A High Speed Feature Matching Architecture for Real-time Video Stabilization

Keng-Yen Huang, Yi-Min Tsai, Tien-Ju Yang, and Liang-Gee Chen
DSP/IC Lab, Graduate Institute of Electronic Engineering
National Taiwan University, Taiwan
{*kyhuang,ymtsai,denru,lgchen*}*@video.ee.ntu.edu.tw*

***Abstract*— An efficient feature matching architecture targets at real-time video stabilization is revealed in this paper. For some applications, such as vehicular application, real-time video stabilization is needed to provide instant stable video input. However, feature matching is usually the bottleneck to achieve high performance. High speed feature matching architecture is proposed to accelerate the performance of video stabilization. Locality sensitive hashing (LSH) helps us realize the feature matching procedure in hardware implementation. By applying the proposed dynamic table allocation and on-chip cache mechanism, this work achieves 422K queries/s and real-time feature matching with 90% in memory reduction and more than 50% in relieving the feature bus burden of the system.**

I. Introduction

Video camera becomes smaller and eventually portable in many applications. As a result, more smart video sensors will be placed around to bring about a more convenient life. However, the captured video may be blurred and shaky owing to unstable camera platforms. This not only results in uncomfortable watching experience but also introduces severe artifact to back-end applications. Therefore, many real-time applications need on-the-fly video stabilization to provide steady video inputs.

The basic idea of video stabilization is to correctly estimate the camera movement and to determine which part of the movement is undesired. Existing video stabilization algorithms are roughly classified into three major categories, special-image-structure-based, block-motion-based, and feature-based methods:

- **Special-image-structure-based method.** Utilizing the image structure or prior knowledge to fix the unstable video. This kind of idea has been commonly adopted in vehicular applications. Liang [1] proposed a stabilization mechanism based on fixing the position and slope of lane lines. However, the requirement for image clearance for traits makes this method hard to maintain robust performance under various scenarios.
- **Block-motion-based method.** This category is the most widely employed approach. Through dividing the image into blocks and calculating the block movements across consecutive frames, the image motion can be estimated [2], [3]. Generally speaking, large number of block motions guarantees a consistent stabilization result, but it usually implies time-consuming computation.
- **Feature-based method.** For reducing the weighty computation for calculating all block motion vectors, recent researches focus on interested points that are regarded as some corners or features. Feature motion vectors are calculated for global motion estimation [4], [5]. Feature-based methods guarantee a more reliable solution for video stabilization.

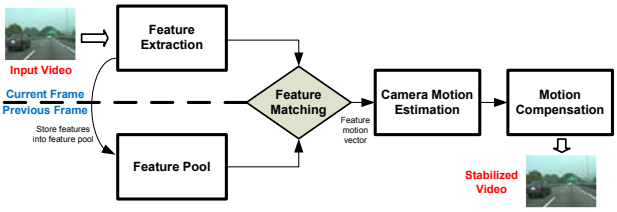

Fig. 1. Block diagram of the feature-based video stabilization system.

However, even with reduced interested feature points, current feature-based methods still can not meet real-time requirement through CPU implementation, especially for a high resolution video input. The bottleneck is the feature matching stage. In order to real-time video stabilization can be achieved, an efficient feature matching engine is required. Previous literature tries to adopt a processor or a RISC to accelerate the matching procedure [6], but the result shows that it can only support small number of features. In this paper, a high speed feature matching architecture is proposed. Firstly, optimized hardware for locality sensitive hashing (LSH) [7] reduces the computational complexity for matching features. Secondly, applying on-chip cache to minimize the bus usage. The result shows that our system provides fast feature matching flow and further assist real-time video stabilization under high video resolution.

This paper is organized as follows. The basic feature-based video stabilization algorithm is described in Sec. II. Feature matching method is deeply discussed in Sec. III Sec. IV reveals the proposed architecture. Architecture analysis is presented in Sec. V and Sec. VI concludes this work.

