High Performance Color Image Processing in Multicore CPU using MFC Multithreading

Anandhanarayanan Kamalakannan
Central Electronics Engineering Research Institute
Chennai Centre, CSIR Madras Complex, Taramani
Chennai – 600113, Tamil Nadu, India

Govindaraj Rajamanickam
Central Electronics Engineering Research Institute
Chennai Centre, CSIR Madras Complex, Taramani
Chennai – 600113, Tamil Nadu, India

***Abstract*—Image processing is an engineering field where stored image data is readily available for parallel processing. Basically data processing algorithms developed in sequential approach are not capable of harnessing the computing power of individual cores present in a single-chip multicore processor. To utilize the multicore processor efficiently on windows platform for color image processing applications, a lock-free multithreading approach was developed using Visual C++ with Microsoft Foundation Class (MFC) support. This approach distributes the image data processing task on multicore Central Processing Unit (CPU) without using parallel programming framework like Open Multi-Processing (OpenMP) and reduces the algorithm execution time. In image processing, each pixel is processed using same set of high-level instruction which is time consuming. Therefore to increase the processing speed of the algorithm in a multicore CPU, the entire image data is partitioned into equal blocks and copy of the algorithm is applied on each block using separate worker thread. In this paper, multithreaded color image processing algorithms namely contrast enhancement using fuzzy technique and edge detection were implemented. Both the algorithms were tested on an Intel Core i5 Quad-core processor for ten different images of varying pixel size and their performance results are presented. A maximum of 71% computing performance improvement and speedup of about 3.4 times over sequential approach was obtained for large-size images using four thread model.**

Keywords*—Color image; fuzzy contrast intensification; edge detection; lock-free multithreading; MFC thread; block-data; multicore programming

I. Introduction

Machine vision systems used in various industrial applications are capable of capturing high resolution images and demands time efficient parallel data processing algorithms in real-time environment. To reduce the processing time of the algorithm on these images, parallel computing in multicore architecture is a well known approach [1]. Different parallel programming libraries such as OpenMP and Message Passing Interface (MPI) are widely applied in the development of parallel image processing algorithms. The authors N.E.A. Khalid et al [2] have implemented parallel multicore sobel edge detection algorithm using MPI and observed that parallel processing performs better than sequential processing in terms of execution speed. Chen Lin et al [3] have proposed a parallel method to perform medical image registration using OpenMP and concluded that multithreading approach saves nearly half of the computing time. Alda Kika and Silvana Greca illustrated the development of multithreaded algorithms for contrast, brightness and steganography applications using Java package and tested their performance on different single-core and multicore processors [4]. The authors concluded that the performance of the complex image processing algorithm on multicore CPU can be improved using multithreaded programming.

In our work, we studied the development of multithreaded C++ algorithms for processing low and high resolution color images on a multicore CPU without using parallel programming library and any other additional hardware. To ensure fine grain (data level) parallelism [5] and computation load balance of the algorithm in a multicore CPU, a lock free multithreaded block-data parallel approach is proposed. In this approach, the image data is shared equally among worker threads and each one manipulates its portion of data.

In VC++ programming, MFC library provides powerful threading Application Programming Interfaces (APIs) [6] for developing concurrent or multithreaded windows based software programs. Multithreaded color image processing algorithms namely contrast enhancement using fuzzy technique and edge detection were developed in Intel Pentium dual-core personal computer and tested on Intel Core i5 CPU. The algorithms were applied on ten selected color image samples of varying size and their execution results are presented. The performance results show that both the algorithms in four thread model attained a speedup of about 3.4 times compared with the sequential approach and saves nearly 71% of algorithm execution time.

The paper is arranged as follows: In section II, MFC multithreading and its application in high performance image processing is described. Section III explains the materials, methods and the color image processing techniques followed in this paper. The performance results of the thread model based parallel algorithms are discussed in section IV. The conclusion is given section V.

II. Multithreaded Image Processing using MFC

MFC is a Microsoft's C++ class library for windows programming. It distinguishes two types of threads namely user interface thread and worker thread [7]. The main use of worker thread is to perform background computation work and it is created by defining the task it should perform. This is done by the declaration of thread function according to the MFC definition. The call function AfxBeginThread() launches

the worker thread [8] and it accepts parameters, which includes thread function name, input to the thread, thread priority and few other required parameters.

