Real-Time Background Compensation for PTZ Cameras Using GPU Accelerated and Range-Limited Genetic Algorithm Search

Thuy Tuong Nguyen and Jae Wook Jeon*

School of Information and Communication Engineering,
Sungkyunkwan University, Suwon, Korea
ntthuy@skku.edu, jwjeon@yurim.skku.ac.kr

Abstract. We propose a range-limited Genetic Algorithm (GA) search with an accelerated Graphics Processing Unit (GPU) based implementation for background compensation where pan-tilt-zoom (PTZ) cameras are used. Our method contains GA with search ranges restricted using histogram matching and GPU implementation of the range-limited GA. First, based on histogram matching, estimation of approximate scale (camera zoom) and translation (camera pan and tilt) parameters is used to restrict the ranges for the later GA search. Next, the GA is applied to find an optimal solution. Experimental comparisons of the proposed method to existing methods show that our work has advantages: robust to critical situations due to using GA, and fast processing.

Keywords: Background compensation, histogram matching, GA, GPU.

1 Introduction

Intelligent visual surveillance, using computer vision techniques, is increasingly important over recent years. PTZ cameras are mainly used in this research area for object detection and tracking. The challenge of using this type of camera is to eliminate the dynamic background caused by camera motion. Hence, motion based tracking with a PTZ camera encounters difficulties such as identifying features of object motion, compensating background motion, and tracking mechanism. Although compensating background requires less computational cost and memory storage compared to mosaicing background and optical flow clustering [1], it typically yields a non-optimal solution.

Alternative approaches have been proposed to compensate background. In [2], specialized hardware is used to measure pan, tilt and zoom parameters. Relationship between pixels representing the same 3-D point in frames is estimated

* This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2011-(C1090-1121-0008)), and by Priority Research Centers Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2011-0018397).

to eliminate background motion. Background motion is represented by an affine transformation in [3], and affine motion parameters are estimated using least median of squares. In this method, a number of feature points in the current image and their corresponding points in the previous image are picked out, and the pixels from the regions of moving objects are considered as outliers of the affine estimator. However, it is sensitive in selecting feature points due to motion blur. In [4], M-estimator like techniques in a multiresolution framework are used as the parametric motion model estimation. Multi-resolution Hough transform is used to estimate affine parameters in [5]. Results in this paper show that the main advantage resides with motion estimation in presence of motion blur. Overcoming disadvantanges of the previous methods, [1] presents the 1-D feature matching and outlier rejection method. This method is robust to motion blur and moving object proportion.

GA have been widely applied in estimating global motion. It is a stochastic search technique based on the principles of natural selection and genetics to find the approximate solutions of optimization and search problems. A GA in the continuous space to estimate global motion is proposed in [6]. A multiresolution GA is proposed in [7] to solve the imagerelated optimization problems image segmentation, stereo vision and motion estimation. Background motion can be effectively determined based on the proposed motion estimation method. For a problem, GA can yield a near-optimal solution when it is well-modeled and its related parameters are appropriately configured. However, there is a trade-off between computational cost and accuracy: higher numbers of chromosomes and generations increase processing time. Therefore, the existing GA based background compensation methods are not appropriate for real-time systems when high accuracy is being considered.

Range-limited GA search implemented using GPU is presented in this paper to achieve high accuracy and low processing time of background compensation for PTZ cameras. Our method overcomes all drawbacks of existing methods: motion blur, a large proportion of moving objects, and difficulty in selecting feature points. The proposed GA model contains two components: 1) estimation of approximate scale (camera zoom) and translation (camera pan and tilt) parameters is used to limit ranges for the later GA search, and 2) GA search is used to reach the optimal solution. The estimation model employs projection histograms to quickly determine the scale and translation parameters. Furthermore, processing time is significantly reduced by the GPU implementation.

Our method of range-limited GA search for background compensation is presented in Section 2. GPU implementation is in Section 3. Section 4 presents experimental results. The paper is drawn to a conclusion in Section 5.

2 Range-Limited GA Search

2.1 Background Motion Model and GA Search

GA typically contains a population of suitably encoded solutions to the problem and contains an evaluation function. It differs from other traditional optimization techniques, because it involves a search from a population of solutions, but not from a single point. Due to its characteristics, GA is perfectly suited to solving the optimization problem represented by background compensation.

