Journal of Information Hiding and Multimedia Signal Processing ©2014 ISSN 2073-4212
Ubiquitous International Volume **5**, Number **1**, January **2014**

A Fast Algorithm of Temporal Median Filter for Background Subtraction

Mao-Hsiung Hung[1],Jeng-Shyang Pan[1,2]and Chaur-Heh Hsieh[3]

[1]Dept. of Electronic Engineering, National Kaohsiung University of Applied Sciences
[2]Innovative Information Industry Research Center (IIIRC), Shenzhen Graduate School, Harbin Institute of Technology
[3]Dept. of Computer and Communication Engineering, Ming Chuan University

*: Corresponding author
E-mail: hsiehch@mail.mcu.edu.tw

Received February, 2013; revised June, 2013

Abstract. *Temporal median filter is one of most popular background subtraction methods. However, median operation is very time-consuming which limits its applications. This paper presents a fast algorithm to reduce the computation time of the temporal median operation. By utilizing the characteristics of high correlation of adjacent frames, the fast algorithm designs simple mechanism to check whether the median of the current frame is equal to that of the previous frame. The proposed algorithm reduces the computing frequency of median operations significantly, and the experimental results indicate it is much faster than the existing algorithms.*

Keywords: background subtraction, temporal median filter, fast algorithm

1. Introduction.

Background subtraction is an important step in many computer vision applications such as video surveillance, people counting, human gesture recognition, moving object detection and tracking, traffic monitoring, and video indexing and retrieval. Background subtraction detects moving objects from the difference between the current frame and a background image. To obtain accurate detection of moving objects, the background image must be a representation of the scene with no moving objects and must be updated regularly so as to adapt to the varying lighting conditions and geometry settings [1]. Many background subtraction methods have been proposed in the literatures such as running Gaussian average, temporal median filter, mixture of Gaussians, kernel density estimation, etc. The major problems exist in these methods are either computation expensive or memory expensive[1].With the rapid advance of memory technology, memory cost is getting less critical in recent years.

Temporal median refers the median of previous frames in a video sequence to establish a statistical background model for background subtraction. Lo and Velast in [2] first presented the temporal median background update technique for congestion detection system of underground platform. Cucchiara et al. [3] pointed out that emporal median filter provides an adequate background model which immediately reflects sudden scene change. Temporal median filter offers acceptable accuracy while achieving a high frame rate and having limited memory requirements [1].Therefore, it has become one of most popular background subtraction methods applied by many computer vision systems [4-8].

The temporal median of the k-th frame, $I_{Med}(x, y, k)$, is obtained by

$$I_{\text{Med}}(x, y, k) = \text{Med}(I(x, y, k-1), I(x, y, k-2), \ldots, I(x, y, k-N)), \tag{1}$$

where $I(x, y, k-1), \ldots, I(x, y, k-N)$, denote pixel values located at (x,y) over the previous Nframes of the k-th frame and Med(.) means the median operation. The methods of the median operation are categorized into sort-based and selection-based. Sort-based median operation isthe simplest method. It sorts input data and then picks the middle-order element from the sorted sequence, which requires significant computational complexity. Selection-based median methods do not need to sort input data to determine the median, so they are more efficient than sort-based. Histogram method [9-10] and divide-and-quer method [11] are good examples of the selection-based median methods.

However, the selection-based method still consumes considerable time, so that it is hard to satisfy real-time requirement for some applications, especially for high video quality applications. This paper presents a fast algorithm to further speed up the temporal median operation, which is based on histogram. Because pixel data are very high correlated frame by frame, it has high possibility that the two medians of the consecutive frames are equal. We present a scheme to check the median repetition between two consecutive frames. The experimental results indicate our proposed method reduces the computation significantly, and it far exceeds the real-time requirement.

