Speedup the Multi-Camera Video-Surveillance System for Elder Falling Detection

Wann-Yun Shieh*, Ju-Chin Huang
Department of Computer Science and Information Engineering
Chang Gung University, Taiwan
wyshieh@mail.cgu.edu.tw, m9529009@stmail.cgu.edu.tw

Abstract

Nowadays, all countries have to face the growing populations of elders. For most elders, unpredictable falling accidents may occur at the corner of stairs or a long corridor due to body functional decay. If we delay to rescue a falling elder who is likely fainting, more serious consequent injury may happen. Traditional secure or video surveillance systems need someone to monitor a centralized screen continuously, or need an elder to wear sensors to detect accidental falling signals, which explicitly require higher costs of care staffs or cause inconvenience for an elder.

In this work, we propose a human-shape-based falling detection algorithm and implement this algorithm in a multi-camera video surveillance system. The algorithm uses multiple cameras to fetch the images from different regions required to monitor. It then uses a falling-pattern recognition approach to determine if an accidental falling has occurred. If yes, the system will send short messages to someone needs to alert.

In addition, we propose a multi-video-stream processing algorithm to speedup the throughput for the video surveillance system having multiple cameras. We partition the workloads of each video-surveillance streaming into four tasks: image fetch, image processing, human-shape generation, and pattern recognition. Each task will be handled by a forked thread. When the system receives multiple video streams from cameras, there are four simultaneous threads executed for different tasks. The objective of this algorithm is to exploit large thread-level-parallelism among those video-stream operations, and apply pipelining technique to execute these threads.

All above algorithms have been implemented in a real-world environment for functionality proof. We also measure the system performance after multi-streaming speedup. The results show that the throughput can be improved by about 2.12 times for a four-camera surveillance system.

Keywords: Falling detection, pattern recognition, pipelining multithread, video surveillance, multiple cameras

1. Introduction

One of the most serious injuries for elders comes from the falling accidents. Most conventional falling detection approaches use wearable sensors to catch the signals from body activities [1]. However, an elder may forget to wear sensors, or feel uncomfortable in daily life. Moreover, in case the elder loses consciousness, we cannot distinguish what happens.

A more intelligent solution to detect falling is to use the video surveillance system [2][6][9]. It uses cameras to capture human activities in different high-risk regions, and applies image processing with pattern recognition techniques to identify falling accidents. However, a multi-camera video-surveillance system is usually hungry for large computation power on video streaming processing, especially with the function of falling detections. When we extended a surveillance system to support the multi-camera falling detection, we found that the performance (or throughput) decayed very quickly, almost proportional to the number of cameras added.

To overcome the performance limitations, we develop a multi-camera video surveillance system, which uses the image processing and pattern recognition techniques to perform falling detection on each video source, and explores the thread-level-parallelism on multiple cameras to enhance system performance. For each single video stream, we design a human-shape retrieving algorithm to get the outside edges from a person image and compare the shape with some falling patterns. If any one of the results hits, a

978-0-7695-3678-1/09 $25.00 © 2009 IEEE
DOI 10.1109/ICESS.2009.62

warning message will be sent to other persons via mobile phones.

On the other hand, for multiple video streams from different cameras, we partition the process described above into four balanced tasks: image fetch, image processing, human-shape generation, and pattern recognition. Each task is performed by a specified thread, and totally we will have four threads to be executed concurrently. Our goal is to apply the pipelining technology on these threads such that the throughput of a multi-camera system can be improved effectively. Though pipelining would not decrease the time to complete the task of single-camera falling detection, but when we have many tasks (cameras) to do, the improvement of throughput decreases the total time to complete the work [7].

The rest of this paper is organized as follows. Section 2 shows the related works about falling detection via video surveillance. Section 3 shows our falling detection approach for each video stream. Section 4 shows our pipelining approach to enhance throughput on multiple video sources. Section 5 shows the experiment results, and finally Section 6 is our conclusions.

2. Related Works

The falling detection via video surveillance has attracted attentions in recent years. Vinay Vishwakarma et al. [12] used the state transitions to detect a falling process. They used a bounding box to represent the person, and with the analyses of the aspect ratio, the gradient of the object in X-Y plane, and the fall angles, they built a state transition diagram, and confirm a falling by states.

