A time-efficient video stabilization algorithm based on Block Matching in a restricted search space

Kevin Joseph[1], Alex Noel Joseph Raj[2], Zhun Fan[3], Vidhyapathi CM[4]
[1,4]School of Electronics Engineering, VIT University, Vellore, India
kevinjoseph1995@gmail.com[1], vidhyapathicm@vit.ac.in[4]
[2,3]The Key Lab of Digital Signal and Image Processing of Guangdong Province, Shantou University, China
{jalexnoel[2], zfan[3]}@stu.edu.cn

***Abstract*- In this paper, we study various video stabilization techniques and develop an algorithm which can perform video stabilization under strict time constraints. To do this, an optimized version of block matching in a restricted search space is utilized to minimize the use of computational resources. We also develop an experimental setup to do real-time video stabilization under various vibrating conditions. In this study, we have also compared our algorithm with an existing contemporary stabilization algorithm and looked at how the two techniques perform under different circumstances.**

Keywords—video stabilization; block-matching; motion-estimation

I. INTRODUCTION

The prevalence of video capturing devices in the modern age is ever increasing and what all these devices have in common is a preview display. This display mirrors the scene that is being captured and it gives the videographer an idea of what the final captured video would be like. In a lot of cases, the video-camera is not mounted on a stationary platform; this results in a jittery video and the preview is also shaky. Our goal is to develop a stabilization algorithm that is fast enough to process the frames as they are captured and to display the preview to the videographer with minimal delay. This gives the videographer a better sense of the scene that is being captured.

We also have self-driving cars, unmanned aerial vehicles, autonomous underwater vehicles, planetary rovers and newly emerging domestic robots which have some sort of video input. The video acquired by these agents are extremely jittery as the cameras are moving about in an unconstrained manner; this reduces the quality of the video. It is essential that the quality of the video be of a certain standard so that the higher level vision systems can process this video information to help in the navigation of these artificial agents. Thus it is important that the raw video feed is stabilized in real time and with minimal delay before being used by the higher level of vision algorithms. The time constraints dictated by these applications are extremely demanding, which is why the stabilization algorithm must be able to process out frames as quickly as they come in with a minimal delay. The primary objective of this study is to develop a robust video stabilization algorithm which would be able to function in real-time while still producing satisfactory stabilization results.

In many animals, including human beings, the inner ear functions as the biological analogue of an accelerometer in camera image stabilization systems, to stabilize the image by moving the eyes. When a rotation of the head is detected, a signal is sent to the extraocular muscles on one side and an excitatory signal to the muscles on the opposite side. The result is a compensatory movement of the eyes. Typically eye movements lag the head movements by less than 10ms. [1].

Video stabilization systems can be classified into electronic image stabilization (EIS) [2], optical image stabilization (OIS) [3], and digital image stabilization (DIS). EIS uses built-in sensors to detect vibrations and takes the image at the position of reverse displacement to eliminate the interference from vibration. EIS is normally equipped with digital zooming or charge-coupled device (CCD) sensors of increased sizes for motion compensation. OIS uses gyro type of sensors to detect vibration and sends signals to a high-precision control circuit with fast computing capability. The circuit calculates the compensated position for a micro-OIS lens driven by a linear motor to allow the light through the lens to be accurately projected onto the CCD light sensor. DIS uses digital image processing techniques to remove the vibrating effects on images. The advantages of DIS possess are easy to operate, higher flexibility, higher stabilizing accuracy, real-time processing and does not require additional motion sensing devices. Simultaneously, it can distinguish jitter and intentional camera movement from the global camera motion.

Stabilization systems dealing with global motion containing translation, rotation and scaling effects are referred as 3D stabilization. In real-time systems, most proposed stabilization methods only deal with translational jitter for computing simplicity, and these models are referred as 2D stabilization. In this study, we only deal with 2D digital image stabilization. Video stabilization can also be dealt with in two ways namely post-processing and real-time processing. In post-processing, the video has already been captured and is processed to remove jitter and increase overall quality. This is done in cases where there are no time constraints. However, in real-time processing, the video is corrected in real-time and the stabilized frame is displayed immediately as it is corrected. The crux of these algorithms is that they have to produce reasonable stabilization while still being fast enough such that there is no significant drop in video frame-rate.

In this paper we go on to discuss other related works pertaining to video stabilization, a brief overview of 2D video stabilization is also dealt with in the following section. We then present our algorithm and the different stages involved in it. Finally, we discuss the performance of our algorithm with respect to another method and also describe our experimental setup for real-time video stabilization.

