Real-Time Feature-Based Video Stabilization on FPGA

Jianan Li, Tingfa Xu, and Kun Zhang

Abstract—Digital video stabilization is an important video enhancement technology which aims to remove unwanted camera vibrations from video sequences. Trading off between stabilization performance and real-time hardware implementation feasibility, this paper presents a feature-based full-frame video stabilization method and a novel complete fully pipelined architectural design to implement it on Field-Programmable Gate Array (FPGA). In the proposed method, feature points are first extracted with the Oriented FAST and Rotated BRIEF (ORB) algorithm and matched between consecutive frames. Next, the matched point pairs are fitted to the affine transformation model using a Random Sample Consensus (RANSAC) based approach to estimate interframe motion robustly. Then the estimated results are accumulated to compute the cumulative motion parameters between the current and reference frames, and the translational components are smoothed by a Kalman filter representing intentional camera movement. Finally, a mosaicked image is constructed based on cumulative motion parameters using an image mosaicking technique, and then a display window is created with the desired frame size according to the computed intentional camera movement to obtain a full motion-compensated frame. Using pipelining and parallel processing strategies, the whole process has been designed using a novel complete fully pipelined architecture and implemented on Altera’s Cyclone III FPGA to build a real-time stabilization system. Experimental results have shown that the proposed system can deal with standard PAL video input including arbitrate translation and rotation and can produce full-frame stabilized output providing a better viewing experience at 22.37 milliseconds per frame, thus achieving real-time processing performance.

Index Terms—Feature extraction, FPGA, motion estimation, video stabilization.

I. INTRODUCTION

Cameras mounted on moving platforms usually suffer from undesired jitter due to platform vibration, making the captured video blurred and shaky, which may lead to an unpleasant viewing experience and affect the performance of video processing applications such as video surveillance [1] and video encoding [2]. Digital video stabilization is therefore becoming an essential technique to remove undesired image motion from the original input, producing a compensated video sequence with smooth global movements only.

Generally, digital video stabilization consists of three steps: motion estimation, motion compensation, and image composition [3]. Among these steps, motion estimation is the most time-consuming, but is also crucial to the accuracy of the whole method.

Existing motion estimation algorithms can be broadly classified as block-based [4] [5] [6] or feature-based [7] [8] [9] [10]. Block-based algorithms divide the image into blocks and match blocks between adjacent frames to obtain local motion vectors, which are used to find a global motion. These algorithms generally produce good results, but usually involve heavy computation for large numbers of block motions. The idea behind feature-based algorithms is to extract features from each frame and then to match features between adjacent frames to estimate motion. Feature-based approaches can provide accurate results with less computational load, which guarantees a more effective video stabilization solution.

Commonly used methods for feature detection and description such as Scale-Invariant Feature Transform (SIFT) [11] and Speeded-Up Robust Features (SURF) [12] have gained wide consensus because of their good performance. However, they are difficult to implement on embedded systems for real-time applications because of their high computational complexity. Oriented FAST and Rotated BRIEF (ORB), first presented by Rublee et al. [13] in 2011, is an efficient local feature descriptor that is rotation-invariant and resistant to noise. It was designed using the Features from Accelerated Segment Test (FAST) keypoint detector [14] and the Binary Robust Independent Elementary Features (BRIEF) descriptor [15], leading to a remarkable reduction in computational complexity and hardware cost. Moreover, it is very suitable for Field-Programmable Gate Array (FPGA) implementation because it consists mostly of basic comparison operations and has parallel computation capability.

This paper offers a good trade-off between algorithm performance and ease of hardware implementation and presents a feature-based full-frame video stabilization method implemented on FPGA using a novel complete fully pipelined architecture for real-time processing. First, feature points are extracted with the ORB algorithm and matched between adjacent frames according to the Hamming distance, followed by robust interframe motion estimation using the affine transformation model with an outlier rejection scheme based on the Random Sample Consensus (RANSAC) algorithm [16]. By cascading the estimated interframe motion, the cumulative motion parameters between the current and reference frames can be computed, which are then used to construct a mosaicked image using an image mosaicking technique [17] to avoid unfilled areas in the output frame caused by motion.
compensation. Then the translational components of the cumulative motion parameters are smoothed with a Kalman filter [18] representing intentional camera movement, which is taken to be the position of a display window with the desired frame size on the mosaicked image, to obtain a full motion-compensated frame.

Considering the power, space, and real-time performance constraints of embedded systems, the proposed process has been designed in a novel complete fully pipelined architecture using parallel processing techniques and implemented on Altera’s Cyclone III FPGA. The results obtained show that the FPGA-based video stabilization system is superior in stabilizing shaky and rotated videos with real-time processing capability.