Other works, e.g., Rougier et al. [3], observed that if a serious falling occurs, the person is almost little movement or not changing the postures. They quantified the extracted images from human historical motion to find the movements like falling and analyzed the variation on the motion or the orientation of the ellipse to confirm if a person falls.

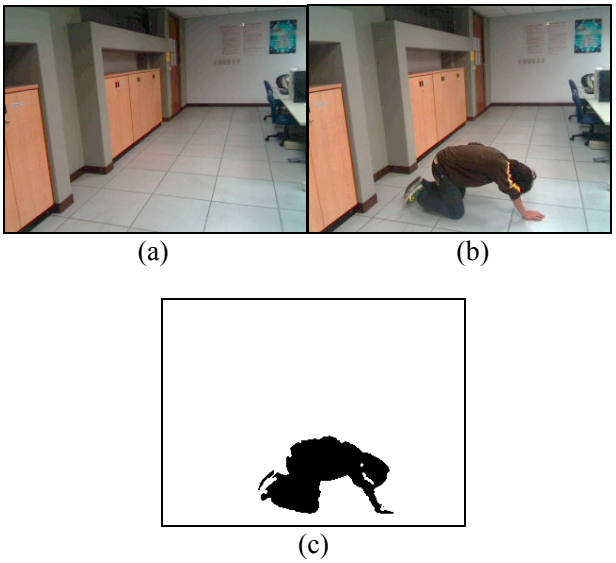
In this paper, instead of detecting the "process" of the falling, we designed a human-shape detection algorithm which is simpler to confirm if a falling accident has occurred, and determine if we have to inform someone as soon as possible. In addition, the most difference between our work and other previous works is that we apply the pipelining and multithreading technique to speedup the image processing from multiple video streams. Such software application is more suitable for current computer architecture which has higher hardware parallelism.

3. Elder Falling Detection via single camera

For each video stream from a single camera, we capture the image and extract the human shape edge in it. Then we perform the falling detection process by a pattern recognition-based approach which compares the extracted human shape with some possible falling postures, e.g., lying or bending down on the ground. The whole processing can be partitioned into four steps:

Step 1: Image Fetch

To grab a human shape from an image, we use the "background subtraction method" [5], which initially captures a static image as the background. After the system is activated, the camera will capture instantaneous images sequentially in a fixed rate. We then subtract the background from each instantaneous image to fetch the differential objects between them. Figure 1 shows such an example, in which (a) is the background image, (b) is one instantaneous image, and (c) is the object of a human shape fetched after the background subtraction.



(a) (b)

(c)

Figure 1. An example of the background subtraction (a) background (b) instantaneous image (c) results after subtraction.

Step 2: Image Processing

The step after the *Image Fetch* is to detect (delineate) the edge from a human shape (e.g., Figure 1(c)) via image processing techniques [5]. The techniques include the noise reduction, dilation/erosion, and edge thinning, etc. After these processes, the edge of a human shape can be limited in one-pixel wide.

(This step is also known as the one-pixel-wide edge detection [8].) Figure 2 shows the result after the edge-detection from the image in Figure 1(c)

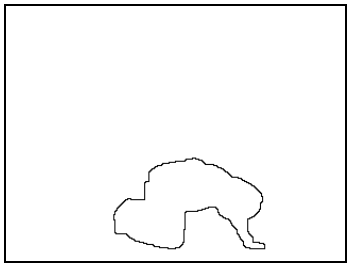


Figure 2. An example of the one-pixel-wide edge-detection from Figure 1(c).

Step 3: Human Edge Generation

In this step, we translate the edge of a human shape (i.e., Figure 2) into a sequence of characters for later pattern recognition. Take the edge of Figure 2 as example. We insert the grids in the image to gather the relative relations among the edge pixels. Figure 3 shows the partial grids with the upper edge of the human shape in Figure 2. If we traverse the grids along the human edge in a clockwise direction, we will obtain a sequence of directions -- each from the comparison between any two of neighbor pixels. The directions include "down", "left", "up", and "right". If we define these four directions with the characters of "0", "1", "2", and "3" respectively, we can generate a sequence of characters finally representing the characteristics of the edge on a human shape. Figure 3 shows the partial results after the translation from the grids of the edge to the sequence of characters.