Based on [1], we initially assume that the relationship between successive frames captured by a PTZ camera can be approximated by a transformation of four parameters (two directional zooms, pan and tilt). Hence, a chromosome in the GA based background compensation problem can be identified with the array formed by motion model parameters. In the case of an affine motion model,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \mathbf{A} \begin{pmatrix} x \\ y \end{pmatrix} + \mathbf{b}, \tag{1}$$

the chromosome is thus formed by the combination of four, in this paper, motion parameters. This means $\mathbf{A}$ and $\mathbf{b}$ are 2×2 and 2×1 matrices, respectively:

$$\mathbf{A} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}; \mathbf{b} = \begin{pmatrix} t_x \\ t_y \end{pmatrix}; s_x, s_y > 0. \tag{2}$$

The chromosome representing four parameters, s_x, s_y, t_x and t_y, are floating point vectors. In GA, a random initial population with a given number of individuals is first generated. If the explicit information of the system is provided in advance, then such knowledge can be included in the initial population (e.g., we propose a method to restrict the search range of scale and translational parameters). In the second step of GA, the fitness of each individual is evaluated. The fitness function is defined for background compensation as:

$$f\left(C_j^i\right) = \frac{1}{\eta} \sum_{\mathbf{x}} |I_t(\mathbf{x}) - Aff\left(I_{t-1}(\mathbf{x})\right)|, \tag{3}$$

where $C_j^i = \left((s_x)_j^i, (s_y)_j^i, (t_x)_j^i, (t_y)_j^i\right)$ is the chromosome j ($j = 1, 2..., N_p$, where N_p is the number of chromosomes) at the ith generation; $I_t(\mathbf{x})$ and $I_{t-1}(\mathbf{x})$ are the pixel values of the point $\mathbf{x} = (x, y)$ in the frame t and $t-1$; Aff is the affine function utilizing $\left((s_x)_j^i, (s_y)_j^i, (t_x)_j^i, (t_y)_j^i\right)$; η is the number of common points in images I_t and I_{t-1}. The third step is selection; it is implicitly coupled with the replacement step. The fourth step is applying the crossover. Two chromosomes (parents) from the current population are randomly selected to be mated. In this paper, we apply uniform crossover to our problem, and we ignore the mutation step for simplicity but not for less effectiveness. All steps are repeated for each generation until a termination condition is met.

To speed up the GA process and to obtain high accuracy, we propose a method to limit the search ranges of scale and translation parameters in the motion model (1). Various studies, attempting to reduce computational cost, use techniques such as hibridizing [8], predicting [9] or imitating [10]. Users should have knowledge about the nonlinearities of real-world applications to define a trade-off between accuracy and computation speed. In this paper, we focus on a method that restricts GA search ranges.

2.2 Limiting the Search Range of Scale Parameters

This section presents the scale parameter estimation algorithm based on histogram matching. It is assumed that scale parameters, s_x and s_y, in the motion model (1) can be separately estimated from translation parameters, t_x and t_y. In this case, the translation matrix becomes $\hat{\mathbf{b}} = (0,0)^T$, and the scale matrix $\hat{\mathbf{A}}$ is formed by $\hat{s}_x$ and $\hat{s}_y$. The estimated parameters, $\hat{s}_x$ and $\hat{s}_y$, are supposed to approximate the actual parameter, s_x and s_y, of the motion model. In fact, s_x and s_y might be inside the range $[\hat{s} - \Delta s; \hat{s} + \Delta s]$ where $\hat{s}$ represents $\hat{s}_x$ or $\hat{s}_y$, and Δs is a value used to define the search range for s_x and s_y around the anchor $\hat{s}$. Hence, the relationship between two images I_t and I_{t-1} is defined as:

$$I_t = I_{t-1}^{\hat{\mathbf{A}}} = \varsigma^{\hat{s}_x}\big(\varsigma^{\hat{s}_y}(I_{t-1})\big) = \varsigma^{\hat{s}_y}\big(\varsigma^{\hat{s}_x}(I_{t-1})\big), \tag{4}$$

where the scale parameter $\hat{\mathbf{A}}$ is decomposed into two directional scale values, one represents the scale with respect to the horizontal direction, the other is for the vertical direction; $\varsigma^{\hat{s}_x}(I_{t-1})$ and $\varsigma^{\hat{s}_y}(I_{t-1})$ are the horizontal and vertical scaling functions of I_{t-1}. The scaling-down functions resize the full-size image I_{t-1} to an image region in I_t, i.e., the scaled I_{t-1} stays inside I_t. Similarly, the scale-up functions resize the full-size image I_t to an image region in I_{t-1}.

We simplify estimating two scale parameters to estimating only $\hat{s}_y$, then we seek for the optimal s_x and s_y inside the range $[\hat{s}_y - \Delta s; \hat{s}_y + \Delta s]$. There are two reasons: 1) it is to reduce the complexity of estimation with an assumption that camera zoom is homogeneous in both horizontal and vertical directions; 2) estimating $\hat{s}_y$ but not $\hat{s}_x$ is to achieve high performance memory access in GPU implementation with the usage of coalesces.