In block-data parallel processing, image region is identified as several blocks of data. The source image data is partitioned vertically or horizontally into multiple large blocks with equal size [9,10]. In our thread model based parallel approach, each thread exclusively performs image processing task on individual image data block as shown in the concurrency model Fig.1. To maintain load balance within threads, it is good to consider the number of image blocks equal to number of worker threads [11]. Since the image data is stored and accessed through global variables no message passing or explicit data access control is required between threads. This makes thread definition simple without data locking mechanism [12].

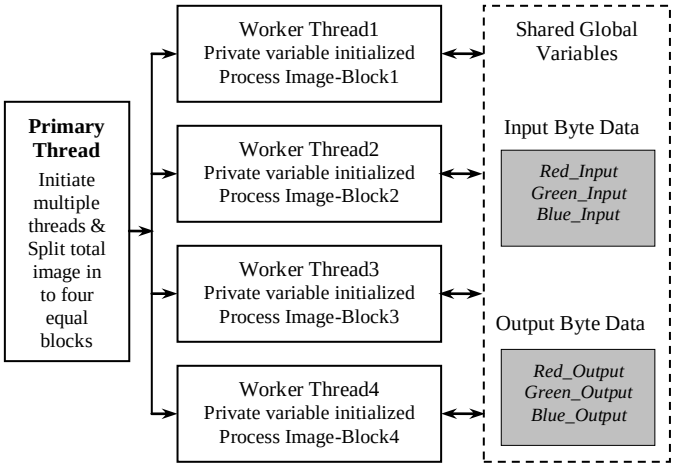

Fig. 1. Concurrency model of block-data parallel algorithm using 4 threads

In this lock free multithreaded approach, threads are free to read and process their portion of image data in a parallel manner, which efficiently reduces the data access time as well as the overall computation time of image processing algorithm. Thus the performance of multithreaded algorithm on a single- chip multicore processor can be fine tuned using shared image data variables [13]. In the case of color image processing algorithm, three input and three output global variables were assigned to each color component (viz. red, green and blue) to enable image reading and processed data writing concurrently using multiple worker threads.

According to the worker thread priority, the operating system schedules each thread to an individual processing unit in a multicore CPU. Due to this scheduling mechanism, all threads do not finish at the same time, so in order to handle this thread completion task, event object is derived from CEvent MFC class. When the thread completes its processing task, the event object is triggered. Using WaitForSingleObject API, event object trigger is noted and worker thread completion is indicated to the primary main thread [6,14,15]. As soon as all the threads complete their processing task, the results are cached and made available in the shared global variable. The synchronization between the main thread and different worker threads was established using event object as shown in Fig.2.

In MFC multithreading, two threads cannot manipulate the same object because MFC objects are thread-safe only at the class level [16]. Hence each thread requires separate objects of the same data structure to operate in a thread-safe manner. To ensure thread safety in the algorithm, each copy of thread parameter data structure is passed as input argument to the corresponding worker thread function. Each thread function uses call by reference method to access the global variables. When the thread calls a image processing function, the private variables declared within the function takes care of storing, processing the intermediate data and also ensures the algorithm execution in parallel manner.

```
//data structure for thread-parameter
typedef struct ThreadParameter
{       int height1, height2, width;
ImageProcessing Object;
} Parameter;

//Global declaration of event object
CEvent threadone, threadtwo;

//Worker thread creation
MainThread()
{
//assign worker thread parameter for image-block1
        Parameter *First=new Parameter;
//assign worker thread parameter for image-block2
        Parameter *Second=new Parameter;
AfxBeginThread(ThreadProcA,First,THREAD_PRIORITY_HIGHEST);
AfxBeginThread(ThreadProcB,Second,THREAD_PRIORITY_HIGHEST);
 ::WaitForSingleObject(threadone,INFINITE);
 ::WaitForSingleObject(threadtwo,INFINITE);
}

//Function Definition_FirstThread
UINT ThreadProcA(LPVOID param)
{       Parameter * ptr=(Parameter *)param;
        ptr->Object.ProcessImage(image-block1);
        threadone.SetEvent();
        return 0;
}

//Function Definition_SecondThread
UINT ThreadProcB(LPVOID waram)
{       Parameter * ptr= (Parameter *)waram;
        ptr->Object.ProcessImage(image-block2);
        threadtwo.SetEvent();
        return 0;
}
```

Fig. 2. Code structure for two thread approach

III. Materials and Methods

A. Sample Images

A total of ten color images with different pixel size were used to evaluate the algorithm performance. All these images were randomly chosen from free online collection of natural scenes and photos. The pixel size of the images varies from 940x474 to 2880x1800. They are labeled as Image1, Image2….Image10.