The contributions of this paper can be summarized as follows. 1) Feature-based video stabilization can be very powerful in most applications. However, its high-performance processing requirements pose a difficult challenge for real-time operation. To the best of our knowledge, the proposed framework in this paper makes the first attempt to implement the whole feature-based video stabilization process on a single FPGA chip, achieving real-time performance. 2) A novel complete fully pipelined FPGA architecture has been first proposed to substantially accelerate feature-based video stabilization in a highly parallel manner, which also provides a reference to accelerating other feature-based video processing tasks such as object tracking and video fusion on FPGA. 3) Using the proposed framework, a real-time miniaturized video stabilization system with low power consumption can be built, which is particularly favorable to portable applications.

This paper is structured as follows. Related work on FPGA implementation of video stabilization is reviewed in Section II. Section III introduces the overall process of the proposed stabilization method. Section IV describes the technical details of the pipelining design for FPGA implementation. Section V discusses the experiments and the results of the proposed system. Conclusions and possible future research are presented in Section VI.

II. RELATED WORK

In recent years, numerous video stabilization techniques have been developed [7] [8] [9]. These methods generally produce good stabilization results, but few of them can achieve real-time performance for practical video stabilization in portable applications due to their high computational complexity. FPGAs provide an effective solution to accelerating real-time video processing because of the parallel computing capability. However, the implementation of complex algorithms on FPGA has always been a difficult challenge. Therefore, a trade-off must be made between algorithm performance and ease of hardware implementation.

So far, several approaches have been presented to implement video stabilization on FPGA [19] [20] [21]. Araneda et al. [19] used gray-scale projection to estimate the translational motion between consecutive frames and produced low-resolution output as a result of motion compensation. Yabuki et al. [20] divided the frame into several image blocks and obtained the global motion vector by tracking each image block through template matching. Thus the correction range was limited by the search window size. Similarly, multiple bit-plane matching was adopted by Li [21] to compute local motion vectors. Then the global motion vector was generated using a median filter approach. All of these methods are intensity-based, which are easy for FPGA implementation and can achieve real-time performance. However, they are only applicable to videos with translational jitter, which is insufficient in most applications where rotation and scaling must be considered. The correction range is also fixed due to a limited search area. Moreover, no video completion operations have been conducted to deal with the unfilled areas in the output frame caused by motion compensation, leading to results with an unpleasant viewing experience.

The proposed system in this paper differs from the above ones in several respects. 1) The feature-based method has been used to estimate the affine parameters between consecutive frames, which can deal with not only translational but also rotational and scaling jitter in the video. 2) The proposed system can handle large-scale shake over an unlimited search area. 3) Using the image mosaicking technique, the proposed system can produce full-frame stabilized output, offering a better viewing experience. 4) Benefiting from the proposed novel complete fully pipelined FPGA architecture, the whole stabilization process has been substantially accelerated in a highly parallel manner, thus achieving real-time performance.

III. PROPOSED STABILIZATION ALGORITHM

The proposed stabilization algorithm consists of five major procedures: feature detection and description, feature matching, motion estimation, image mosaicking, and motion filtering.

A. Feature Detection and Description

First of all, the corners of each frame are extracted using the ORB algorithm [13], which is an efficient feature descriptor composed of a FAST corner detector and an oriented BRIEF descriptor.

1) FAST Detector: The FAST detector [14] determines whether a pixel is a corner by comparing its intensity value with those of surrounding pixels in a circular ring. For a given intensity threshold \( t \), pixels around the center point can have one of three states:

\[
\begin{align*}
I_i & \leq I_p - t \quad \text{(Darker)} \\
I_p - t < I_i < I_p + t \quad \text{(Similar)} \\
I_i & \geq I_p + t \quad \text{(Brighter)}
\end{align*}
\]  

where \( I_p \) and \( I_i \) denote the intensity value of the center point and of a surrounding pixel, respectively. Pixel \( p \) is classified as a corner if \( N \) contiguous pixels in the circular ring are simultaneously darker or simultaneously brighter than the center point by threshold \( t \), as illustrated in Fig. 1. \( N \) was chosen to be 9, which has been proven to give good performance.

2) Orientation: The orientation of the detected corner is determined by Rosin’s corner intensity [22]. The moment of a patch is defined as:
where \( I(x,y) \) denotes the intensity value of a pixel in the patch and \( x \) and \( y \) represent the coordinate offsets from the patch center. Then the centroid is determined as:

\[
C = \left( \frac{m_{00}}{m_{00}}, \frac{m_{01}}{m_{00}} \right).
\]

The corner orientation is the angle of the vector from the patch center to the centroid:

\[
\theta = \text{atan2}(m_{01}, m_{10}).
\]

In the proposed method, the patch is taken as a circular region of radius 3 to improve the rotation invariance of this measure. The orientation is quantized to increments of \( 2\pi/32 \) (11.25 degrees), and a lookup table of orientation values is created based on the numerical relationship between \( m_{01} \) and \( m_{10} \) to facilitate implementation on embedded systems.

