Examples include the number of moving objects, their shape and their motion type. Based on the events happening in scene, these assumptions might become invalid. For example, KLT tracking scheme [5, 6, 7] assumes that the moving object moves across the camera and its size doesn't change from the camera point of view. However, if the object moves towards the camera, this assumption is no longer valid and KLT tracking will not be effective anymore. Therefore, the system has to be able to adapt to external events. However, the external events are hard - and in some cases impossible - to pre-characterize. Hence, it is practically impossible to determine the required algorithms a priori.

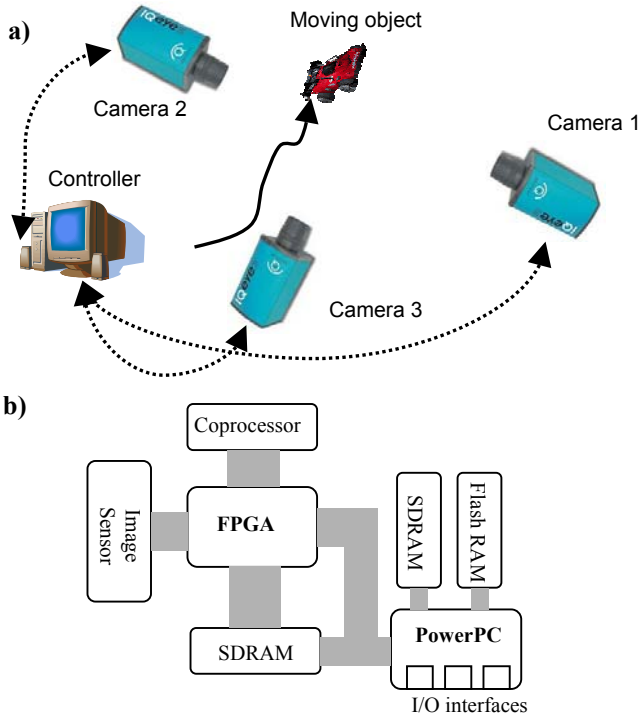


Figure 1. a) The implemented target tracking system using Iqeye3 cameras with embedded processors. b) IQeye3 camera architecture. Courtesy of IQinVision Inc.

The aforementioned arguments, introduce the reconfigurable fabrics as the only viable solution for implementing such applications. Reconfigurable hardware units not only demonstrate real time performance by exploiting the intrinsic parallelism of image processing algorithms, but also provide the required flexibility and adaptability for the system. This cannot be achieved by traditional pure software or hardware implementations.

Figure 1.a depicts an intruder detection and object tracking system that has been built as part of this work. The system