2. **Proposed Method.**

2.1. **Median selection based on histogram.**
For the convenience of the representation, we redefine the temporal neighborhood of the pixel at (x,y) of the k-th frame as

$$D_k = \{d_{k-1}, d_{k-2}, \ldots, d_{k-N}\}, \tag{2}$$

Where the data set $D_k$stores the previous N pixel data(d_{k1},$d_{k-2}, \ldots, d_{k-N}$) located at (x, y). Here we assume that the pixel value is in the range of 0 and 255. O_{Mid} means the middle order of the N data, i.e. $O_{\text{Mid}} = (N-1)/2$ (N must be odd). Then, the median selection based on the histogram is described as the following function.

```
**Function:** $m$=medhist($D$)
// Input: $D$ stores the previous $N$ pixel data
// Output: $m$ returns the median of $D$

$hn$=hist($D$) // hist(.) returns the histogram of the data set
$csum$=0 //$csum$ means the cumulative function of the histogram
for $i$=0 to 255
    if $hn[i] > 0$ then
        $csum$+=$hn[i]$
        if $csum \geq O_{\text{Mid}}$ then break
    end if
end for
return $i$
```

The above median determination first calculates the histogram of the input data set. Then, the cumulative function of the histogram is evaluated by incrementing index from 0 to 255. When the cumulative function reaches the middle order, the current index is the median of the data set required.

2.2. **Median selection based on histogram and repetition checking.** To develop the fast algorithm of the median determination, we first design a lower bound and anupper bound of the cumulative function at the median, denoted by lb and ub. By slightly modifying the above histogram method, we obtain the following function to evaluate the median as well as the two bounds of a data set.

```
Function:{$m$,$lb$, $ub$} =medhist_bnd($D$)
//Input:$D$ stores the previous $N$ pixel data
/* Ouput: $m$ returns the median of $D$. $lb$ and $ub$ respectively returns the lower bound
and upper bound of the cumulative function at the median. */
$hn$=hist($D$) // hist(.) returns the histogram of the data set
$csum$= 0 // $csum$ is the cumulative function of the histogram
for $i$=0 to 255
    if $hn[i] > 0$
        $csum$+=$hn[i]$
        if $csum \geq O_{\text{Mid}}$ then
            $lb = csum - hn[i]+1$, $ub = csum$
            break
        endif
    end if
end for
return $i$,$lb$,$ub$
```

Similar to the original histogram method, when the cumulative function of the histogram ($csum$) reaches the middle order, lb can be obtained by $lb = csum - hn[i]+1$, where $hn[i]$ is the value of the indexed histogram and ub is equal to ($csum$). Fig. 1 shows an example of cumulative function of histogram, and obviously the middle order satisfies the relation of $lb \leq O_{mid} \leq ub$. We apply the relation to develop the repetition checking scheme.

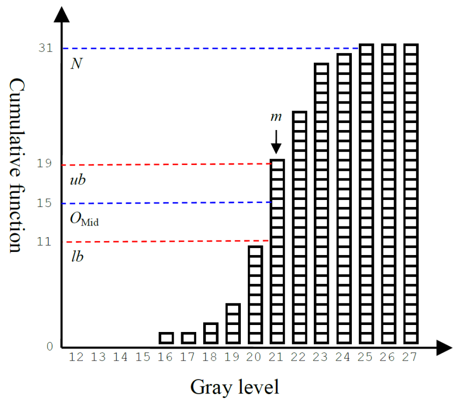

FIGURE 1. An example of cumulative function of histogram

The data set of D_k contains pixel data within the previous N frames of the k-th frame, as shown in Eq.(2). The next data set of $D_{k+1}$for (k+1)-th frame is as

$$D_{k+1} = \{d_k, d_{k-1}, \ldots, d_{k-N+1}\}, \tag{3}$$

where d_k is the pixel data of the k-th frame. The difference of D_{k+1} and D_k is just $d_k$and d_{k-N}, which implies D_{k+1} and D_k are highly correlated. Thus, it has high possibility that the medians of the two data sets are equal, which we call median repetition. Consequently, the proposed repetition checking of the median between the two consecutive frames has promise to greatly reduce the median operation in temporal direction.