B. Hardware and Software

The entire coding for developing the multithreaded application software was carried out in Intel Pentium dual-core processor @ 2.8GHz on Windows XP operating system pre-loaded with Microsoft Visual Studio version 6. Using MFC library, multithreaded image processing software for contrast enhancement using fuzzy technique and edge detection has been developed in VC++. Separate menu buttons are provided in the software to load image and execute the algorithms. The developed algorithms were tested using the color image samples on an Intel Core i5-760 @ 2.80 GHz Quad-core CPU on 32 bit Windows 7 operating system with 4GB RAM.

C. Color Image Processing Algorithms

a) Contrast enhancement using fuzzy technique

Image enhancement is a preprocessing technique usually employed to improve the brightness and contrast of the images. In color image enhancement, red, green and blue channels were processed separately and added together to produce composite color value. But this approach does not maintain the color balance in the image. To avoid this change in color information, YIQ color space was chosen, where Y represents the luminance information; I and Q together represent the chrominance information. This color space exploits certain characteristics of human-eye color response and improves the appearance of the color image in terms of human brightness perception. In this technique, the contrast enhancement using fuzzy intensification operator was applied only on luminance component; hence color information of the original image is preserved [17].

Steps involved in image contrast enhancement:

1) Convert RGB image in to YIQ color space [18].

2) Perform fuzzification [19] on luminance component 'Y_{ij}' using the following expression.

$$\mu_{ij} = f(Y_{ij}) = \left[1 + \frac{Y_{max} - Y_{ij}}{f_d}\right]^{-f_e} \quad (1)$$

Where f_e and f_d denote the exponential & the denominational fuzzifier, respectively and μ_{ij} is called the fuzzy property plane of the image. Value of the f_e can be set as 1 or 2. Value of f_d is determined using the cross-over value with respect to fuzziness value 0.5. Y_{max} represents the maximum luminance value.

3) Apply fuzzy intensification operator $T\left(\mu_{ij}'\right)$ on luminance component for contrast enhancement [20].

$$T\left(\mu_{ij}'\right) = \begin{cases} 2\left[\mu_{ij}\right]^2, & 0 \le \mu_{ij} \le 0.5 \\ 1 - 2\left[1 - \mu_{ij}\right]^2, & 0.5 \le \mu_{ij} \le 1 \end{cases} \quad (2)$$

4) Enhanced luminance component 'Y_{ij}' is obtained using defuzzification defined as follows.

$$Y_{ij} = Y_{max} - f_d \left[\frac{1 - \left(\mu_{ij}'\right)^{\frac{1}{f_e}}}{\left(\mu_{ij}'\right)^{\frac{1}{f_e}}}\right] \quad (3)$$

5) Convert YIQ color space to RGB image.

6) Contrast enhanced color image is obtained at the end.

b) Edge detection

Edges in color images can be obtained by applying gray scale edge detection method to each of the RGB bands separately and then results were summed to produce composite value [21]. Further thresholding was performed to get fine binary edges and a set of four Robinson compass masks used in this method are given below.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}$$

(a) (b) (c) (d)

D. Multithreaded Block-data Parallel Approach Steps

1) Image block-data decomposition; the main thread splits the image data in to several blocks of equal size to maintain load balance [22].

2) Multiple worker threads are created in the main thread and size parameter of each block is passed as input to the worker thread.

3) Created worker threads are initiated with high priority level to avoid delay due to operating system scheduling.

4) Each worker thread applies its copy of sequential image processing algorithm on a particular image data portion.

5) Each worker thread uses their private copy of data structure for execution.

6) Processed image data are stored in output variable.

7) As shown in Fig.3, main thread exits only when all worker threads complete their assigned task.