In this paper, vertical and horizontal histograms of gray images are constructed. The matching value of two histograms is calculated based on values of the two histograms. The vertical histogram of an image is constructed by projecting along vertical lines (columns), and one histogram is constructed per vertical line (column). Let $H^V(I)$ be the vertical histogram of the image I:

$$H^V(I) = \big\{h_j^V(I) : j = 0,1,...,W-1\big\}, \tag{5}$$

where W is the number of columns; $h_j^V(I)$ is the histogram of column j of I:

$$h_j^V(I) = \big\{h_{jl}^V(I) : l = 0,1,...,L-1\big\}, \tag{6}$$

where L is the number of bins; $h_{jl}^V(I)$ is the number of pixels with intensities in bin l. The matching value $d^V(.)$ of vertical histograms $h_i^V(I_{t-1}^{\mathbf{r}})$ and $h_j^V(I_t^{\mathbf{r}})$ is

$$d^V\big(h_i^V(I_{t-1}^{\mathbf{r}}), h_j^V(I_t^{\mathbf{r}})\big) = 1 - \xi_4/(2H), \tag{7}$$

where $\xi_4 = \sum_{l=0}^{L-1} \big|h_{il}^V(I_{t-1}^{\mathbf{r}}) - h_{jl}^V(I_t^{\mathbf{r}})\big|$, $I_{t-1}^{\mathbf{r}} = I_{t-1}$, $I_t^{\mathbf{r}}$ is an image region inside I_t if camera zooms out. Similarly, $I_t^{\mathbf{r}} = I_t$, and $I_{t-1}^{\mathbf{r}}$ is an image region of I_{t-1}

when camera zooms in. Let $D^V\big(H^V(I_{t-1}), H^V(I_t), \hat{s}_y\big)$ be the matching value of two vertical histograms of I_{t-1} and I_t:

$$D^V\big(H^V(I_{t-1}), H^V(I_t), \hat{s}_y\big) = \xi_5/(W-1), \tag{8}$$

where ξ_5 is calculated by: If $\hat{s}_y < 1$ (zoom out), $\xi_5 = \sum_{i=0}^{W-1} d^V\big(h_i^V(I_{t-1}), h_i^V(I_t^{\mathbf{r}})\big)$, and if $\hat{s}_y \geq 1$ (zoom in), $\xi_5 = \sum_{i=0}^{W-1} d^V\big(h_i^V(I_{t-1}^{\mathbf{r}}), h_i^V(I_t)\big)$, where $I_{t-1}^{\mathbf{r}}$ and $I_t^{\mathbf{r}}$ are image regions of I_{t-1} and I_t. Scale ratio between two images with respect to vertical direction is determined by searching for the value $\hat{s}_y$ that maximizes the matching value of the vertical projection histograms of two images. Thus, $\hat{s}_y$ is

$$\hat{s}_y = \arg \max_{s_{min} < \hat{s}'_y < s_{max}} D^V\big(H^V(I_{t-1}), H^V(I_t), \hat{s}'_y\big), \tag{9}$$

Fig. 1 shows vertical scaling of the ith column in I_{t-1} and I_t. In case of camera zoom-out, I_t is approximately I_{t-1} added with two marginal regions. Similarly, the case of zoom in, I_{t-1} is approximately I_t plus two marginal sub-images.

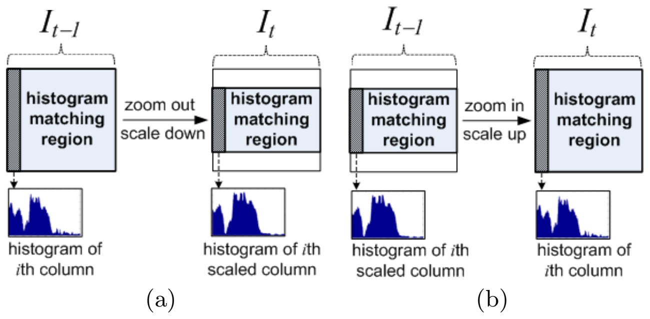
I_{t-1}
I_t
I_{t-1}
I_t
histogram matching region
zoom out
scale down
histogram matching region
histogram matching region
zoom in
scale up
histogram matching region
histogram of ith column
histogram of ith scaled column
histogram of ith scaled column
histogram of ith column
(a)
(b)

Fig. 1. Vertical scaling of a column in two successive images

2.3 Limiting the Search Range of Translation Parameters

This section presents the translation parameters estimation based on histogram matching. It is assumed that the translation parameters, t_x and t_y, in the motion model (1) can be separately estimated from the scale parameters, s_x and s_y. The goal is to estimate $\hat{\mathbf{b}}$ and then to define the GA search range of $\mathbf{b}$ before executing GA for background compensation. The estimated parameter, $\hat{\mathbf{b}}$, is supposed to approximate the actual parameter, $\mathbf{b}$, of the affine model. Indeed, $\mathbf{b}$ might be inside the range $[\hat{\mathbf{b}} - \mathbf{\Delta b}; \hat{\mathbf{b}} + \mathbf{\Delta b}]$ where $\Delta\mathbf{b} = [\Delta t_x, \Delta t_y]^T$ is a value used to define the search range for $\mathbf{b}$ around the anchor $\hat{\mathbf{b}}$.