3) Feature Descriptor: The ORB algorithm generates a rotationally invariant feature description by submitting the corner orientation to the BRIEF algorithm. The BRIEF algorithm [15] constructs a bit string description of an image patch through a set of binary intensity tests. The ORB algorithm chooses 256 test point pairs with a Gaussian distribution in a \( 31 \times 31 \) pixel around the corner, where each test point is a \( 5 \times 5 \) patch for better noise immunity. For any feature set of \( n \) binary intensity tests at location \((x_0, y_0)\), the locations of the test point pairs can be expressed as a \( 2 \times n \) matrix:

\[
S = \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{bmatrix}.
\]

By multiplying these locations by the rotation matrix \( R_\theta \), the test point pairs can be rotated around the patch center according to the corner orientation:

\[
S_\theta = R_\theta S.
\]

Then the rotated version \( S_\theta \) of \( S \) is used for binary testing, producing a rotationally invariant description.

In the proposed method, the locations of the 256 test point pairs, as well as their rotated versions corresponding to each orientation from 1 to 32, are computed previously as a lookup table for ease of implementation on embedded systems.

B. Feature Matching

The feature matching step finds matched point pairs between adjacent frames based on the similarity of the detected corners, which can be measured using the Hamming distance. For each detected corner in the current frame, the candidate match with the minimum distance in the previous frame can be found. Meanwhile, the match with the second-minimum distance is also recorded. Then the ratio between the minimum and the second-minimum distances is checked against a preset threshold [11]:

\[
\frac{\text{Minimum Distance}}{\text{Second-Minimum Distance}} < \text{Threshold}.
\]

Here, the threshold was set equal to 0.8. If the condition is met, the candidate match with the minimum distance is considered good. This method functions well because correct matches are supposed to have lower ratios, whereas wrong matches are supposed to have ratios close to one.

The proposed method contains another strategy for further removal of potential wrong matches. Because small translations and rotations occur between adjacent frames, correct matches are supposed to have similar locations and orientations. Based on this, the coordinate difference and the orientation difference of the matches obtained in the previous step can be checked to remove potential wrong matches.

Suppose that the coordinates of the matches are \((X, Y)\) and \((X', Y')\) and the orientations are \(R\) and \(R'\). The coordinate difference of the matches can then be checked as:

\[
\sqrt{(X - X')^2 + (Y - Y')^2} < \text{Threshold}_1.
\]

Using a similar procedure for the orientation difference leads to:

\[
|R - R'| < \text{Threshold}_2.
\]

Here, \(\text{Threshold}_1\) and \(\text{Threshold}_2\) were set to 40 and 1, respectively.

C. Motion Estimation

Once the matched point pairs have been obtained, the motion estimation step is performed to estimate interframe motion.

To describe the geometric transformation between two consecutive frames, a six-parameter 2D affine model was used in this research, which trades off model stability against representation capacity. The affine model can describe translation, rotation, scaling, and skew of images, which is enough for most video stabilization applications.

Three point pairs are necessary to compute the affine parameters. Selection of point pairs that are incorrectly matched or related to moving objects in the scene, here called outliers, may lead to improper motion estimation. Because three points form a triangle, the three point pairs selected for affine parameter estimation may form two triangles in adjacent frames [23], as shown in Fig. 2. Suppose that ABC and A'B'C' are the triangles formed in the previous frame and the current frame, respectively. The area difference between the two triangles can be checked to reject outliers for affine parameter estimation based on the fact that three correctly matched point pairs in the scene background have similar triangle areas in adjacent frames.

The proposed method includes a robust motion estimation
approach based on the triangle area matching technique and the RANSAC algorithm [16]. The procedure is as follows:

1. Randomly select three point pairs from those produced in the feature matching step.
2. If the selected point pairs are not collinear and the area difference between the two triangles formed in the adjacent frames is less than a given threshold, go to (3), else go back to (1).
3. Compute the affine parameters using the selected point pairs.
4. Calculate the geometric distance error for each point pair produced in the feature matching step using the computed affine model, and check the number of inliers for which the geometric distance error is less than a given threshold.

Repeat (1)-(4) for a number of trials and choose the affine model with the most inliers as the result.

Furthermore, to improve the robustness of the approach, a numerical threshold test of the final estimated affine parameters is performed to reject obvious estimation errors.

D. Full-Frame Stabilized Output

To produce a full-frame stabilized video sequence, the proposed method uses the image mosaicking technique and a Kalman filter.

1) Image Mosaicking: In the previous steps, motion parameters between consecutive frames have been estimated. By cascading these interframe motion parameters, the cumulative motion parameters between the current frame and the selected reference frame can be calculated. Assuming that the first frame is the reference frame and the $N$th frame is the current frame, the cumulative motion parameters between them can be computed as:

$$P_{N-1} = P_{2-1}P_{3-2} \cdots P_{N-N-1},$$

where $P_{N,N}$ denotes the estimated interframe parameters between frames $N$ and $N-1$.