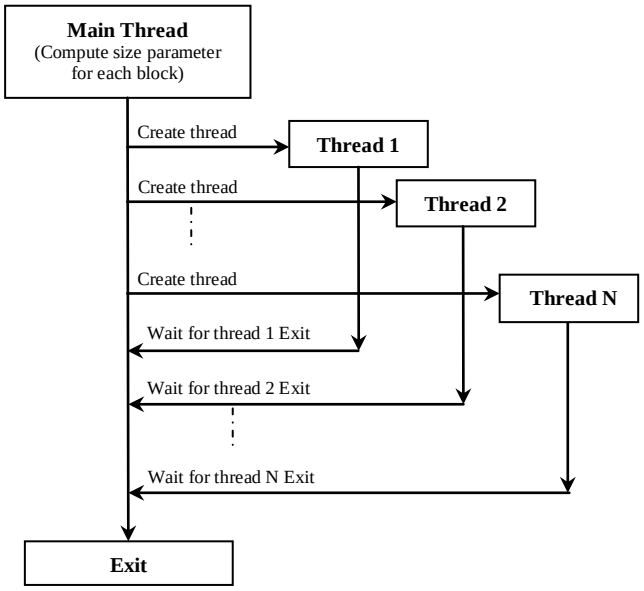

Fig. 3. Thread sequence diagram

IV. Results and Discussion

To measure the execution time of the parallel algorithms while processing the given image data in different threads, the

VC++ clock function was used. Using the execution time, the speedup and performance improvement (P.I) parameters of the algorithms were calculated. The speedup parameter measures how much a parallel algorithm is faster than a corresponding sequential approach [2]. The P.I predicts the relative improvement due to parallel implementation over the sequential approach. The equations for computing the two parameter values are given below.

$$\text{Speedup} = \frac{\text{Sequential\_Approach(ms)}}{\text{Parallel\_Approach(ms)}} \quad (4)$$

$$\text{P.I}\ (\%) = \frac{\text{Sequential\_Approach(ms)} - \text{Parallel\_Approach(ms)}}{\text{Sequential\_Approach(ms)}} \times 100 \quad (5)$$

A. Results of Multithreaded Contrast Enhancement Algorithm

The developed contrast enhancement algorithm was applied on all the ten sample images and the test results were evaluated. The algorithm execution time for each image in sequential and multithreaded approach (for 2, 4 and 8 threads) was recorded in a data file. To determine the average execution time in both the approaches, the algorithm was executed five times successively on each image and the mean time was calculated. The speedup and performance improvement between sequential and four thread approaches was computed using Eq.4 and Eq.5 for images of different pixel size. The algorithm results viz., average execution time, speedup and performance improvement are shown in Table I.

TABLE I. 'Performance Results of Color Contrast Enhancement Algorithm

Image name & pixel size	**Average execution time in milliseconds (ms)**				**Four thread approach**	
	Sequential Approach	*Two Thread*	*Four Thread*	*Eight Thread*	*SpeedUp*	*P.I (%)*
Image1(940 x474)	172	93	68	62	2.53	60.47
Image2(1024 x 728)	265	145	94	93	2.82	64.53
Image3(1280 x 1024)	437	234	140	140	3.12	67.96
Image4(1600 x 1200)	624	328	196	198	3.18	68.59
Image5(1920 x 1080)	675	353	209	212	3.23	69.04
Image6(1920 x 1200)	749	390	231	237	3.24	69.16
Image7(2048 x 1536)	1019	530	312	312	3.27	69.38
Image8(2288 x 1712)	1248	645	375	379	3.33	69.95
Image9(2560 x 1920)	1571	796	458	458	3.43	70.85
Image10(2880 x 1800)	1659	843	483	499	3.43	70.89

To find the maximum possible number of threads needed to speed up the algorithm execution in the Intel Core i5 processor, eight thread approach was also attempted and the execution time results are included in Table I. It is found from the Table I, the execution time of four and eight threads are nearly same which infers that for a quad-core processor, minimum of four MFC thread is enough to achieve optimum execution time.

As seen from the tabulated results, the average execution time of the algorithm decreases with the number of threads, whereas the speedup and performance improvement goes up with increase in image size. In the four thread implementation, the speedup parameter varies from 2.53 to 3.43 times and the performance improvement variation is found to be between 60.47% and 70.89%.

The input image and processed color image outputs of contrast enhancement algorithm are shown in Fig.4a, Fig.4b & Fig.4c. The processed results of sequential and multithreaded approach are looking similar.