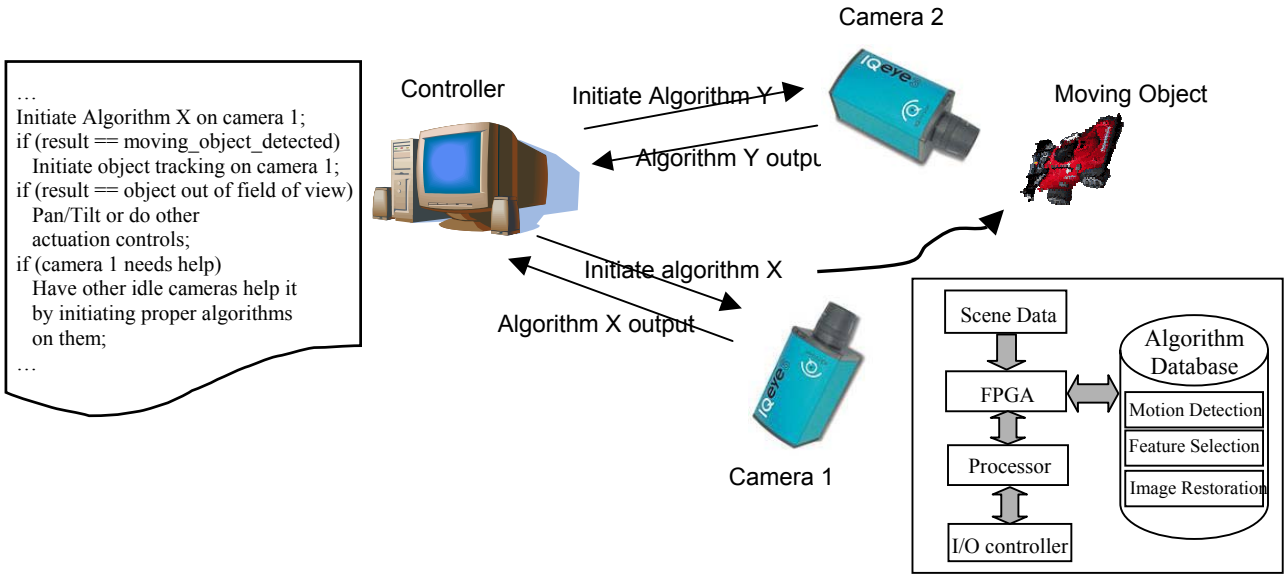


Figure 2. An overview of the tracking system architecture: Each camera has a set of the required configurations available. The controller communicates with the cameras via an implemented message-passing scheme and can initiate the proper algorithm on each camera, organizing the collaboration among cameras.

consists of multiple IQeye3 cameras [4]. The cameras are connected to the local area network and communicate with each other or the control unit in order to collaborate and share their information. An outline of the architecture of one of IQeye3 cameras is demonstrated in figure 1.b. A Xilinx VirtexE [13] FPGA and a general-purpose processor (IBM PowerPC) are embedded in all of the cameras. The FPGA and the processor can both be used to implement the comprising blocks of an application. In this paper, we present the crucial role of the embedded reconfigurable devices (i.e. Xilinx FPGA) in our system. We exploit these devices to achieve both real-time performance and dynamic adaptability of the system to the external events.

We proceed to describe our system framework and its application in the next section. In section 3, 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. Experimental results including algorithms implementation and their performance for some scenes, has been presented in section 4. Finally, section 5 outlines the conclusions and future directions of this paper.

2. Reconfigurable Tracking System

In this section, we present the reconfigurable tracking system that has been built as part of this work. First, we present the framework of our system along with its application. In next section, we discuss the algorithms that are needed for implementing the system application. We describe how these algorithms have to be tuned based on the changes in the scene and hence, highlight the "*reconfigurability*" feature of our tracking system.

2.1 System Framework

The framework for our system is comprised of several components including: IQeye3 cameras provided by IQinVision [4], pan-tilt units to enable the actuation of the cameras, a main controller residing on a PC, and a network for collaboration and data communication.

An IQeye3 camera, as a "smart" vision sensor, with embedded computation resources [1, 2, 3], 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 (Figure 1.b). 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.

In addition, 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.

Each pan-tilt actuation unit can be controlled using simple commands that specify the pan angle, pan speed, pan acceleration, tilt angle, tilt speed, and tilt acceleration. 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 port.

Figure 2 demonstrates our system with two cameras and the main controller. The main supervisory controller resides on a PC 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 (Figure 2)). This is performed by sending a message to the designated camera.

2.2 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 [5, 6, 7] can effectively track the motions.

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, various 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. Camera 1 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 computation 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 will violate 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 environment changes. 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 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 variations in the scene. 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.

3. 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 am 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.

3.1 Feature selection

KLT tracking scheme [5, 6, 7] is carried out in two stages. In the first stage, called feature selection, proper points in the images are selected. These 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[1] and feature tracking is currently performed on the PowerPC embedded in the IQeye3 cameras.