Simple motion compensation by transforming the current frame using the computed cumulative motion parameters may result in unfilled areas in the output frame. To reconstruct the unfilled areas for a better viewing experience, the image mosaicking technique [17] is used to align the current frame onto the reference frame by means of the cumulative motion parameters. Then a display window with the desired frame size is set onto the mosaicked image to obtain a full motion-compensated output, as illustrated in Fig. 3.

2) Motion Filtering: Camera movement includes intentional camera movement and random jitter, which can be regarded as low-frequency signal and high-frequency noise, respectively. Generally, intentional camera movement contains only translational motion. To remove undesired noise due to jitter while preserving intentional camera movement, a low-pass filter can be used.

In the proposed method, the translational components of the cumulative motion parameters against frame number, referred to as $(X(n), Y(n))$, which can be considered as the original absolute position of the display window on the mosaicked image, are low-pass filtered by the Kalman filter [18] to remove undesired noise due to jitter. The filtered result with intentional camera movement only can be used as the final absolute position of the display window to produce a full-frame stabilized video sequence.

IV. FPGA IMPLEMENTATION OF STABILIZATION SYSTEM

FPGAs are a good solution to dealing with mass computation for real-time video processing using parallel and pipelined architectures. To implement the proposed stabilization method on FPGA, its effectiveness was first validated using Visual Studio 2010 (VS). Then, by means of spatial and temporal parallelization strategies, the proposed method was designed in a novel fully pipelined structure and implemented on FPGA.

Using the designed novel complete FPGA architecture, the whole stabilization process can be substantially accelerated in a highly parallel manner. A block diagram of the system architecture is given in Fig. 4.

The basic system structure consists of two parts. One part is the underlying data flow architecture with an interface for the video processing module, which controls the system data flow and constructs mosaicked output frames based on the parameters provided by the processing module. The second is the video processing module with stabilization algorithm, which computes the cumulative motion parameters between the current and reference frames as well as the stabilized position of the display window on the mosaicked image.

A. Underlying Data Flow Architecture Design

The underlying data flow architecture provides the pixel intensity values of the input frames to the processing module and uses the processed results to construct mosaicked frames for output. A block diagram of the architecture is given in Fig. 5.

In general, three input frame buffers and three output frame buffers are distributed in an external DDR2 SDRAM for a Ping-Pong scheme. First, the video stream in ITU-R BT.656 format with $720 \times 576$ resolution at a refresh rate of 25 frames per second is sent to the FPGA. The Video Decoder module decodes the input video stream and extracts active pixel intensity values, which are written to the input frame buffers through the Input Video Write module. Next, the buffered input frames are sent to the Processing module pixel by pixel in order through the Input Frame Read module to compute the cumulative affine parameters between the current and reference frames, as well as the stabilized position of the display window. Meanwhile, the luminance components of the buffered input frames are written to the reference frame buffer (for the first frame) and the current frame buffer (for subsequent frames) through the Current Frame Read module. Then, by aligning the current frame onto the reference frame and setting a stabilized display window onto the mosaicked image based on the results provided by the Processing module, the Address Generator module computes the memory address of each pixel in the display window in order, producing a memory address queue pointing to the Reference Buffer and the Current Buffer for construction of the output frame. Next, the Constructed Frame Read module reads the pixel data of the constructed frame from
memory according to the memory address queue. These data are then written to the output frame buffers through the Output Frame Write module. Finally, the buffered output frames are sent to the Video Encoder module through the Output Video Read module to reconstruct the video stream in ITU-R BT656 format for output.

Because several modules have read or write access to the DDR2 SDRAM, bus conflicts caused by simultaneous access to memory may happen. Therefore, a DDR2 Bus Arbiter module has been implemented to carry out bus assignment according to the preset access priority of each module, which is necessary to ensure the normal operation of the underlying data flow architecture. The DDR2 Controller module generates the necessary signals for accessing the DDR2 SDRAM according to read or write commands from the DDR2 Bus Arbiter module, which is an intellectual property (IP) core provided by Altera (the FPGA vendor).

B. Stabilization Algorithm Implementation

The stabilization processing module receives valid pixel data from the underlying data flow architecture and executes the video stabilization algorithm. This module consists of five submodules: Feature Detection and Description, Feature Matching, Affine Parameter Estimation, Cumulative Affine Parameter Calculator, and Kalman Filter. These modules are designed as a fully pipelined structure that can run simultaneously in parallel.

The Feature Detection and Description module extracts feature points from every frame and saves them into FIFO. If there are newly detected features in FIFO that have not been matched, the Feature Matching module matches them with the features of the previous frame, producing matched point pairs which are stored into a two-port RAM. The Affine Parameter Estimation module fits the existing matched point pairs in RAM to the affine transformation models round by round and retains the one with the most inliers for each round to estimate the optimum interframe motion parameters. These three modules run in parallel until processing of one frame is complete. Then the estimated interframe affine parameters are accumulated by the Cumulative Affine Parameter Calculator module, producing the final cumulative affine parameters between the current and reference frames. These cumulative parameters are then smoothed by the Kalman Filter module to obtain the stabilized position of the display window. All this information is used by the underlying data flow.