Fig. 4. a. Input photograph image

Fig. 4. b. Contrast enhancement in sequential approach

Fig. 4. c. Contrast enhancement in four threads

Table II illustrates the execution time of individual threads measured for contrast enhancement algorithm executed five times successively on a single image. It shows that each thread takes nearly same amount of CPU time to compute the data processing task and the load balance within multiple threads is well maintained. Therefore change in execution time is mainly dependent on varying image size.

TABLE II. Individual Thread Execution Time for Four Thread Contrast Enhancement Algorithm (Image Size: 1920x1200)

Running Iteration No.	**Execution time in milliseconds (ms)**				
	Thread1	*Thread2*	*Thread3*	*Thread4*	*Algorithm*
1	218	234	234	234	234
2	218	218	234	234	234
3	219	234	234	234	234
4	218	218	218	234	234
5	218	218	218	218	218

B. Results of Multithreaded Edge Detection Algorithm

A similar approach as followed for the contrast enhancement algorithm was applied for the edge detection algorithm on all the ten color images and the results are presented in Table III. In four thread approach, the speedup varies from 2.82 to 3.44 times and the performance improvement achieved is between 64.50% and 70.94%.

TABLE III. Performance Results of Color Edge Detection Algorithm

Image name & pixel size	Average execution time in milliseconds (ms)				Four thread approach	
	Sequential Approach	*Two Thread*	*Four Thread*	*Eight Thread*	*SpeedUp*	*P.I(%)*
Image1(940 x474)	307	167	109	104	2.82	64.50
Image2(1024 x 728)	520	265	171	172	3.04	67.12
Image3(1280 x 1024)	837	431	265	255	3.16	68.34
Image4(1600 x 1200)	1229	634	369	369	3.33	69.98
Image5(1920 x 1080)	1304	676	390	405	3.34	70.09
Image6(1920 x 1200)	1466	765	437	442	3.35	70.19
Image7(2048 x 1536)	1992	1024	588	588	3.39	70.48
Image8(2288 x 1712)	2475	1269	728	733	3.40	70.59
Image9(2560 x 1920)	3100	1571	905	899	3.43	70.81
Image10(2880 x 1800)	3276	1648	952	952	3.44	70.94

The input image and processed color image outputs of edge detection algorithm are shown in Fig.5a, Fig.5b & Fig.5c. The processed image outputs of the algorithm are found to be similar.

Fig. 5. a. Input MatLab demo image

Fig. 5. b. Edge detection in sequential approach

Fig. 5. c. Edge detection in four threads

Thus the two multithreaded color image processing algorithms with different complexity levels were tested in Intel Core i5 processor and found that four thread approach utilized the quad-core CPU efficiently on Windows 7 platform.

V. CONCLUSION

This work was carried out to explore the parallel processing ability of the multicore CPU in processing high resolution images using MFC multithreading. In this paper, a lock-free multithreaded block-data parallel approach based color image processing algorithms for fuzzy contrast enhancement and edge detection were developed using VC++ on windows platform without using any parallel programming library. The purpose of this implementation is to improve the performance and reduce the execution time of the image processing algorithms on multicore processor by partitioning the given image into equal blocks and processing each block of data in a parallel manner. In four thread approach, the algorithm speed is found to be about 3.4 times faster than the sequential approach. With regard to performance improvement, the thread model saves nearly 71% computation time compared to sequential implementation. No performance improvement and speedup is noted in processing nearly same size images of marginal difference in pixel size. The performance results indicate that multithreaded image processing algorithms efficiently utilize the computing capability of multicore CPU like Intel Corei5 processor. Hence the developed multicore programming approach using MFC thread can be applied to improve the performance of various color image processing algorithms.

REFERENCES

[1] P.N.Happ, R.S.Ferreia, C.Bentes, G.A.O.P.Costa and R.Q.Feitosa, "Multiresolution Segmentation: A parallel approach for high resolution image segmentation in multicore architectures", International Conference on Geographic Object-Based Image Analysis, ISPRS Vol.XXXVIII-4/C7, June-July 2010.

[2] N.E.A.Khalid, S.A.Ahmad, N.M.Noor, A.F.A.Fadzil and M.N.Taib, "Parallel approach of sobel edge detector on multicore platform", International Journal of Computers and Communications, Vol.5 Issue.4, 2011, pp.236-244.

[3] Chen Lin, Li Jian, Zhou Jun and Jiang Murong, "Multithreading method to perform the parallel image registration", IEEE Xplore, International Conference on Computational Intelligence and Software Engineering, DOI:10.1109/CISE.2009.5366052, Dec. 2009.