Step 4: Pattern Recognition

To determine whether an unknown human shape represents a falling elder, we compare the sequence of characters generated in Step 3 with some falling-like postures (patterns). The postures include the sitting, bending, and lying on the ground in different directions. These postures will be preprocessed by the same work in Step 3. Therefore we will have their sequences of characters stored for later pattern recognition. The remaining work in this step is to apply the sequence comparison algorithm [4][11] to calculate how similar between the unknown human shape and those falling-like patterns.

4. Pipeline Processing for Multi-Image-Streams

To improve the throughput of the system supporting multi-camera falling detection (in different places), we apply the multi-threading technique to above four steps and exploit their thread-level parallelism by pipelining execution.

We create four threads, and each of them performs one specific task: one thread for the image fetch (*IF*), one for the image processing (*PR*), one for the human-edge sequence generation (*SG*), and one for the pattern recognition (*PR*). Every thread has equally balanced workloads to execute different tasks on different video streams. They will be activated at the same time. Therefore there will be four threads executed concurrently like the function units in a pipeline.

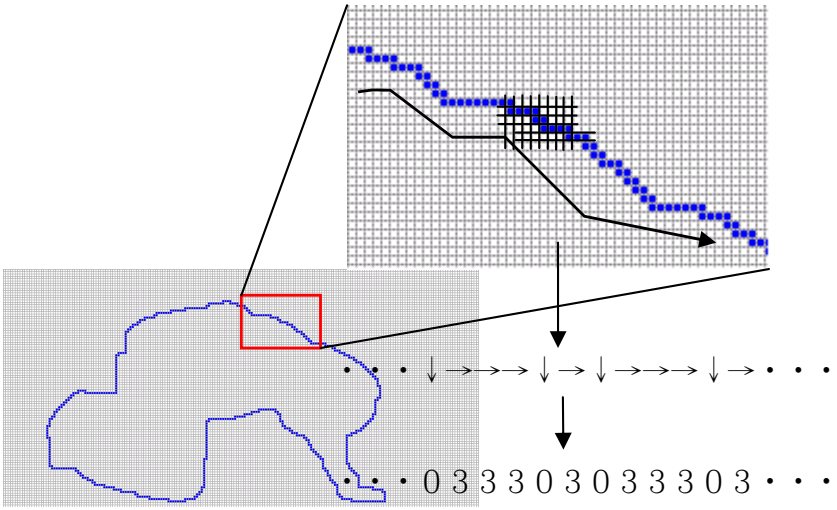


Figure 3. The retrieve of the human shape.

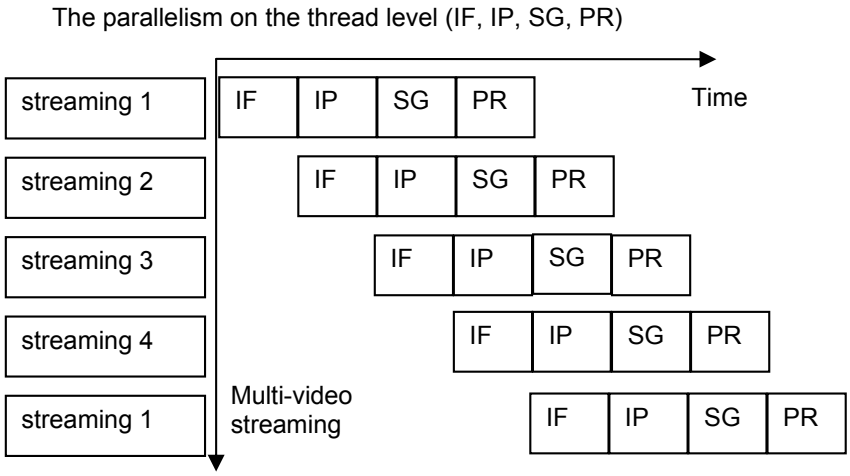


Figure 4. The pipeline of the multi-video-streaming processing.

Figure 4 shows how the pipeline flows as executing those four threads. The horizontal axis is the time and the vertical axis represents the incoming video streams. At first, the *IF* thread fetches and stores the image from the first video streams. Next, the *IP* thread performs the image processing on the image fetched by the *IF* thread. At the same time, the *IF* thread turns to fetch the image from the second video stream. After that, the *SG* thread generates the human-edge sequence for stream 1, the *IP* thread performs the image processing for stream 2, the *IF* thread performs the image fetch for stream 3, and so on. All of them communicate through an allocated shared memory, and synchronize by four status bits (like semaphores). Each status bit indicates that one thread has finished its current task. Only when the four threads have their works finished they can switch to next round again.