To ensure the accuracy of the FPGA implementation, every Verilog computation module has a corresponding VS version. Both random and targeted inputs are used to test each Verilog computation module in a simulator, and the results are compared to those from the VS version to verify the accuracy of the FPGA implementation.

1) Feature Detection and Description: The Feature Detection and Description module detects and describes the corners in every frame. It is made up of four submodules: Data Distribution, FAST Detector, Orientation Calculator, and Feature Descriptor. A functional block diagram is shown in Fig. 6.

The grayscale values of the input frame are loaded pixel by pixel from the processing interface. The Data Distribution module buffers the pixels in a $7 \times 7$ subwindow and the $5 \times 5$ patches in a $31 \times 31$ subwindow around each point and distributes them to the subsequent modules in parallel according to their respective needs. The surrounding pixels in the circular ring and the $7 \times 7$ subwindow are sent to the FAST Detector module and the Orientation Calculator module, respectively to determine whether the current point is a corner and to calculate its orientation. The surrounding $5 \times 5$ patches in the $31 \times 31$ subwindow are sent to the Feature Descriptor module to generate a description of the current point based on the computed orientation. Moreover, the coordinates of the current point are computed by a pixel counter in the Data Distribution module. Timing adjustment among these modules ensures that the output results from each module correspond to the same point.

As a result, a flag bit (active high) from the FAST Detector module is output as an indicator of the detected corner. The orientation (6 bits) computed by the Orientation Calculator module, the binary descriptor (256 bits) produced by the Feature Descriptor module, and the coordinates (20 bits) from the Data Distribution module are concatenated together into a feature vector (282 bits), which is saved into FIFO if the flag bit is active high. The results obtained from the Altera “Signal Tap” tool are shown in Fig. 7.

2) Feature Matching: The Feature Matching module stores the feature vectors from the Feature Detection and Description module and matches them with those of the previous frame, producing matched coordinate pairs. This module is composed of three submodules: FIFO Switcher, Matching Controller, and Matcher. A functional block diagram is shown in Fig. 8.

a) FIFO Switcher: The FIFO Switcher module consists of three FIFOs, among which FIFO Reference stores the feature vectors of the previous frame whereas FIFO Current and FIFO Backup store those of the current frame before and after matching, respectively. The newly produced feature vectors from the Feature Detection and Description module are first written to FIFO Current to be matched with those in FIFO Reference and are then written to FIFO Backup as reference vectors for the matching process of the next frame.

b) Matching Controller: The Matching Controller module controls the matching process to perform one round of matching for each feature vector in FIFO Current. If there are unmatched feature vectors in FIFO Current, the Matching Controller module reads one of these at a time, writes it to FIFO Backup, and sends it to the Matcher module along with a start signal to start a round of matching with the feature vectors in FIFO Reference.

c) Matcher: The Matcher module is responsible for finding the best match for the input feature vector among those in FIFO Reference. Once the active input start signal has been detected, the Matcher module registers the input feature vector to be matched from the Matching Controller module. Then it reads all the feature vectors in FIFO Reference one by one, writing them back simultaneously for further use in the next matching round, to calculate the Hamming distance from the registered feature vector using the Hamming Distance Calculator module. The computed distances are sent to the Min Distance Selector module to select the minimum and second-minimum distances. The point pair with the minimum
distance is selected as the candidate match, and its coordinate and orientation pairs are recorded. Then the ratio between the minimum and the second-minimum distances, as well as the coordinate and orientation differences of the candidate match, are checked against preset thresholds. If all the conditions are met, the minimum-distance candidate match is identified as the best match, and its coordinate pair is output with a matched flag (active high) signal.

If the feature detection process has been finished for the current frame and there are no unmatched feature vectors left in FIFO Current, a Matching Done flag (active high) is produced by the Matching Controller module, marking completion of the matching process for the current frame. Meanwhile, FIFO Reference is cleared, and FIFO Reference switches roles with FIFO Backup for the matching process of the next frame.