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 selected as good feature points due to the good trackability. In summary, the algorithm works as follows [8]:

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. For each pixel:

a) Sum the gradients in the surrounding window in order to compute the Z matrix, where

$$Z = \iint_W \begin{bmatrix} g_x^2 & g_x g_y \\ g_x g_y & g_y^2 \end{bmatrix} dx$$

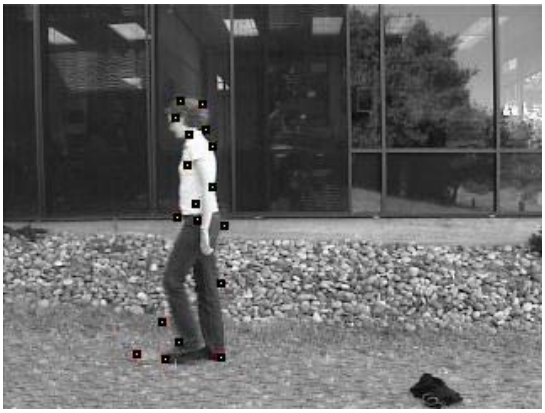
b) Compute λ_1 and λ_2, the eigenvalues of the Z matrix. Let $\lambda_1 = \min(\lambda_1, \lambda_2)$. λ_1 represents the trackability of the pixel.

c) Given λ as the threshold value, If $\lambda_1 > \lambda$ then declare this pixel as a feature.

Figure 3 demonstrates the output of the feature selection on a selected region of sample images.

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.

[1] Our implementation is based on [12]



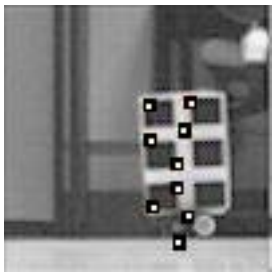


Figure 3. Sample outputs of feature selection algorithm executed on the camera. Features are denoted by black squares with white centers.

The feature tracking stage of the KLT tracking method, strives to locate the selected features, in the next frame. 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.

3.2 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. The following degradation model holds in a large number of applications [11]:

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

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).
Common Implementations of image restoration iterate over the entire image and perform local changes. Though effective in many applications, this implementation is not practical in our constrained platform. We have implemented another version of image restoration that does not need to iterate over the entire image and hence, decreases the memory requirement of the system. This has been thoroughly discussed in the next subsection.

4. Vision Algorithms Implementations

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

4.1 Platform Constraints

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. FPGA then performs several operations on the stream such as image correction, windowing and down sampling. Then, 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. For example, a sample application running on the processor embedded in the camera compresses the image data into jpeg format and exports the jpeg file through Ethernet.
Within this environment and platform, image-processing applications sitting on the FPGA need to meet some 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 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 off-chip memory unit will impose additional latency, which might not be tolerable for some designs.
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.

4.2 Implementations

4.2.1 Feature selection

Feature selection algorithm has been implemented on the same platform in a previous work [8, 12]. 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. 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, because threshold has been implemented as a constant, which should be specified at design time.
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 number of selected features. 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 [12] and have modified the original design such that the threshold value can be controlled by a program running on camera PowerPC at runtime. This has enabled dynamic adaptation feature of the feature selection. 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.

4.2.2 Image restoration

Image restoration has a variety of implementation and iterative method is a widely used one. The purpose is to estimate the original image given the degraded image. The original method performs operations on an entire image iteratively. The iteration is stopped when the restored image converges with insignificant residual ε [14].

Our constrained platform (refer to section 4.1) 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.

We have made several modifications to adapt the original method to our environment. 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 (Figure 4) does not have any information about pixels out of the restoration window. However, this enables processing of image stream using a small-sized storage.

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. Note that memory requirement is indirectly corresponding to performance. Figure 4 shows the idea of our implementation and the amount of storage required for 3x3 windows.