The temporal median filter with histogram method and repetition checking is implemented by the following function.

```
Function: {$m_{k+1}$,$lb_{k+1}$, $ub_{k+1}$}=medhist_repchk($D_k$, $d_k$,$m_k$,$lb_k$,$ub_k$)
/* Input: $D_k$ stores the previous $N$ pixel data of the $k$-th frame. $d_k$ is the pixel data of the $k$-th frame. $m_k$, $lb_k$ and $ub_k$ are respectively the median, $lb$ and $ub$ of $D_k$. */
/*Ouput: $m_{k+1}$returns the median of $D_{k+1}$. $lb_{k+1}$ and $ub_{k+1}$ respectively return $lb$ and $ub$ of $D_{k+1}$.*/

insert $d_k$ into $D_k$ and delete $d_{k-N}$ from $D_k$ to obtain $D_{k+1}$
{$tf$,$lb_{k+1}$, $ub_{k+1}$} =repchk($d_{\mathrm{kt}N}$,$d_k$, $m_k$,$lb_k$,$ub_k$)//call function repchk(.) for repetition checking
if $tf$ then
    $m_{k+1} = m_k$// if repchk(.) returns $tf$=1
else
    {$m_{k+1}$,$lb_{k+1}$, $ub_{k+1}$} =medhist_bnd($D_{k+1}$) // if repchk(.) returns $tf$=0
end if
return $m_{k+1}$,$lb_{k+1}$, $ub_{k+1}$
```

Given the data and parameters of the k-th frame, d_{k-N}, d_k, m_k, lb_k and ub_k, the repetition checking algorithm first calculate the parameter of the next frame, lb_{k+1} and ub_{k+1}, according to the relations of the values of d_{k-N}, d_k and m_k. The relation of d_{k-N} and m_k contains three cases:"less than", "equal to" and "greater than". The relation of d_k and m_k also include the same three cases. Thus, the two relations generate nine permutations, which can be utilized tocalculatelb_{k+1} and ub_{k+1} from lb_k and ub_k. For examples, when the deleted element d_{k-N} and the inserted element d_k are less than the previous median of m_k, lb and ub are unchanged, i.e. lb_{k+1}=lb_k and ub_{k+1}=ub_k. When d_{k-N} is less than m_k and the d_k is equal to m_k, lb is decreased by 1 but ub is unchanged, i.e. $lb_{k+1} = lb_k - 1$ and $ub_{k+1} = ub_k$. Similarly, the other seven conditions are used to update the next lower bound and upper bound. Finally, if $lb_{k+1} \leq O_{\mathrm{Mid}} \leq ub_{k+1}$, the median of the next frame is equal to that of the current frame; i.e., $m_{k+1} = m_k$. The repetition checking is implemented by the following function.

```
Function:{$tf$,$lb_{k+1}$,$ub_{k+1}$}=repchk($d_{\mathrm{kt}N}$,$d_k$,$m_k$,$lb_k$,$ub_k$)
/* Input: $d_{k-N}$ and $d_k$ respectively denote the deleted and inserted element for the next data set.
$m_k$ means the previous median. $lb_k$ and $ub_k$ are the previous bounds of the cumulative function at the median. */
/* Output: $tf$ returns 1 if the current median is equal to the previous median, otherwise $tf$ returns 0. */

if $d_{k-N} < m_k$ and $d_k < m_k$, then $lb_{k+1}$=$lb_k$, $ub_{k+1}$=$ub_k$
else if $d_{k-N} < m_k$ and $d_k = m_k$, then $lb_{k+1}$=$lb_k$−1, $ub_{k+1}$=$ub_k$
else if $d_{k-N} < m_k$ and $d_k > m_k$, then $lb_{k+1}$=$lb_k$−1, $ub_{k+1}$=$ub_k$−1
else if $d_{k-N} = m_k$ and $d_k < m_k$, then $lb_{k+1}$=$lb_k$+1, $ub_{k+1}$=$ub_k$
else if $d_{k-N} = m_k$ and $d_k = m_k$, then $lb_{k+1}$=$lb_k$, $ub_{k+1}$=$ub_k$
else if $d_{k-N} = m_k$ and $d_k > m_k$, then $lb_{k+1}$=$lb_k$, $ub_{k+1}$=$ub_k$−1
else if $d_{k-N} > m_k$ and $d_k < m_k$, then $lb_{k+1}$=$lb_k$+1, $ub_{k+1}$=$ub_k$+1
else if $d_{k-N} > m_k$ and $d_k = m_k$, then $lb_{k+1}$=$lb_k$, $ub_{k+1}$=$ub_k$+1
else if $d_{k-N} > m_k$ and $d_k > m_k$, then $lb_{k+1}$=$lb_k$, $ub_{k+1}$=$ub_k$

if $lb_{k+1} \leq O_{\text{Mid}}$ and $O_{\text{Mid}} \leq ub_{k+1}$, then $tf$=1 else $tf$=0
return $tf$,$lb_{k+1}$, $ub_{k+1}$
```