3) Affine Parameter Estimation: The Affine Parameter Estimation module estimates the affine transformation parameters between adjacent frames from the matched point pairs produced by the Feature Matching module. It consists of three submodules: Point Pair Storage, Affine Model Generator, and Optimal Model Voting. A block diagram is shown in Fig. 9.

a) Point Pair Storage: The Point Pair Storage module stores the matched point pairs from the Feature Matching module. The newly generated point pairs are saved into a two-port RAM under the control of the Write Controller module. The number of currently stored point pairs as well as the read port of the RAM are output for subsequent operations.

b) Affine Model Generator: Once the number of point pairs stored in RAM is greater than a preset threshold, the Affine Model Generator module begins operation. First, the Trio-Point Pair Selector module randomly selects three-point pairs from those in RAM, using collinear and area similarity tests to reject point pair trios that fall in a straight line or on moving objects. Then the selected trio of point pairs is sent to the Affine Parameter Calculator module to generate an affine model, which is saved into FIFO Model. These two steps are performed four times in a row to generate four affine models at one time.

c) Optimal Model Voting: The Optimal Model Voting module counts the number of inliers of the five candidate affine models in one estimation round and retains the one with the most inliers as the currently optimal model to participate in the next estimation round. First, the four affine models generated by the Affine Model Generator are loaded from FIFO Model and are stored in registers along with the model with the most inliers from the previous estimation round, giving five candidate affine models in the current estimation round. Then the Optimal Model Voting module reads all the point pairs stored in RAM one by one. Five Inlier Judge modules corresponding to the five candidate models are used to judge whether the present point pair read from RAM is an inlier of each model. Thus, the number of inliers for each candidate model can be counted, and these counts are sent to the Most Votes Model Selector module to select the currently optimal model with the most inliers as a candidate for the next estimation round.

The model generation and voting processes are performed round after round until the active Matching Done Flag signal from the Feature Matching module is detected. Then the optimal affine model from the latest estimation round that passes a numerical threshold test is output as the final estimated affine transformation between the adjacent frames. Meanwhile, a Motion Estimation Done Flag signal is output, marking the completion of the motion estimation process for the current frame.

4) Cumulative Affine Parameter Calculator and Kalman Filter: The Cumulative Affine Parameter Calculator module cascades the estimated affine parameters between consecutive frames from the Affine Parameter Estimation module to calculate the cumulative affine parameters between the current and reference frames. Then the translational components of the computed cumulative affine parameters are low-pass filtered by the Kalman Filter module to remove noise due to jitter, producing the stabilized absolute position of the display window. A block diagram is shown in Fig. 10.

Both the cumulative affine parameters and the position of the display window are sent to the underlying data flow architecture to construct full-frame stabilized output frames.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

The proposed method was implemented on Altera’s EP3C120F780 FPGA which is an ideal low-power video processing platform with large quantities of multipliers and M9K memory blocks and thus offers a good solution to achieving the miniaturized system with low power consumption. External DDR2 SDRAM and PAL video decoder/encoder chips were also necessary to build the whole system. Thus a real hardware platform was realized.

The Peak Signal-to-Noise Ratio (PSNR) was used to evaluate the effectiveness of the stabilization method. The PSNR between consecutive frames is defined as:

\[
PSNR(I_1, I_0) = 10 \log \frac{I_{\text{max}}^2}{MSE(I_1, I_0)},
\]  
(11)

where \(I_{\text{max}}\) is the maximum intensity value of a pixel and \(MSE(I_1, I_0)\) is the mean square error between consecutive frames, which is defined as:

\[
MSE(I_1, I_0) = \frac{1}{(N \times M)} \sum_{i=1}^{N} \sum_{j=1}^{M} (I_1(i, j) - I_0(i, j))^2,
\]  
(12)

where \(N\) and \(M\) represent the frame dimensions. PSNR measures the similarity between frames. Because the stabilized sequence has more continuous frames than the original jittered one, the PSNR of the stabilized sequence should be higher than that of the original sequence.

ITF is the average value of PSNR, which can be used to evaluate numerically how much a sequence has been stabilized. It is defined as:

\[
ITF = \frac{1}{N_{\text{frame}} - 1} \sum_{k=1}^{N_{\text{frame}} - 1} PSNR(k),
\]  
(13)
where $N_{\text{frame}}$ denotes the number of frames. Higher ITF values represent higher effectiveness of the stabilization method.

For purposes of this research, a measurement environment was set up, as shown in Fig. 11. The camera provides standard PAL video with translational and rotational jitter to the FPGA-based stabilization system. The system executes the stabilization algorithm and sends stabilized PAL video output to a monitor for observation. Meanwhile, both the original and stabilized videos are sent to the computer through video splitters for video capture.

**B. Experimental results and discussion**

For fair stabilization performance comparison with other methods, the proposed algorithm was first evaluated on the VS against 10 publically available video sequences covering different types of scenes. For illustration purpose, the results for two representative challenging sequences including fast large-scale shake and moving objects, respectively, are reported. Both sequences are publically available online at [24]. Some frames of them are shown in Fig. 12.

The stabilization performance of the proposed method was compared to that of a traditional video stabilization program, Deshaker 3.1 [25] on the above video sequences which were scaled to $640 \times 480$ resolution. 200 frames of each sequence were analyzed. Fig. 13 gives the comparisons between the original PSNR and the stabilized PSNR resulted by the proposed algorithm and Deshaker. Correspondingly, the ITF values for the original and stabilized video sequences are also given. As shown in Table I, substantial improvements in ITF were obtained, confirming the effectiveness of the proposed algorithm under challenging conditions. Moreover, the proposed method outperformed Deshaker, with ITF increments of approximately 45-50% compared to 27-35%.