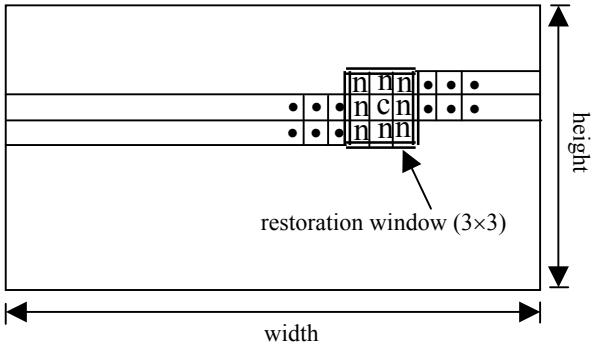


Figure 4. Image frame and restoration window. c is the center pixel on which the computation is carried and n's represent the neighbor pixels. In this example, two rows of the frame need to be stored on-chip.

5. 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 will address the issue of system adaptation to the environment variations in this section. Then, we present the results of our approach to support our approach.

5.1 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 (this can be determined using the value of the threshold in feature selection for selecting a certain number of features), then the FPGA will be reconfigured to enable the image restoration before feature selection. 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.

5.2 Experimental Results

In the following sets of experiments, we examine the effect of our proposed techniques. The first two 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 any outdoor places where the natural lighting changes throughout the time. Another example is many indoor scenes under various lighting conditions. For the experiment, we varied the lighting condition in the laboratory and observed the results of the feature selection application.

Figures 5, 6 and 7 show the result of feature selection with fixed threshold (FS-FIX). Figure 8, 9 and 10 show the results of feature selection with automatically adjusted threshold (FS-AUTO).

Figures 5 and 8 are under the same lighting so that both FS-FIX and FS-AUTO select about 150 features (with 10% tolerance). Figure 6 and 9 are under the same lighting, which is brighter than in the previous setting. Extra brightness causes edges and corners to have greater intensity difference from their adjacent pixels, therefore they are chosen as features. Figure 6 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 9. It selects features at almost same locations as in Figure 8 even after the significant change in brightness.

Figures 7 and 10 are under the same dark lighting. The object is observable by eyes, but FS-FIX is unable to find any features. FS-AUTO decreases the threshold value from 512 to 160 and

Figure 5. 152 features are selected by fixed threshold (FS-FIX) under default lighting conditions.

Figure 8. 148 features are selected by automatic threshold adjustment (FS-AUTO) under default lighting conditions.

Figure 6. 1150 features are selected by fixed threshold (FS-FIX) under bright lighting.

Figure 9. 150 features are selected by automatic threshold adjustment (FS-AUTO) under bright lighting.

Figure 7. No feature is selected by fixed threshold (FS-FIX) under dark lighting.

Figure 10. 156 features are selected by automatic threshold adjustment (FS-AUTO) under dark lighting.

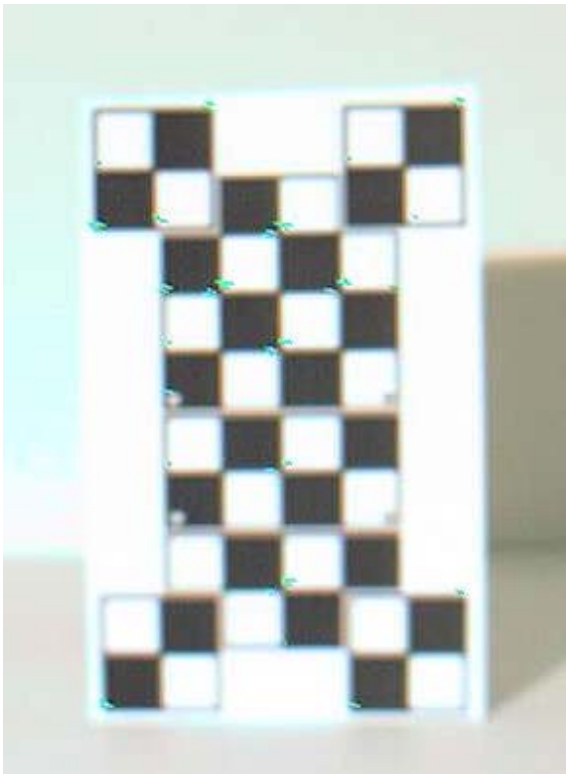


Figure 11. Selected features using automatically threshold adjustment (FS-AUTO) without image restoration