Similar to scale parameter estimation, the translational displacement between I_{t-1} and I_t in the x-axis direction is determined by searching for the value $\hat{t}_x$ that maximizes the matching value of the vertical histograms of the two images:

$$\hat{t}_x = \arg \max_{-W < \hat{t}'_x < W} D^V\left(H^V\left(I_{t-1}\right), H^V\left(I_t\right), \hat{t}'_x\right). \tag{10}$$

Displacement in y-axis direction is determined by searching for the value $\hat{t}_y$ that maximizes the matching value of the horizontal histograms of the two images:

$$\hat{t}_y = \arg \max_{-H<\hat{t}'_y<H} D^H \left(H^H \left(I_{t-1} \right), H^H \left(I_t \right), \hat{t}'_y \right). \tag{11}$$

where $H^H(I)$ is the horizontal histogram of the image I; $D^H(\cdot)$ is the matching value of two horizontal histograms.

Fig. 2(a) shows horizontal translation of the ith column in I_{t-1} to the $(i+\hat{t}'_x)$th column in I_t with a horizontal displacement $\hat{t}'_x$. Histograms of the two corresponding columns are matched with respect to the common histogram matching region (bold border rectangles) of the two images. Similarly, Fig. 2(b) shows the jth row in I_{t-1} is vertically translated by a displacement $\hat{t}'_y$ to the $(j+\hat{t}'_y)$th row in I_t. With every trial displacement $\hat{t}'_x$ (or $\hat{t}'_y$), the matching value of two vertical (or horizontal) histograms is calculated; later, the best match is found using (10) (or (11)).

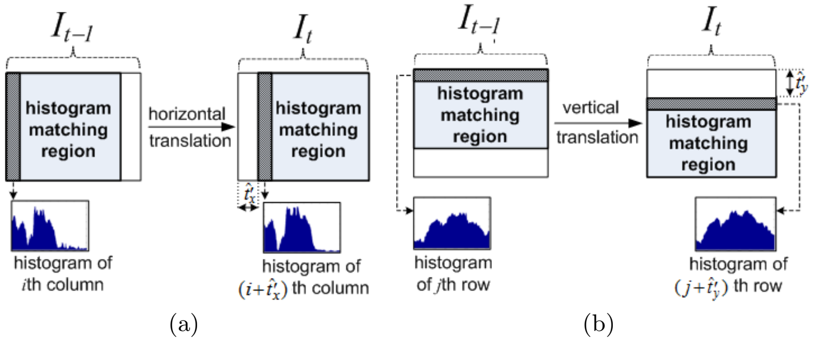
I_{t-1}
I_t
histogram matching region
horizontal translation
histogram matching region
$\hat{t}'_x$
histogram of ith column
histogram of $(i+\hat{t}'_x)$ th column
(a)
I_{t-1}
I_t
histogram matching region
vertical translation
histogram matching region
$\hat{t}'_y$
histogram of jth row
histogram of $(j+\hat{t}'_y)$ th row
(b)

Fig. 2. Horizontal and vertical translation of a column and a row in images

3 GPU Accelerated Implementation

In recent years, processing ability of GPU has rapidly increased. NVIDIA has developed the CUDA (compute unified device architecture) technology to indicate the problems of GPU. In this paper, the proposed range-limited GA search is implemented using CUDA. Fig. 3 summaries the high-level implementation architecture with <<< *kernel* >>> represents a function that is callable from the host (CPU) and executed on the GPU device simultaneously by parallel threads. Stages of GPU implementation are described as below.

3.1 Implementation of Translation Parameter Estimation

Initializing and Calculating $H^V(I_{t-1})$, $H^V(I_t)$, $H^H(I_{t-1})$ and $H^H(I_t)$. $H^V(I_{t-1})$, $H^V(I_t)$, $H^H(I_{t-1})$ and $H^H(I_t)$ are all set to zeros. In our implementation, N_{bins}, the number of bins in a histogram, is equal to the number of threads;