5. Experiment Results

All above algorithms have been implemented in a falling detection system for functionality proof. We evaluated the system in different places, including the corner of stairs and a long corridor, by the terms of the pipelining throughput, falling-detection precision rate, error rate, and delay time etc. The experiment methodology and the results are demonstrated as follows.

5.1. Experiment Methodology

The system works with four USB web-cameras (QuickCam Orbit AF by Logitech© [10]) to represent the multi-video streaming sources. The falling detection algorithm is implemented in C# using .Net framework 2.0, and the video image is captured via DirectShow [13]. The microprocessor in the system is a 4-core CPU with 2.0 GHz frequency, and 5GB main memory to perform all operations. We setup two cameras at a 100m corridor and other two cameras at the corner of stairs. The testers are standing, walking around, running, falling down, or bending down randomly. Each falling or bending posture will continue over 10 seconds to simulate the "real" falling accidents. (We suppose that a true falling accident required to be alarmed would be the case that the person cannot stand up from falling in 10 seconds.)

5.2. Results

A. Throughput

At first, we evaluate the effects of the pipelining processing on the system throughput. We compare three approaches:
(i) Non-pipelining: Apply four independent threads to each video stream; that is, each thread performs whole operations (from image fetch, processing, to pattern recognition etc) just for one video stream.
(ii) Unbalanced pipelining: Apply the pipelining technique to four threads as shown in Section 4, but the workloads of each stage are unbalanced.
(iii) Balanced pipelining: The same as (ii), but the workloads of each stage are almost equally balanced.

Figure 5 shows the speedup ratios of the FPS (frames per second) performed by these three

approaches. From the results we found that although these three approaches all executed in multithreading, without exploiting the software parallelism among the threads, the system cannot achieve better throughput.

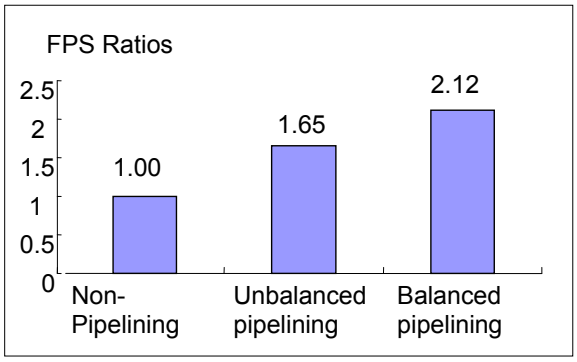


Figure 5. Effects of the pipelining processing.

We also measured the average processing time on each video frame by those three approaches, as shown in Figure 6. The results show that applying the pipelining technique would enlarge the processing time for each individual frame, especially with unbalanced pipelining, due to the communication and synchronization overheads. However, when we have many tasks (video frames) to do, the improvement of the throughput decreases the total time to complete the work. Figure 5 and Figure 6 conclude that the balanced pipelining approach suffers 1.5 times processing time on each video frame than the non-pipelining approach, but earns 2.12 times on throughput improvement.

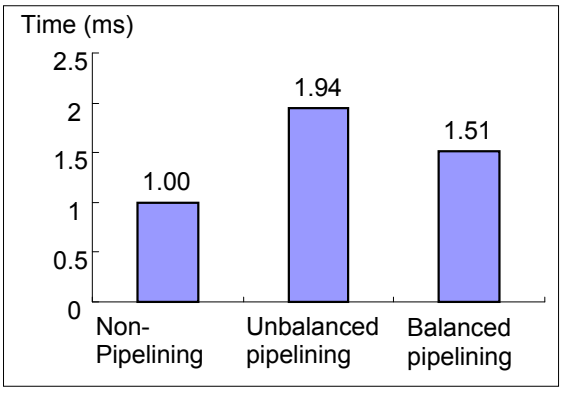


Figure 6. Average processing time of one video frame.

B. Precision rate and error rate

Next, we focus on the precision rate and the error rate performed by the proposed falling detection algorithm. We define the precision rate as the ratio of the cases in which the system issues an alarm if a falling accident indeed occurs and keeps silent otherwise, to the total testing cases. We also define the error rate as the ratio of the cases in which the system issues an alarm but no accidents occurs, to the total cases in which the system issues alarms. These two rates are quite different. The former measures the system's functionality, but the later measures the confidence when the system issues an alarm.