II. Feature-based Video Stabilization Algorithm

Based on the previous work [8], Fig. 1 illustrates the block diagram of the feature-based video stabilization system. Input video is first processed to extract those potentially interested feature points. Each feature is represented by a feature description. Secondly, we find the corresponding position for every extracted feature by probing the feature pool. The matched

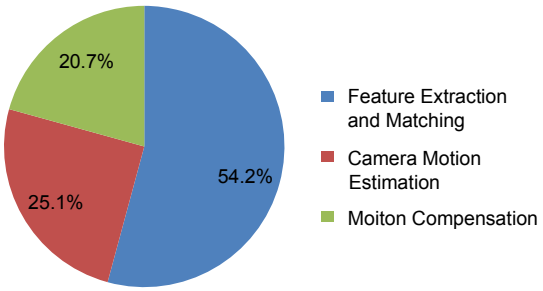

Fig. 2. Timing analysis for the feature-based video stabilization algorithm.

pairs are collected to estimate the camera motion and to compensate the unwanted camera movement.

From the timing analysis illustrated in Fig. 2, we observe that more than half of computation time is spent on feature extraction and feature matching. In the feature extraction and matching step, matching features is much more difficult when the feature pool is too large for a feature to find its corresponding point. This causes frequent memory access to the off-chip memory. Thus, we present a searching scheme for achieving efficient matching in a large feature pool.

III. Feature Matching via LSH

Most techniques adopt traversing the tree-structure feature pool such as KD-tree to search for the most possible feature candidates. However, when the feature dimension is above about 20, the tree based search methods do not yield any performance improvement over brute force methods [9], as in the case of our algorithm, a 64-dimension SURF descriptor.

Approximate methods, such as LSH, can dramatically reduce the computational complexity. The idea of LSH is shown in Fig. 3. Traditional hashing tries to spread out the features into buckets randomly to achieve that each bucket contains just a few features. In LSH, the basic motivation is to spread out in the buckets while preserving the spatial relationships between features. Through utilizing the locality information, we can search the corresponding feature in the feature pool by examining all its neighboring buckets.

In order to maintain the spatial information, given a feature vector V, the hashing function $h_{\mathbf{a},b}$ is depicted in Eq. 1:

$$h_{\mathbf{a},b}(V) = \lfloor \frac{\mathbf{a} \cdot V + b}{r} \rfloor \tag{1}$$

where $\mathbf{a}$ is a n dimensional vector with entries chosen independently from Stable distribution. The parameter b is a real number chosen uniformly from the range $[0, r]$, in which r represents the maximum value of a single hash index. The $h_{\mathbf{a},b}$ maps a n dimensional vector V onto a set of integers. Through selecting k sets of hashing functions, we can combine these single hash indexes into a final LSH index function g_i as shown in Eq. 2

$$g_i = \{h_{\mathbf{a_1},b_1}, h_{\mathbf{a_2},b_2}, \cdots, h_{\mathbf{a_k},b_k}\} \tag{2}$$

The resulting index function guarantees the probability of finding the most similar feature in the neighboring indexes

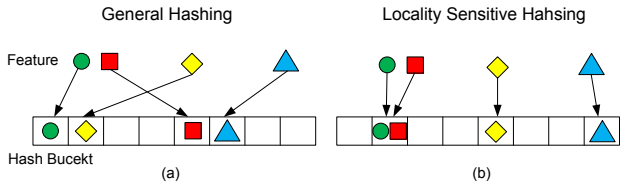

Fig. 3. Two hashing methods. (a) General hashing randomly select the assigned bucket. (b) Despite the random assigning, locality sensitive hashing preserves the spacial information.

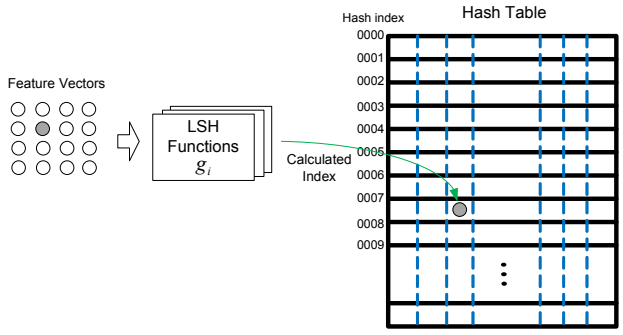

Fig. 4. For each hash table, there is an unique LSH function g_i to determine the bucket index for insertion.