To further speed-up the repetition checking above, we design a table look-up scheme which maps the nine rules above into a table. We define the differential bounds as $\Delta lb = lb_{k+1} - lb_k$ and $\Delta ub = ub_{k+1} - ub_k$. The nine conditions are encoded into the index with value from 0 to 8, which is calculated by c=3×$a + b$. Table 1 lists the relation of Δlb and Δub with the index c. When Δlb and Δub are obtained by looking up the table with the index c, the new bounds lb_{k+1} and ub_{k+1} can be respectively calculated by $lb_k + \Delta lb$ and $ub_k + \Delta ub$. The table look-up scheme is listed in the following.

Code: Table look-up scheme to calculate lb_{k+1} and ub_{k+1}

```
if $d_{k-N} < m_k$, then $a$=0 else if $d_{k-N} = m_k$, then $a$=1 else if $d_{k-N} > m_k$, then $a$=2
if $d_k < m_k$, then $b$=0 else if $d_k = m_k$, then $b$=1 else if $d_k > m_k$, then $b$=2
$c$=3×$a + b$
$lb_{k+1}$=$lb_k + \Delta lb[c]$ , $ub_{k+1}$=$ub_k + \Delta ub[c]$
```

Table 1. Lookup table of Δlb and Δub

a	b	c=3×a+b	$\Delta lb[c]$	$\Delta ub[c]$
0	0	0	0	0
0	1	1	–1	0
0	2	2	–1	–1
1	0	3	1	0
1	1	4	0	0
1	2	5	0	–1
2	0	6	1	1
2	1	7	0	1
2	2	8	0	0

The cost paid by the proposed fast algorithm is extra memory requirement for the repetition checking. It needs $(2 \times \lceil \log_2 N \rceil + 8) \times H \times W$ bits to store lb_k, ub_k and $m_k$in advance for all pixels in the frame, where $H \times W$ is the frame size. For example, when N=31, the additional memories are 2.25 frames, where 1.25 frames are used for storing lb_k and ub_k, and 1 frame for storing m_k. In this case, the extra memory cost is approximate 7% of the previous frame buffer, which is relatively small.

3. **Experimental Results.** In the experiments, we evaluate our proposed method with six standard testing videos. Among those videos, "Akiyo" and "news" are with low motion, "hall monitor" and "container" are with middle motion, and "foreman" and "coast guard" are of high-motion videos. The formats of the videos consist of CIF (352×288) and QCIF (176×144)with frame rate of 30 Hz. Each video contains 300 frames. We use 31 previous frames to calculate the temporal median, i.e. N=31.The testing bench uses Visual C++ with Intel Core2 Duo at 2.53GHz. We compare the processing times of our method with existing four methods, and the results are listed in Table 2 and Table 3 for CIF and QCIF, respectively. The methods for comparison contain two sort-based methods and two selection-based methods. The former are bubble sort and quick sort, and

the later are divide-and-conquer and histogram methods. Our proposed method is based on histogram method and repetition checking. The average processing time of the five videos is listed in the last second column. The time ratio with respect to the proposed method is listed in the last column. The time ratio is defined as the average processing time of a method divided by that of our method.