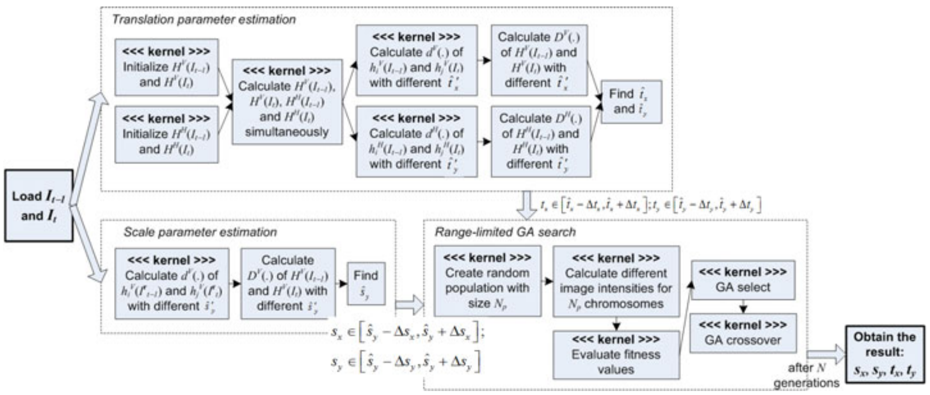
Translation parameter estimation
<<< kernel >>> Initialize $H^V(I_{t-1})$ and $H^V(I_t)$
<<< kernel >>> Initialize $H^H(I_{t-1})$ and $H^H(I_t)$
<<< kernel >>> Calculate $H^V(I_{t-1})$, $H^V(I_t)$, $H^H(I_{t-1})$ and $H^H(I_t)$ simultaneously
<<< kernel >>> Calculate $d^V(.)$ of $h_i^V(I_{t-1})$ and $h_j^V(I_t)$ with different $\hat{t}'_x$
<<< kernel >>> Calculate $d^H(.)$ of $h_i^H(I_{t-1})$ and $h_j^H(I_t)$ with different $\hat{t}'_y$
Calculate $D^V(.)$ of $H^V(I_{t-1})$ and $H^V(I_t)$ with different $\hat{t}'_x$
Calculate $D^H(.)$ of $H^H(I_{t-1})$ and $H^H(I_t)$ with different $\hat{t}'_y$
Find $\hat{t}_x$ and $\hat{t}_y$
Load I_{t-1} and I_t
$t_x \in [\hat{t}_x - \Delta t_x, \hat{t}_x + \Delta t_x]; t_y \in [\hat{t}_y - \Delta t_y, \hat{t}_y + \Delta t_y]$
Scale parameter estimation
<<< kernel >>> Calculate $d^V(.)$ of $h_i^V(I'_{t-1})$ and $h_j^V(I'_t)$ with different $\hat{s}'_y$
Calculate $D^V(.)$ of $H^V(I_{t-1})$ and $H^V(I_t)$ with different $\hat{s}'_y$
Find $\hat{s}_y$
$s_x \in [\hat{s}_y - \Delta s_x, \hat{s}_y + \Delta s_x]$;
$s_y \in [\hat{s}_y - \Delta s_y, \hat{s}_y + \Delta s_y]$
Range-limited GA search
<<< kernel >>> Create random population with size N_p
<<< kernel >>> Calculate different image intensities for N_p chromosomes
<<< kernel >>> Evaluate fitness values
<<< kernel >>> GA select
<<< kernel >>> GA crossover
after N generations
Obtain the result: s_x, s_y, t_x, t_y

Fig. 3. High-level GPU implementation architecture of the proposed range-limited GA

and the number of thread blocks corresponds to $imgHeight$ (or $imgWidth$ in the case of horizontal histograms). Next, pixels from two images are accumulated to vertical and horizontal histograms. At each pixel position, the gray value is first stored to a vertical histogram, then it is accumulated to a horizontal histogram. Hence, a thread is responsible for accumulating two pixel values from two images to projection histograms. The projection histograms, stored in global memories, are distributed among threads. Updating those histograms is data dependent, since many threads might attempt to update the same memory location. This situation results in writing conflicts. The conflict is effective solved with atomic operators [11] used in device functions.

Calculating Matching Values with Different Displacements. The kernel requires a minimum number of threads, $N_t = 16$, and a large number of blocks (e.g., in vertical histogram matching, the number of blocks is $(imgWidth \times numberOfDisplacements)/N_t$). The global index belonging to this kernel, calculated using built-in variables $blockIdx$, $blockDim$ and $threadIdx$, is used to encode the succession of displacements $\hat{t}'_x$ (or $\hat{t}'_y$) and ith column (or row). Hence, differently matching values of every pair of vertical (or horizontal) histograms are calculated and stored in an array of floating point numbers. We notice that sequentially changing the displacements $\hat{t}'_x$, $\hat{t}'_y$ are replaced by simultaneously processing a number of displacements in only one kernel. Thus, there are no loops in kernel code. Afterwards, the best matches $\hat{t}_x$ and $\hat{t}_y$ are found.

3.2 Implementation of Scale Parameter Estimation

This section presents the vertically scaled histogram matching. Similar to the previous section, however, columns in images are scaled and matched in this case. For simplicity, the nearest-neighbor interpolation method is applied to resize image columns. Distinct from separating codes into three kernels in the previous section, the code in this section is integrated into one kernel with the optimized utilization of shared memories. Steps of initializing, accumulating histograms (with interpolation), and calculating matching values are completely manipulated on shared memories. Reasons of this implementation are: 1) only vertical histograms are considered (each image pixel is essentially visited once), 2) based on the first reason, memory conflicts are eliminated in this implementation: one thread processes one column and calculates the corresponding matching value of two vertically scaled histograms, and 3) shared memories, which are potentially 150× faster than global memories and can be as fast as registers, are utilized.

3.3 Implementation of GA

Random Population Generation. A random initial population is generated with a given number of chromosomes. A efficient parallel random number generator [12] is utilized for our purpose. Each initial chromosome is formed by four floating point numbers that satisfy the conditions $s_x \in [\hat{s}_y - \Delta s_x, \hat{s}_y + \Delta s_x]$, $s_y \in [\hat{s}_y - \Delta s_y, \hat{s}_y + \Delta s_y]$, $t_x \in [\hat{t}_x - \Delta t_x, \hat{t}_x + \Delta t_x]$ and $t_y \in [\hat{t}_y - \Delta t_y, \hat{t}_y + \Delta t_y]$; therefore, there are $4N_p$ generated floating point numbers. Each thread is assigned to work on four numbers at once, and access the population as a vector of *float4* to minimize the number of memory accesses.