II. Related works

This study aims to develop a fast video stabilization algorithm that has the ability to work in multiple scenarios. The proposed algorithm is also compared with the standard stabilization technique that is recommended by the MATLAB documentation which is based on the work by Lee *et al* [4]. The proposed method in this study aims to work in a wider range of scenes; one of the limitations of Lee *et al* [4] method is that it often fail terribly to handle videos with large moving foreground objects. Another limitation of their method is that it requires enough detected and tracked features. Thus our study is to improve upon these disadvantages while still being able to run in real-time. Traditional 2D video stabilization methods [5, 6] proceed in three steps: First, a 2D motion model, such as an affine transformation or a projective transformation, is estimated between successive frames using feature point matching or block matching. Second, the parameters of this motion model are temporally low-pass filtered. Third, frame positions are corrected to remove high-frequency camera shaking. While 2D stabilization is useful for general camera de-shaking, it cannot synthesize an idealized 3D camera path because it has no knowledge of the 3D trajectory of the input camera.

In the first step discussed above, namely *2D motion model estimation* a common technique used is *block matching*. The block matching algorithm (BMA) was commonly used for camera motion estimation [7]. However, when there are moving objects in the scene and regions of plain textures the BMA's can often provide erroneous motion information in those areas. To overcome this problem, feature matching methods for motion estimation have been applied to video stabilization [6, 8]. The performance of these 2D methods is basically limited in two aspects. One is that a full-frame warp cannot model the parallax induced by a translational shift in viewpoint, and another is that there is no connection between the 2D warp and a 3D camera motion. Therefore, several stabilization algorithms using 3D camera motion have been proposed recently [9–12]. Among them, the content-preserving warping (CPW)-based stabilization algorithm (CPWS) developed by Liu is known as a state-of-the-art method [9]. The CPWS algorithm consists of three stages. First, it recovers the 3D camera motion and a sparse set of 3D, static scene points using an off-the-shelf structure-from-motion (SFM) system. Second, the user interactively specifies the desired camera path or chooses one of three camera path options: linear, parabolic, or a smoothed version of the original; the CPWS algorithm automatically fits a camera path to the input. Finally, a least-squares optimization algorithm that computes a spatially varying warp from each input video frame into an output frame is performed. However, in spite of superior visual performance of the CPWS, the least-squares optimization process at the third stage requires a significant number of calculations. For example, it may take three minutes to warp a single 1280x 720 frame. This delay is unacceptable in cases where the frame processing time has to be around 30ms. Tae [13] presents a fast 3D video stabilization algorithm that provides noticeably less computational cost than the state-of-the-art method with the same stabilization performance. Their proposed algorithm reduced the processing time by up to 14% compared to Liu's algorithm but is still not fast enough to be implemented for cases where real-time stabilization is required. The proposed technique in this study aims to compensate stabilization performance with the time efficiency. Thus this technique aims to be implemented in situations where the demands of stabilization performance are not as stringent as time-efficiency. Therefore with the time constraint given prime importance, the next section looks into 2D stabilization techniques in more detail.

III. The Proposed Algorithm

A. *Overview of 2D stabilization*

Most 2D stabilization techniques follow the below 3 steps (refer to Fig.1) as given in [5] and even our proposed algorithm follows similar steps-

Step 1- Estimate translational motion between successive pairs of images.

Step 2- Integrate the cumulative motion curve in the horizontal and vertical directions independently.

Step 3- Translate each image with respect to the previous image such that they follow the integrated motion trajectory. If there is a loss of frame content going out of view (resulting from translation), is to be compensated by interpolating the edges and mosaicing as given in [14].

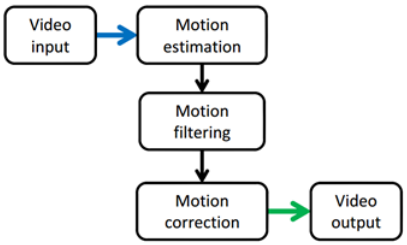

Fig. 1. General 2D video stabilization flow.

Motion estimation is the most expensive step in DIS, both in execution time and in its use of computational resources. This process searches for the optimal parameters of estimated motion and geometrical transformation that makes the current video frame most similar to the preceding frame. Several algorithms for motion estimation have been published in the literature, each exhibiting different features that make them suitable for different applications. In particular, intensity-based methods [15-18] compare the two video frames based on pixel intensity alone, either in the spatial or frequency domain. Feature-based methods [19, 20] apply various transformations to the image to detect

features such as edges or specific objects and keep track of their location to estimate the displacement between consecutive frames. Intensity-based methods rely on consecutive frames having similar lighting conditions, which is expected for video sources, especially as the frame rate increases. Feature-based methods can be very powerful but pose high computational requirements that make them in general unsuitable for portable applications.

In the sections that follow the specifics of the proposed algorithm are discussed.

B. *Video Acquisition and Buffering*

In any video processing technique, it is essential that the acquired frames are collected from the source device and buffered in the physical memory. This is because it is impossible for the stabilization algorithm to process each frame with no delay at all. If instead we capture a frame only after the current one has been stabilized then our frame rate drops drastically and there is a probability that the change in the scene might be drastic. The stabilization algorithm might get delayed on a particular frame that requires more time. This also leads to a discontinuity between the frames. Thus as the first step in the stabilization pipeline, this section discusses how the frames are buffered and processed.