The experimental results indicate that the proposed method performs significantly faster than other methods for all cases. Moreover, the proposed fast algorithm can achieve67.5 fps (=300/4.443) for CIF and 336.7 fps (=300/0.891) for QCIF that greatly excesses the real-time requirement. The proposed method performs 2.07 times and 2.32 times faster than the histogram method for CIF and QCIF, respectively. The results prove that the repetition checking effectively reduces the computing frequency of the temporal median operation.

Table 2. Comparison of processing times of various methods for CIF videos(in sec)

Methods	Akiyo	News	Hall monitor	Container	Foreman	Coast guard	Average	Time ratio
Bubble sort	30.390	33.921	45.359	43.859	49.468	52.828	42.638	9.60
Quick sort	32.688	32.797	33.906	32.812	33.563	34.031	33.300	7.50
Divide-and-conquer	7.453	9.032	13.875	12.735	15.140	16.312	12.425	2.80
Histogram	7.906	7.781	8.969	9.015	10.985	10.391	9.175	2.07
Proposed	3.344	3.703	3.781	4.125	5.859	5.844	4.443	1.00

Table 3. Comparison of processing times of various methods for QCIF videos (in sec)

Methods	Akiyo	News	Hall monitor	Container	Foreman	Coast guard	Average	Time ratio
Bubble sort	7.078	7.890	9.828	9.703	11.671	12.500	9.778	10.98
Quick sort	7.875	7.985	8.000	7.890	8.063	8.296	8.018	9.00
Divide-and-conquer	1.531	1.906	2.922	2.516	3.453	3.719	2.675	3.00
Histogram	1.766	1.781	2.015	2.031	2.500	2.328	2.070	2.32
Proposed	0.640	0.734	0.703	0.813	1.266	1.188	0.891	1.00

To investigate the computational complexity of the five methods, we measure the processing times under six different previous frame numbers (N) including 15, 31, 45, 61, 75 and 91. Fig. 2(a) and Fig. 2(b) show the experimental results for hall monitor and foreman sequences respectively. The results indicate that the processing times of the two sort-based methods increase approximately linearly with the increase of N, and those of the other three methods slightly raise with the increase of N. Comparing to the other four methods, the proposed method provides the least increasing rate over previous frame numbers.

Table 4 lists the comparison of the computational complexity of the five algorithms in terms of computation order of best case, worst case and average case. Here we use 50% of median repetition rate to evaluate the computational complexity of our algorithm in the average case and obtain $O(N)$. According to our investigation, the median repetition rates of Akiyo, news, hall monitor, container, foreman and coastguard sequences are respectively 94.72%, 90.52%, 92.53%, 87.6%, 69.1% and 68.12% for QCIF at N=31. Therefore, the computational complexity of the proposed algorithm would be close to the best case, i.e. $O(1)$, in practice.