Fitness Evaluation. Fitness function is the average of the sum of absolute differences between two images. The kernel is launched with one thread per pair of pixels (one pixel is from I_{t-1}, one is from I_t). Each thread calculates the different intensity of $I_t(\mathbf{x})$ and $Aff(I_{t-1}(\mathbf{x})$ as in (3). With the specification of the GeForce GTX 460 used in this paper, our implementation can process the maximum number of pixels $N_t \times N_b = 1024 \times 65535$. For instance, with a fixed size of the two images 320×240, $(N_t \times N_b)/(320 \times 240) \approx 873$ chromosomes can be computed in parallel. We work on $float4$ and compute the final value of each individual's fitness via parallel reduction [13].

Selection and Crossover. The grid of selection and crossover kernels depends on population size. In selection, we define the number of threads equal to population size, a multiple of 16 (to optimize, especially for SIMD-type processing). This means only one block is required. Similarly, in crossover, one block is used.

4 Experimental Results

Eight image sequences were used in experiments. These sequences were acquired by a PTZ camera while tracking moving objects. The camera used in the experiment was a Sony EVI-D100 CCD video camera. The captured images had a resolution of 320×240 pixels. Table 1 specifies their corresponding descriptions, such as environment, lighting condition, background complexity, and distance from camera to object. Fig. 4 shows sample images of test sequences.

We used a PC with an AMD Athlon ×2 Processor 2.90 GHz and 4–GB RAM, and NVIDIA GeForce GTX 460. We compared the error (intensity difference

Table 1. Description of test image sequences

	Indoor/Outdoor	Lighting condition	Background complexity	Distance to object	# images
SQ1	Indoor	Normal	High	4-5m	150
SQ2	Outdoor	Bright	High	10-15m	190
SQ3	Indoor	Dark	Low	11-15m	250
SQ4	Indoor	Dark	Medium	0-7m	320
SQ5	Indoor	Dark	Low	8-32m	340
SQ6	Outdoor	Normal	High	5-7m	440
SQ7	Indoor	Dark	High	4-16m	480
SQ8	Outdoor	Bright	High	50-100m	700

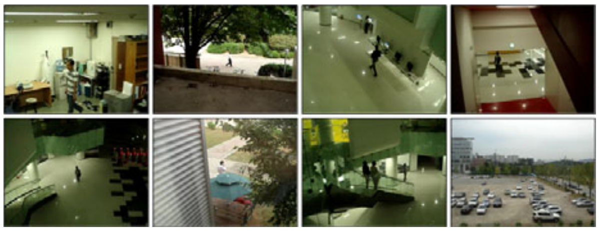

Fig. 4. Sample images of test sequences and correspondences. First row: SQ1, SQ2, SQ3, SQ4. Second row: SQ5, SQ6, SQ7, SQ8.

mean) and execution time of our method with other four methods: two Motion2D methods (M2D_ARN and M2D_AH2N) [4], multi-resolution Hough transform (MHT) [5], and Suhr's method [1]. For MHT, we set parameters as same as in [5]: number of binary images was 32, tolerance value was 0.1, scale range was $[0.9, 1.1]$, $\Delta\theta = \Delta\rho = 0.0003$, and reduction factor $\sigma = 2$, $\mu = 1$. With the method in [1], we set the same parameter values: $t_x, t_y \in [-25, 25]$ with 1 pixel resolution, $s \in [0.9, 1.1]$ with 0.01 resolution, 7 and 5 sub-images for horizontal and vertical feature extraction, window size for local minima and maxima was 3, matching threshold was 10, and matching search radius was 40 pixels.

We set the parameters so they were as similar to those of the other methods as possible. We defined $\hat{t}_x, \hat{t}_y \in [-25, 25]$ with 1 pixel resolution, and $\hat{s}_y \in [0.9, 1.1]$ with 0.01 resolution. GA parameters were: $N_p = 32$, crossover rate $p_c = 0.5$, and number of generations $n_{gen} = 5$. GA search ranges were defined as $\Delta t_x = \Delta t_y = 2$ and $\Delta s_x = \Delta s_y = 0.02$. It was noticed that $\hat{t}_x$, $\hat{t}_y$ and $\hat{s}_y$ were determined using histogram matching; therefore, $s_x \in [\hat{s}_y - \Delta s_x, \hat{s}_y + \Delta s_x]$, $s_y \in [\hat{s}_y - \Delta s_y, \hat{s}_y + \Delta s_y]$, $t_x \in [\hat{t}_x - \Delta t_x, \hat{t}_x + \Delta t_x]$ and $t_y \in [\hat{t}_y - \Delta t_y, \hat{t}_y + \Delta t_y]$. Table 2 shows the experimental results. Our method outperformed the other methods, especially in the critical cases of small difference in background and motion blur. Fig. 5 shows three typical cases of compensation results with comparing the proposed method to the others. Moreover, our processing time was stable to different image sequences where panning, tilting, and zooming have large changes.