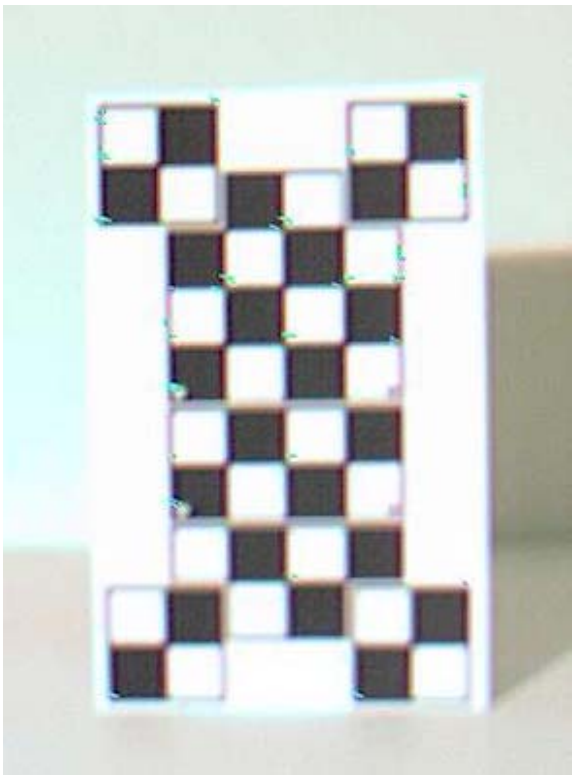


Figure 12. Image restoration sharpens the edges and corners. Therefore, FS-AUTO selects better (selected with larger threshold) features.

finds 156 features. Locations of features are almost same as Figure 8 and 9.

Figures 11 and 12 demonstrate the effect of image restoration on feature selection results. While, FS-AUTO is capable of selecting the required number of features, the threshold value at the presence of image restoration is larger. Hence, the image 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 and can create fake features in the image. Therefore, the system should be reconfigured to enable or disable image restoration based on the requirements.

Table 1 summarizes the number of selected features and the utilized threshold value for selecting those features for pictures 5 to 12. 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 11, sharpens its edges and corners (see Figure 12) 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 1.

Figure Number	Threshold	Feature count
5	512 (Fixed)	152
6	512 (Fixed)	1150
7	512 (Fixed)	0
8	465	148
9	1552	150
10	160	156
11	1279	146
12	2083	148

Table 1. Feature selection threshold value and feature count for figures presented in experimental result section.

6. Conclusions 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 system, enhancing the collaboration schemes and applying the system reconfiguration idea to other applications or application domains.

7. 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] 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.

[4] IQinVision Online Documentations, IQinVision Inc., http://www.iqinvision.com.

[5] 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

[6] C. Tomasi, T. Kanade, “Detection and Tracking of Point Features”, *Carnegie Mellon University Technical Report CMU-CS-91-132*, April 1991.

[7] J. Shi, C. Tomasi, “Good Features to Track”, *IEEE Conference on Computer Vision and Pattern Recognition*, pages 593-600, 1994

[8] 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.

[9] P. Athanas and 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.

[10] X. Feng, P. Perona, “Real Time Motion Detection System and Scene Segmentation”, *CDS TR CDS98-004*, Caltech, 1998

[11] 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.

[12] M. Maire, “Design and Implementation of a Realtime Visual Feature Tracking System on a Programmable Video Camera”, Technical Report, California Institute of Technology, 2002.

[13] Xilinx Online Documentations, Xilinx Inc., http://www.xilinx.com.