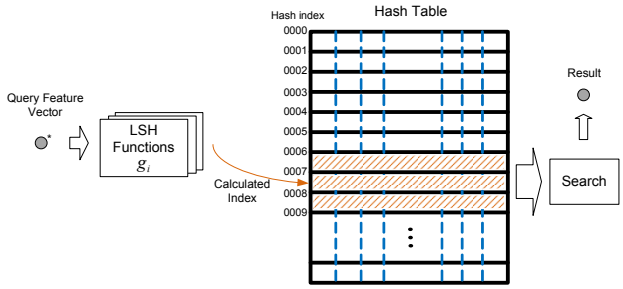

Fig. 5. During the feature matching step, a primary searching index is determined by the index function for each hash table. Then we probe the primary bucket and its neighboring buckets to achieve better feature matching accuracy.

is higher than ρ [7]. Fig. 4 indicates the way of inserting features into a LSH table. There are multiple LSH tables for feature insertion. Each LSH table bas its unique g_i, which helps to vary the insertion rules to achieve better performance. Generally speaking, more LSH tables would lead to higher accuracy of feature matching but more processing time and memory requirement. The feature searching process is shown in Fig. 5. After determining the hashing index, we search the corresponding bucket and its neighboring buckets.

IV. Proposed Architecture

The proposed architecture for feature matching is shown in Fig. 6. For matching features, the LSH index is assigned through index computation. It determine which addresses should be probed to find all possible feature candidates by fetching from the on-chip cache and the feature bus. The final matched result is outputted after sequentially comparing their feature descriptors. After finish feature matching, we store the

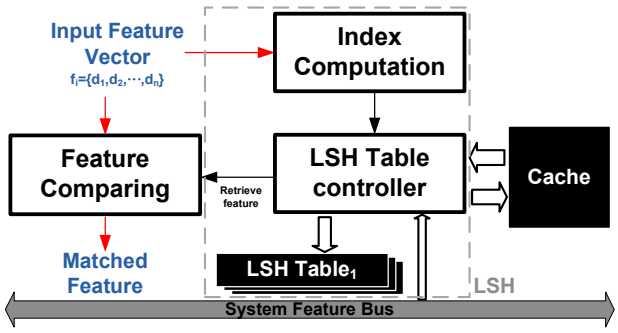

Fig. 6. The proposed architecture for feature matching.

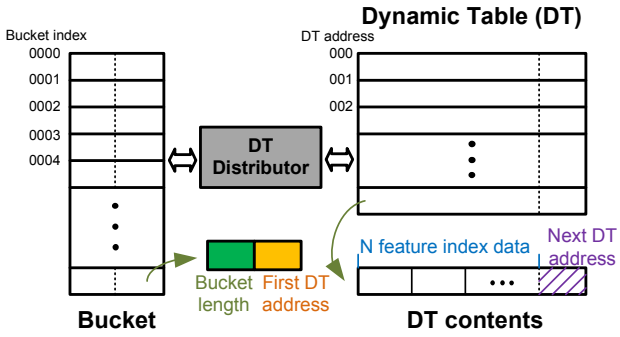

Fig. 7. Dynamic table structure.

newly collected feature into the LSH tables. Through the index calculation procedure, features can be inserted into the feature pool and waited to be compared by the features extracted from the next frame.

However, this process requires large memory space to construct LSH tables. Besides, limited bus bandwidth degrades the system performance while fetching the off-chip feature pool. Two major schemes, dynamic table allocation and cache-based feature searching, are brought up to solve the problems. The memory usage is significantly reduced and the processing time is largely accelerated. The following subsections describe the detailed architecture and operation.

A. Dynamic Table Allocation

In order to avoid huge memory allocation for constructing hash tables, we adopt the dynamic memory allocation in our architecture design. We cut the data space in each bucket into small pieces which are gathered together and controlled by a dynamic table distributor (DT distributor). The dynamic table structure is revealed in Fig. 7. The LSH table is composed of a DT distributor, a bucket, and a DT.