Robustness of our method was evaluated in two ways. First, the error was analyzed with respect to the proportion of background with small difference. Based on Table 1, SQ5 and SQ8 were selected in the criterion of low background

Table 2. Error and time (ms) comparison of our method and others

	M2D_ARN		M2D_AH2N		MHT		Suhr's		Ours	
	Error	Time	Error	Time	Error	Time	Error	Time	Error	Time
SQ1	2.77	23.2	2.76	25.9	3.7	999.6	2.94	25.6	**2.61**	**15.6**
SQ2	2.43	30.6	2.42	34.9	3.18	843.9	3.37	33.7	**2.33**	**15.6**
SQ3	1.68	37.7	1.68	41.8	2.04	797.3	2.2	31.0	**1.58**	**16.0**
SQ4	1.5	51.0	1.49	53.6	1.9	649.8	1.69	27.2	**1.4**	**15.7**
SQ5	1.41	53.3	1.4	58.4	1.69	619.1	2.29	33.0	**1.34**	**15.4**
SQ6	3.41	71.7	3.4	77.5	4.52	903.2	5.17	28.4	**3.27**	**15.4**
SQ7	1.72	71.2	1.72	76.8	1.99	867.2	1.91	28.2	**1.63**	**15.4**
SQ8	2.24	108.8	2.23	122.2	2.83	956.6	2.26	30.4	**2.09**	**15.3**
Avg.	2.0	68.2	2.0	73.1	2.49	798.7	2.49	30.6	**1.9**	**15.6**

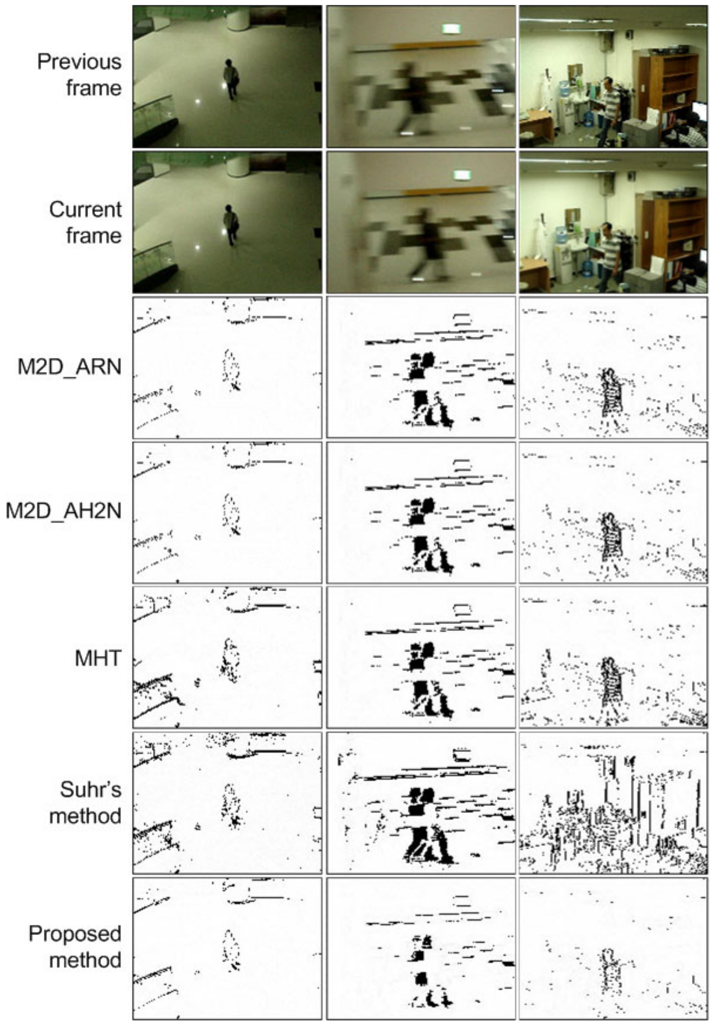

Fig. 5. Results of three typical examples of background compensation. From left to right: small different in background, severe blur, and zoom-in.

complexity. These sequences were corresponding to the background proportion 30–39%, 60–69%, 40–49%, and 70–79%. Fig. 6(a) shows error with different proportion of background with small difference. Suhr's method was not stable because the Hough space used to reject outliers had a large bin size. Second, the error was analyzed in terms of motion blurring effect. There were 530 blurred images manually selected from all sequences for this experiment. Fig. 6(b) presents the error comparison to blurring effect.