[4] Alda Kika and Silvana Greca, "Multithreading image processing in single-core and multi-core CPU using Java", International Journal of Advanced Computer Science and Applications, Vol.4, No.9, 2013, pp.165-169.

[5] Luis Moura E Silva and Rajkumar Buyya, "Chapter1: Parallel programming models and paradigms", High Performance Cluster Computing: Programming and Applications, Vol.2, Prentice Hall PTR, 1999, pp.4-28.

[6] S.Akhter and J.Roberts, "Chapter 5: Threading APIs", Multicore Programming: Increasing performance through Software Multi-threading, Intel Press, 2006, pp.75-133.

[7] J. Prosise, "Chapter 17: Threads and thread synchronization", In Programming Windows with MFC, 2nd Edition, Microsoft Press, 1999, pp.985-1000.

[8] Stanford Taylor Jones and Chi Ngoc Thai, "Multithreaded Design of Spectral Imaging Software", ASAE Meeting Presentation, Paper No.053010, July 2005.

[9] Winser E.Alexander, Douglas S.Reeves and Clay S.Gloster, "Parallel image processing with the block data parallel architecture", IEEE Xplore, Proceedings of the IEEE, DOI:10.1109/5.503297, Vol.84, No.7, July 1996, pp.947-968.

[10] A.Fakhri A.Nasir, M.Nordin A.Rahman and A.Rasid Mamat, "A study of image processing in agriculture application under high performance computing environment", International Journal of Computer Science and Telecommunications, Volume 3, Issue 8, 2012, pp.16-24.

[11] Sanjay Saxena, Neeraj Sharma and Shiru Sharma, " Image processing tasks using parallel computing in multicore architecture and its applications in medical imaging", International Journal of Advanced Research in Computer and Communication Engineering, Volume 2, Issue 4, 2013, pp.1896-1900.

[12] Jonathan R.Engdahl and Dukki Chung, "Lock-free data structure for multi-core processors", International Conference on Control, Automation and Systems, October 2010, pp.984-989.

[13] Devrim Akgun, "Performance evaluations for parallel image filter on multi-core computer using Java threads", International Journal of Computer Applications, Vol.74, No.11, July 2013, pp.13-19.

[14] S.S.Ilic, A.C.Zoric, P. Spalevic and Lj. Lazic, "Multithreaded application for real-time visualization of ECG signal waveforms and their spectrums", Intl. Journal of Computer, Communication & Control, 8(4), 2013, pp.548-559.

[15] Faran Mahmood, "Parallel Implementation of Imaging Filters on Multi-Core Processors for Win32 platform", Proceedings of the 4th International Conference on Open-Source Systems and Technologies, December 2010.

[16] Multithreading: Programming Tips – "Accessing objects from multiple threads", Available from http://msdn.microsoft. com/en-us/library/h14y172e.aspx.

[17] Zhuqing Jiao and Baoguo Xu, "An image enhancement approach using Retinex and YIQ", IEEE Xplore, Inter-national Conference on Information Technology and Computer Science, DOI:10.1109/ITCS.2009.104, July 2009, pp.476-479.

[18] Gwanggil Jeon, "Image enhancement in YIQ space", Proceeding of First International Conference on Advanced Computer and Information Technology, ASTL vol.22, 2013, pp.109-112.

[19] Sankar K.Pal and Robert A.King, "Image enhancement using smoothing with fuzzy sets", IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-11, No.7, 1981, pp.494-501.

[20] Peng Dong-liang and Xue An-ke, "Degraded image enhancement with applications in robot vision", IEEE Xplore, International Conference on Systems, Man and Cybernetics, DOI:10.1109/ICSMC.2005.1571414, Vol.2, October 2005, pp.1837-1842.

[21] Scott E.Umbaugh, "Chapter 4: Segmentation and Edge/Line Detection", Computer Imaging- Digital image analysis and processing, CRC Press, 2005, pp.184-188.

[22] Young-Jip Kim and Byung-Kook Kim, "Load balancing algorithm of parallel vision processing system for real –time navigation", IEEE Xplore, International Conference on Intelligent Robots and Systems, DOI:10.1109/IROS.2000. 895242, Vol.3, Oct-Nov 2000, pp.1860-1865.