To evaluate the real-time performance of the proposed system, standard PAL video with $720 \times 576$ resolution at a frame rate of 25 fps was used as input. A time counter located in the FPGA code was used to measure the computation time of one frame. For comparison, the computation time of the VS implementation using the default compilation options on a 2.93 GHz Intel Core i3 Quad was also measured. The results are shown in Table II. The average computation time per frame on the FPGA at a clock frequency of 80MHz was 22.37 milliseconds compared to 116.74 milliseconds on the VS, making the proposed implementation more than four times faster, which confirms that the proposed fully pipelined FPGA architecture can substantially accelerate the feature-based stabilization method. Moreover, the results show that the system can complete the operation on one frame within a frame duration of 40.00 milliseconds, which meets the requirements of real-time processing.

In addition, the real-time performance of the proposed system was also compared to that of the intensity-based systems presented by Yabuki et al. [20] and Li [21], as illustrated in Table III. The results show that although the proposed system uses the feature-based method with much stronger stabilization capability and higher computational complexity, it can perform as well as or better than the intensity-based systems in terms of real-time performance, which also confirms the effectiveness of the proposed fully pipelined FPGA implementation.

Fig. 14 shows the final stabilization results of the system. The tenth, fifteenth, and twentieth frames of the original and stabilized video sequences as captured by the computer are extracted and shown. The upper row shows the original frame sequences; the lower row shows the mosaicked and stabilized results from the system. The results show that the system effectively removes translational and rotational camera jitter from the original video sequence. Benefiting from the rotation-invariant feature-based method, the system can stabilize input videos with not only large-scale translational but also rotational shake over an unlimited search area. Moreover, the image mosaicking technique properly reconstructs unfilled areas resulting from motion compensation, making the stabilized videos look natural and coherent.

Table IV shows the FPGA resources utilization of the proposed system. It was also compared to that of the system presented by Li [21], as illustrated in Table V. The results show that the proposed system is more expensive in terms of FPGA resources utilization than the system presented by Li [21]. The causes of this mainly lie in two respects. On the one hand, the proposed feature-based method includes more complex processing procedures than the intensity-based method adopted by Li [21], which leads to more FPGA resources consumption while bringing much stronger stabilization capability. On the other hand, to guarantee the accuracy of the FPGA implementation, the proposed feature-based method was first simulated on the VS using single-precision floating-point arithmetic. At the FPGA implementation stage, for ease of checking the results of each processing step obtained from the FPGA against those obtained from the VS to prove implementation correctness, the same single-precision floating-point arithmetic has simply been implemented on FPGA, which is particular expensive in terms of logic resources usage. To reduce the FPGA resources utilization, the fix-point representation of a floating-point number on FPGA can be considered instead.

Finally, the proposed framework can be feasible for high-definition videos mainly through the following modifications. 1) Adjust the video decoder/encoder modules according to the encoding format of the high-definition video to be processed. 2) Resize the frame buffers in the external DDR2 SDRAM according to the frame size of the video. 3) Resize the line FIFOs in the modules that have read or write access to the DDR2 SDRAM according to the line size of the video frame. Moreover, additional FPGA timing optimization can also be considered for real-time high-definition video processing.

**VI. CONCLUSION**

In this paper, a feature-based stabilization method and its FPGA implementation using a designed novel complete fully pipelined hardware architecture have been presented. Using an ORB feature descriptor and robust RANSAC-based motion estimation, the method is applicable to videos with arbitrary translational and rotational jitter and has a certain robustness to moving objects in the foreground. In addition, the image mosaicking technique in conjunction with a Kalman filter produces full-frame stabilized output, yielding results which
provide a better viewing experience. Experiments on real video data have been conducted to verify the effectiveness of the proposed system. The observed results demonstrate that the proposed fully pipelined FPGA architecture substantially accelerates the feature-based stabilization method in a highly parallel manner and that the proposed system is attractive for practical video stabilization with real-time requirements. Future work will be devoted to implementing the proposed framework on high-definition videos. Moreover, because the system is based on an FPGA and a few peripherals, it can achieve miniaturization with low power consumption, and therefore applications on moving platforms will be considered.

REFERENCES

Fig. 1. Pixel allocation around the center point.

Fig. 2. Triangles formed by three selected point pairs.

Fig. 3. Output frame based on mosaicked image.
Fig. 4. Overview of system architecture.

Fig. 5. Underlying data flow architecture.
Fig. 6. Feature Detection and Description module.

Fig. 7. Feature description results of FPGA implementation.
Fig. 8. Feature Matching module.
Fig. 9. Affine Parameter Estimation module.

Fig. 10. Cumulative Affine Parameter Calculator and Kalman Filter modules.
Fig. 11. Measurement environment diagram.