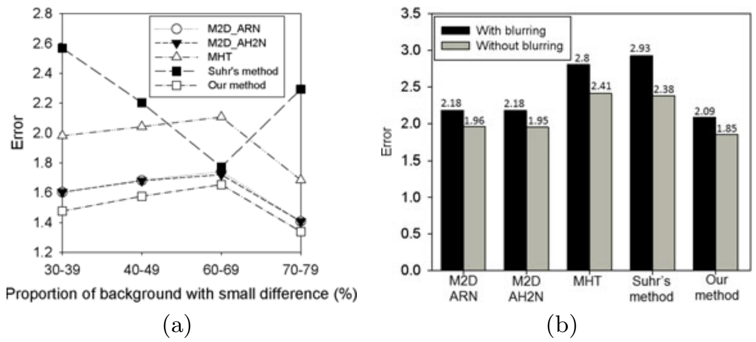

Fig. 6. (a) Error with different proportion of background with small difference; (b) Error comparison to blurring effect

Table 3 presents time percentages of GPU processing. Code to estimate scale parameter were merged into one kernel to achieve full optimization; therefore, its processing time is lowest. Kernels of GA search occupied nearly all GPU time. The fitness evaluation kernel took most of GA time.

Table 3. GPU time percentages of our implemented kernels

Translation parameter estimation	4.516 %	Initialize $H^V(.)$ and $H^H(.)$	0.714 %
		Calculate $H^V(.)$ and $H^H(.)$	14.288 %
		Calculate $d^V(.)$ and $d^H(.)$	84.998 %
Scale parameter estimation	3.226 %	Initialize and calculate scaled histogram matching in one kernel	100 %
Range limited GA search	92.258 %	Create population	3.497 %
		Evaluate fitness	92.702 %
		Select	2.524 %
		Crossover	1.277 %

5 Conclusion

Our GPU accelerated and range-limited GA search is faster than the Motion2D methods by 4.5 times, the MHT method by 53 times, and the Suhr's method by 2 times. The computational improvement of our method is not only due to limiting the GA search ranges but also due to the GPU implementation techniques.

In this paper, the combination of GA and GPU implementation techniques is successfully applied to the background compensation problem for PTZ cameras. By experimentally comparing the results of our method to other methods, our work has two advantages. First, our method is robust in coping with critical situations, because the GA was proved that it can reach an optimal solution. Second, its processing time is very fast with the graphics card based implementation.

We plan to port the current implementation to a NVIDIA Tegra SoC mobile processor [14] to achieve high applicability on mobile computing platforms.

References

1. Suhr, J.K., Jung, H.G., Li, G., Noh, S.I., Kim, J.H.: Background Compensation for Pan-Tilt-Zoom Cameras using 1-D Feature Matching and Outlier Rejection. IEEE Trans. Circuits Syst. Video Technol. 21(3), 371–377 (2011)
2. Murray, D., Basu, A.: Motion Tracking with an Active Camera. IEEE Trans. Pattern Anal. Mach. Intell. 16(5), 449–459 (1994)
3. Araki, S., Matsuoka, T., Yokoya, N., Takemura, H.: Real-time Tracking of Multiple Moving Object Contours in a Moving Camera Image Sequence. IEICE Trans. Inform. Syst. E83–D(7), 1583–1591 (2000)
4. Odobez, J., Bouthemy, P., Temis, P.: Robust Multi-resolution Estimation of Parametric Motion Models in Complex Image Sequences. J. Vis. Commun. Image Represent. 6, 348–365 (1995)
5. Pham, X.D., Cho, J.U., Jeon, J.W.: Background Compensation using Hough Transformation. In: IEEE Int. Conf. Robot. Autom., pp. 2392–2397 (2008)
6. Moscheni, F., Vesin, J.: A Genetic Algorithm for Motion Estimation. In: 15eme Colloque sur le Traitement des Signaux et Images, France, pp. 825–828 (1995)
7. Gong, M., Yang, Y.H.: Quadtree-Based Genetic Algorithm and Its Applications to Computer Vision. Pattern Recognit. 37(8), 1723–1733 (2004)
8. Rodehorst, V., Hellwich, O.: Genetic Algorithm Sample Consensus (GASAC) - A Parallel Strategy for Robust Parameter Estimation. In: IEEE Int. Conf. Comput. Vis. Pattern Recognit. Workshop, pp. 103–110 (2006)
9. Mutoh, A., Nakamura, T., Kato, S., Itoh, H.: Reducing Execution Time on Genetic Algorithm in Real-World Applications using Fitness Prediction: Parameter Optimization of SRM Control. In: IEEE Congress Evol. Comput., pp. 552–559 (2003)
10. Jin, Y., Sendhoff, B.: Reducing Fitness Evaluations using Clustering Techniques and Neural Network Ensembles. In: Genetic Evol. Comput. Conf., pp. 688–699 (2004)
11. Shams, R., Kennedy, R.A.: Efficient Histogram Algorithms for NVIDIA CUDA Compatible Devices. In: Int. Conf. Signal Process. and Commun. Syst., pp. 418–422 (2007)
12. Tzeng, S., Wei, L.Y.: Parallel White Noise Generation on a GPU via Cryptographic Hash. In: Symp. on Interactive 3D Graphics and Games, pp. 79–87 (2008)
13. Harris, M., Sengupta, S., Owens, J.D.: Parallel Prefix Sum (Scan) in CUDA. Chapter 39 – GPU Gems 3 (2008)
14. NVIDIA Tegra Developer Zone, `http://tegradeveloper.nvidia.com/tegra`