Each calculated LSH table index points to a bucket index. The bucket contains two information, the number of data stored in this bucket and the first memory address where we can find the feature data in DT. Assume that a new feature is going to be inserted into a certain bucket index, the bucket check whether there is enough space or not. If there is free space, the feature is inserted according to the calculated bucket index. Otherwise, the feature will be inserted after additional bucket memory allocated by the DT distributor. When the DT distributor receives a request, an empty DT slot would be released to the bucket. A DT slot describes N number of feature index data. The last part of the slot indicates the next DT slot if there is any descendant.

B. Cache-based Feature Matching

Owing to the bandwidth of the system bus, we need at least 18 cycles to complete a feature transmission to or from the off-chip memory. For each feature matching procedure, it might involve hundreds of features loading from off-chip memory and there are nearly a thousand features waiting to run the matching procedure. Fortunately, there is a high probability that different features may retrieve the same feature candidates from the feature pool. In order to prevent loading frequently-matched features several times from off-chip memory, we propose cache-based feature matching to reduce multiple off-chip access and relieve the burden of feature bus.

We adopt a 2-way set-associative cache to avoid retrieving redundant features. Each time we try to find the matched pair of a feature, the cache is searched first, the feature information, which is the feature descriptor, will be obtained in 2 cycles. This is much less than the cycles needed to access feature information from the off-chip memory. Increasing the hit rate can avoid the time needed to access the outside memory.

V. Analysis and Synthesis Result

A. Design Analysis

The proposed architecture is synthesized and analyzed in a 90nm CMOS process. We set the specification of the targeted video input is a 1280×960 video stream with 30 frames per second. Approximately 1000 features are extracted for each frame and stored in the feature pool. We refresh contents in the feature pool and match features every frame. The SURF feature adopted here needs 544 bits to represent its 64-dimension descriptor. In addition, the bandwidth of the feature bus is set to reasonable 32 bits.

There are two significant factors that determine the system performance. The first one is the length N of a slot in the dynamic table. The second one is the cache size S. Large cache size relieves the bus loading and compensates the penalty incurred by the cache miss. We examine the factors and decide the proper setting for the proposed architecture.

- **Dynamic table length, N.** Larger N means there are more data can be stored in one DT slot. However, if most of the buckets are empty or contain only a few data, large N would lead to excessive memory allocation as shown in Fig. 8(a). In our profiling, the memory usage meets the minimum when $N = 2$. Fig. 8(b) depicts that with increasing N, feature probing time can be largely reduced. Considering that we want to reduce the table memory, $N = 2$ is set in our architecture.
- **Cache size, S.** In order to avoid penalty due to cache miss, larger cache memory size would lead to higher hit rate as shown in Fig. 9(a). Without frequently accessing the off-chip memory, the total processing time to match all features is reduced as depicted in Fig. 9(b). Choosing large cache size

TABLE I

COMPARISON BETWEEN CURRENT FEATURE MATCHING METHOD AND THE PROPOSED FEATURE MATCHING ENGINE.

	ASSCC2010(KAIST)	Proposed feature matching engine
Table number	2	3
Hash function number	5	6
Feature descriptor type	SIFT(128)	SURF(64)
Main improved idea	Huffman coding	Dynamic table allocation, Cache
Operating frequency	200MHz	200MHz
Gate Count	255K	410K
Memory	64KB	39.6KB
Matching performance	43.2K queries/s	422K queries/s

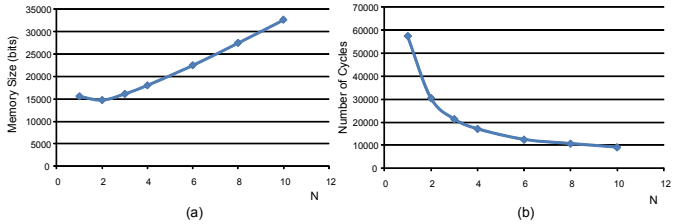

Fig. 8. (a) With the increasing of N, larger memory space is required to allocate enough space for DT. (b) The latency is reduced significantly by selecting larger N.