Table1 shows the average results when we tested the system at the corridor and at the corner of stairs, respectively. Each testing continued over 20 minutes, and constituted at least 100 postures changed randomly by the testers. From the results we found that the precision rates can achieve about 92% but the error rates only have 1%~3%. This shows that our system can be an effective falling-accident alarming system which assists to take care of the activity safety for elders.

Table 1. The precision rate and the error rate

	At the corridor	At the corner
Precision rate	92.3%	92.7%
Error rate	1.6%	3.5%

C. Delay time

Finally we measured the delay time of the system when detecting the fallings, which is defined from the time a falling accident occurs to the time the system detects and issues the alarm. Note that, in order to avoid too intensive and too sensitive alarms, the system would not issue the alarm immediately if the tester has been detected that his current posture just matches a falling pattern. Instead, the system will monitor the tester's current posture continuously at least 10 seconds. Only when the posture continuously matches the falling patterns in this time interval the system will then issue the alarm. Table 2 shows the average delay time evaluated. Without considering the delayed alarm just mentioned (in 10 seconds), the process from the image fetch to the pattern recognition requires about 2 to 3.1 seconds. This would be acceptable in case an elder falls and needs to inform someone as soon as possible.

Table 2. Delay time

	At the corridor	At the corner
Average Delay Time	12 (sec)	13.1 (sec)

6. Conclusion

In this paper we propose a pattern recognition-based solution to detect elder's falling accidents. In addition, we apply the pipelining and multithreading technique to improve the throughput when the system connects with multiple cameras and thus has multiple video streams. We implement the algorithms and apply the system to some high-risk places, e.g. the corridor or the corner of the stairs, to measure the functionality and the performance. Experiment results show that the system can precisely detect the tester's falling postures with only 1%~3% error rates, and the throughput (in FPS) can have obvious improvement (i.e., 2.12 times speedup).

7. References

[1] A. Sixsmith and N. Johnson, "A Smart Sensor to Detect the Falls of the Elderly," *IEEE Pervasive Computing*, Vol.3, No. 2, pp. 42-47, April 2004.

[2] Aaron F. Bobick and James W. Davis, "The recognition of human movement using temporal templates," *IEEE Transaction on Pattern Analysis And Machine Intelligence*, Vol. 23, No. 3, pp. 257-267, March 2001

[3] Caroline. Rougier, J. Meunier, A. St-Arnaud, and J. Rousseau, "Fall Detection from Human Shape and Motion History Using Video Surveillance," *International Conference on Advanced Information Networking and Applications Workshops*, Vol. 2, pp. 875-880, 2007.

[4] Eric Sven Ristad and Peter N. Yianilos, "Learning String-Edit Distance," *IEEE Transactions on Pattern Analysis And Machine Intelligence*, Vol. 20, No. 5, MAY 1998.

[5] Gonzalez, R. C. and Woods, R. E, *Digital Image Processing,* 2nd ed., Prentice Hall, Upper Saddle River, NJ, 2002.

[6] Hammadi Nait-charif and Stephen J. Mckenna, "Activity Summarisation and Fall Detection in a Supportive Home Environment," *In Proceedings of the 17th International Conference on Pattern Recognition*, Vol. 4, pp. 323-326, 2004.

[7] Hennessy, John L. et al., *Computer Architecture: A Quantitative Approach,* 3rd Ed. Morgan Kaufmann, 2002.

[8] J. Shou-Pyng Shu, "One-pixel-wide edge detection," *Pattern Recognition*, Vol. 22, No. 6, pp. 665-673, November, 1989.

[9] K. Kim, T.H. Chalidabhongse, D. Harwood and L. Davis, "Real-time foreground-background segmentation using codebook model," *Real-Time Imaging*, Vol. 11, No. 3, pp. 172–185, 2005.

[10] Logitech, http://www.logitech.com/

[11] P. Hall, and G. Dowling, "Approximate String Matching," *Computing Surveys*, vol. 12, No. 4, pp. 381-402, 1980.

[12] Vinay Vishwakarma, Chittaranjan A. Mandal, and Shamik Sural, "Automatic Detection of Human Fall in Video," *International Conference on Pattern Recognition and Machine Intelligence,* pp. 616-623, 2007.

[13] DirectShow, http://www.sourceforge.net/projects/directshownet/