(a)

(b)

Fig. 12. Frames of test video sequences.
Fig. 13. PSNR of original and stabilized test video sequences.
Fig. 14. Original and stabilized video frames of the system.
<table>
<thead>
<tr>
<th>Video sequence</th>
<th>Original (dB)</th>
<th>Deshaker (dB)</th>
<th>Proposed (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>19.58</td>
<td>24.85</td>
<td>29.66</td>
</tr>
<tr>
<td>b</td>
<td>15.15</td>
<td>20.48</td>
<td>22.13</td>
</tr>
</tbody>
</table>

**TABLE II**

**Comparison of Computation Times**

<table>
<thead>
<tr>
<th>Frame Duration</th>
<th>Computation Time on VS</th>
<th>Computation Time on FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>40.00 ms</td>
<td>116.74 ms</td>
<td>22.37 ms</td>
</tr>
</tbody>
</table>

**TABLE III**

**Comparison of Real-Time Performance**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Resolution (pixels)</th>
<th>Frame Rate (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yabuki et al. [20]</td>
<td>542×496</td>
<td>30</td>
</tr>
<tr>
<td>Li [21]</td>
<td>720×576</td>
<td>25</td>
</tr>
<tr>
<td>Proposed</td>
<td>720×576</td>
<td>25</td>
</tr>
</tbody>
</table>

**TABLE IV**

**Statistics of FPGA Resources Utilization**

<table>
<thead>
<tr>
<th>Resources</th>
<th>Utilized</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Elements</td>
<td>79,264</td>
<td>119,088</td>
<td>67%</td>
</tr>
<tr>
<td>Registers</td>
<td>33,174</td>
<td>121,673</td>
<td>27%</td>
</tr>
<tr>
<td>Memory Bits</td>
<td>573,864</td>
<td>3,981,312</td>
<td>14%</td>
</tr>
<tr>
<td>Embedded Multipliers</td>
<td>245</td>
<td>576</td>
<td>43%</td>
</tr>
<tr>
<td>PLLs</td>
<td>1</td>
<td>4</td>
<td>25%</td>
</tr>
<tr>
<td>Pins</td>
<td>88</td>
<td>532</td>
<td>17%</td>
</tr>
</tbody>
</table>

**TABLE V**

**Comparison of FPGA Resources Utilization**

<table>
<thead>
<tr>
<th>Resources</th>
<th>Li [21]</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Elements</td>
<td>13,182</td>
<td>79,264</td>
</tr>
<tr>
<td>Registers</td>
<td>8,025</td>
<td>33,174</td>
</tr>
<tr>
<td>Memory Bits</td>
<td>479,592</td>
<td>573,864</td>
</tr>
<tr>
<td>Embedded Multipliers</td>
<td>0</td>
<td>245</td>
</tr>
<tr>
<td>PLLs</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Pins</td>
<td>134</td>
<td>88</td>
</tr>
</tbody>
</table>
List of Biographies

Jianan Li is a Ph.D. student from the School of Optoelectronics, Beijing Institute of Technology, Beijing, China. His research interests mainly include image/video processing and computer vision.

Tingfa Xu received the Ph.D. degree from the Changchun Institute of Optics, Fine Mechanics and Physics, Changchun, China, in 2004. He is currently a Professor with the School of Optoelectronics, Beijing Institute of Technology, Beijing, China. His research interests include optoelectronic imaging and detection and hyper-spectral remote sensing image processing.

Kun Zhang is currently working toward the Ph.D. degree with the School of Optoelectronics, Beijing Institute of Technology, Beijing, China. His research interests include image processing and digital video stabilization.
List of Footnotes

Manuscript received April 28, 2015. (Corresponding author: Tingfa Xu.)

The authors are with the School of Optical Engineering, Beijing Institute of Technology University, Beijing 100081, China (e-mail: 20090964@bit.edu.cn).
List of Figure Captions

Fig. 1. Pixel allocation around the center point.
Fig. 2. Triangles formed by three selected point pairs.
Fig. 3. Output frame based on mosaicked image.
Fig. 4. Overview of system architecture.
Fig. 5. Underlying data flow architecture.
Fig. 6. Feature Detection and Description module.
Fig. 7. Feature description results of FPGA implementation.
Fig. 8. Feature Matching module.
Fig. 9. Affine Parameter Estimation module.
Fig. 10. Cumulative Affine Parameter Calculator and Kalman Filter modules.
Fig. 11. Measurement environment diagram.
Fig. 12. Frames of test video sequences.
Fig. 13. PSNR of original and stabilized test video sequences.
Fig. 14. Original and stabilized video frames of the system.
Table I. ITF of original and stabilized test video sequences.
Table II. Comparison of computation times.
Table III. Comparison of real-time performance.
Table IV. Statistics of FPGA resources utilization.
Table V. Comparison of FPGA resources utilization.