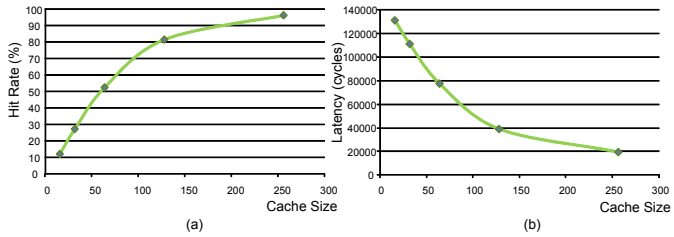

Fig. 9. (a) Hit rate increases by allocating large cache memory. (b) Latency reduction due to cache allocation.

remarkably accelerate the system performance, but memory usage comes as the cost. We set the baseline that the hit rate should be higher than 50% to reduce the potential cache miss punishment. Therefore, 64 entries 2-way set-associative cache is selected.

B. Synthesis Result

Combined with the two techniques mentioned above, the proposed architecture provides a high efficiency feature matching engine, which is the most critical part in the video stabilization system. Synthesis result reveals that the proposed architecture supports 422K feature queries per second. Compared with state-of-the-art feature matching architecture [10], the proposed feature matching architecture shows better performance as described in Table I. The proposed architecture supports more hash functions and hash tables with less on-chip memory. The matching performance is approximately 10x higher. With fast feature matching engine, the feature motion vectors can be calculated much faster and leads to overall system enhancement.

VI. CONCLUSION

A high performance feature matching architecture is proposed to accelerate the feature-based video stabilization system. Locality sensitive hashing is utilized to realize the feature matching procedure in hardware implementation. To cope with redundant memory usage and insufficient bandwidth for feature bus, dynamic hash table allocation and cache-based feature matching are employed to enhance system performance. By applying dynamic table allocation, the memory consumption is reduced by more than 90% compared to allocate full size of hash tables. 2-way set-associative cache is adopted to reduce frequent feature fetching from the off-chip memory. More than 50% reduction of the bandwidth requirement of feature bus is achieved. Through the proposed architecture, 422K feature queries/s allows the video stabilization to achieve real-time assistance even with high resolution video input.

REFERENCES

[1] Y. M. Liang, H. R. Tyan, S. L. CHang, H. Y. M. Liao, and S. W. Chen, *Video stabilization for camcorder mounted on a moving vehicle*, IEEE Transactions on Vehiclar Technology, vol. 53, pp. 1636-1647, Nov. 2004

[2] H. C. Chang, S. H. Lai, and K. R. Lu, *A robust real-time video stabilization algorithm*, Journal of Visual Communication and Image Representation, vol. 17, pp. 659-673, Jan. 2006.

[3] S. Manigiat and Y. J. Chiu, *Block based completion for video stabilization*, in Proc. of Asilomar Conference on Signals, Systems and Computers, 2010, pp. 222-255.

[4] C. Y. Chung and H. H. Chen, *Feature-based full-frame image stabilization*, in Proc. of International Symposium on Multimedia, 2007, pp. 100-106.

[5] Y. G. Ryu, H. C. Roh, and M. J. Chung, *Long-time video stabilization using point-feature trajectory smoothing*, In Proc. of IEEE International Conferene on Consumer Electronics (ICCE), 2011, pp. 189-190.

[6] S. Lee, J. Oh, J. Park, J. Kwon M. Kim, and H. J. Yoo, *A 345 mW heterogeneous many-core processor with an intelligent inference engine for robust object recognition*, IEEE Jounal of Solid-State Circuits, vol. 46, pp. 42-51, Jan. 2010.

[7] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, *Loaclity-sensitive hashing schemem based on p-stable distributions*, in Proc. of Symposium on Computational Geometry (SCG), 2004, pp. 253-261.

[8] K. Y. Huang, Y. M. Tsai, C. C. Tsai, and L. G. Chen, *Video stabilization for vehicular applications using SURF-like descriptor and KD-tree*, IEEE International Conference on Image Processing (ICIP), pp. 3517-3520, 2010.

[9] A. Gionis, P. Indyk, and R. Motwani, *Similarity search in high dimensions via hashing*, in Proc. of International Conference on Very Large Data Bases (VLDB), 1999, pp. 518-529.

[10] S. LEE, J. Kwon, J. Oh, J. Park, and H. J. Yoo, *A 92mW 76.8GOPS vector matching processor with parallel Huffman decoder and query reordering buffer for real-time object recognition*, IEEE Asian Solid-State Circuits Conference (ASSCC), pp. 1-4, 2010.