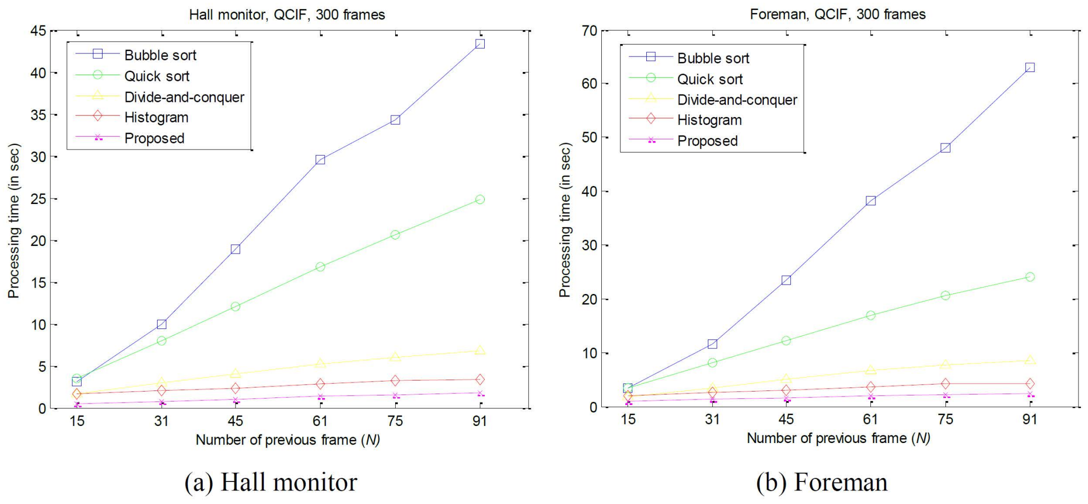

Figure 2. The time complexity analysis of the five algorithms

Table 4. Computational complexity of the five algorithms

	Best case	Average case	Worst case
Bubble sort	$O(N\log N)$	$O(N^2)$	$O(N^2)$
Quick sort	$O(N\log N)$	$O(N\log N)$	$O(N^2)$
Divide-and-conquer	$O(N)$	$O(N)$	$O(N^2)$
Histogram	$O(N)$	$O(N)$	$O(N)$
Proposed	$O(1)$	$O(N)$	$O(N)$

4. **Conclusion.** In this paper, we have presented a fast algorithm to speed up temporal median filter for background subtraction. This algorithm is mainly based a novel repetition checking scheme. The lower and upper bounds of the cumulative function of the histogram are developed for the check of median repetition. Due to the high correlation of the pixel data between consecutive frames, the repetition checking effectively reduces the computing frequency of the median operation. The results indicate that the proposed algorithm performs approximate two times faster than histogram method, which is the state-of-the-art. Because our algorithm far exceeds the real-time requirement, it makes the temporal median filter much more applicable.

Acknowledgement:
This work was supported in part by National Science Counsel Granted NSC 98-2221-E-151-036-MY3 and NSC 98-2811-E-214-151-001.

REFERENCES

[1] M. Piccardi, Background subtraction techniques: a review, *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, pp. 3099-3104, 2004.

[2] B. P. L. Lo, and S. A. Velastin, Automatic congestion detection system for underground platforms, *Proc. Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing*, pp. 158-161, 2001.

[3] R. Cucchiara, C. Cranna, M. Piccardi, and A. Prati, Detecting moving objects, ghosts and shadows in video streams, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 25, no. 10, pp. 1337-1442, 2003.

[4] S. -C. S. Cheung, and C. Kamath, Robust techniques for background subtraction in urban traffic video, *Proc. of Visual Communications and Image Processing*, SPIE 5308, pp. 881-892, 2004.

[5] R. Cutler, and L. Davis, View-based detection and analysis of periodic motion, *Proc. of the 14th International Conference on Pattern Recognition*, pp. 495-500, Aug 1998.

[6] Q. Zhou, and J. Aggarwal, Tracking and classifying moving objects from videos, *Proc. of IEEE Workshop on Performance Evaluation of Tracking and Surveillance* , pp. 52-59, 2001.

[7] N. J. B. McFarlance, and C. P. Schofield, Segmentation and tracking of piglets in images, *Journal of Machine Vision and Applications*, vol. 8, no. 3, pp. 187-193, 1995.

[8] J. Cai, M. Shehata, and W. Badawy, A robust video-based algorithm for detecting snow movement in traffic scenes, *Journal of Signal Processing Systems*, vol. 56, no. 2-3, pp. 307-326, 2009.

[9] T. S. Huang, G. J. Yang, and G. Y. Tang, A fast two-dimensional median filtering algorithm, *IEEE Trans. Acoustics, Speech, And Signal Processing*, vol. 27, no. 1, pp. 13-18, 1979.

[10] S. Perreault, and P. Hebert, Median filtering in constant time, *IEEE Trans Image Processing*, vol. 16, no. 9, pp. 2389-2394, 2007.

[11] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*, McGraw-Hill, New York, USA, 2008.