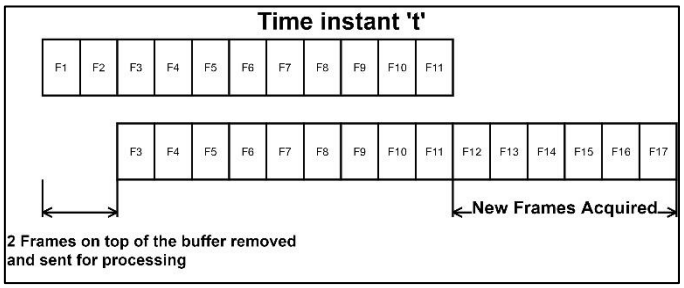

Fig. 2. Frame buffer in memory.

The diagram above (refer to Fig.2) illustrates the framework that allows the stabilization algorithm to use two frames at a time to estimate the motion vector at that instant. Thus the current frame and the previous frame are removed from the buffer and sent for processing. This does not halt the video acquisition in the background at the natural frame rate of the camera. The size of the frame-buffer gradually grows and may overflow memory available if the frames are not processed quickly enough. Thus we must ensure that the stabilization algorithm works fast enough to prevent the available buffer memory from overflowing.

C. *Motion Estimation*

For the purpose of stabilization it is required to estimate a motion vector which provides the direction and magnitude by which the current frame in question has moved with respect to the previous frame. A motion vector is estimated at every time instant and is the most computationally expensive stage of any 2D stabilization scheme. The motion vector is often called *global motion vector* (*GMV*) and can be found by *Full-Search Block method* [21]. A block here is defined as a subset of the image. The size of the block is arbitrarily chosen at first. Later, Mean Average Difference (*MAD*) between the blocks is calculated using equations (1) and (2).

$$MAD(p,q) = \frac{1}{M\times N}\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}|I(t-1,x+k,y+l) - I(t,x+k+p,y+l+q)|. \quad (1)$$

$$p\epsilon\{a+1, a+2, \dots A-a\},$$
$$q\epsilon\{b+1, b+2, \dots B-b\}. \quad (2)$$

Here ‘I’ is the pixel intensity value at a particular location, ‘M’ and ‘N’ are the sizes of the block, (x, y) is the coordinates of the pixel at the top left corner of the block and t is the frame number. ‘A’ and ‘B’ are the sizes of the frame. The vector (p, q) with the minimum ‘*MAD*’ is the desired motion vector. The variables ‘p’ and ‘q’ are assigned numbers within the limits of the search area during the search process, in this case being the entire frame. However some values of ‘p’ and ‘q’ will require us to use the pixel-intensity of undefined boundary regions. These regions are avoided. Here, ‘a’ and ‘b’ are the sizes of the boundary of the search space in both axes. Therefore we basically minimize the *MAD* by searching for the current frame block in the previous frame.

$$GMV = [p\ q\ ]. \quad (3)$$

We have the freedom to choose any block size, but if the block size is too small the chances of occurrence of that block in different place is very high, especially in images with repetitive textures and patterns. This will lead to incorrect estimation of the *GMV*. If the block size is too large, computation time increases but there is also a chance that the scene changed and the block we are trying to locate was not in the previous scene. This again will lead to incorrect motion estimation. For the purpose of real time stabilization, it would be more efficient to choose a small block size. But to ensure that the chosen block does not contain plain surfaces and repeated textures, we perform histogram analysis on multiple small patches and choose the best block out of these. This is discussed below. We select a predetermined number of observation blocks at specific locations in the current frame as illustrated in Fig.3a. Each of these observation blocks is of size m × m pixels and the locations of which are known. For our study, we chose m=46 pixels and 3 observation blocks were distributed along the left diagonal of the current frame.

The size and distribution of the observation blocks are arbitrary but, the position of the blocks is such that they are not on the edge of the frames. We perform a local histogram analysis on each of the observation blocks to get a statistical distribution of the pixel intensity values. By looking at the variance of the pixel intensity values for the three histograms we select the one highest spread/variance.

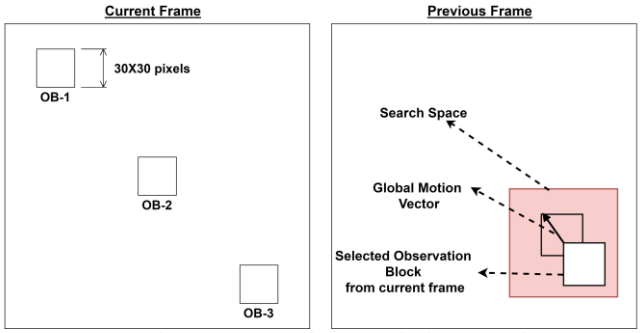

Fig. 3a. Distribution of observation blocks (OB's) in the current frame on the left. Fig. 3b. On the right we have the selected observation block and the search space in the previous block.

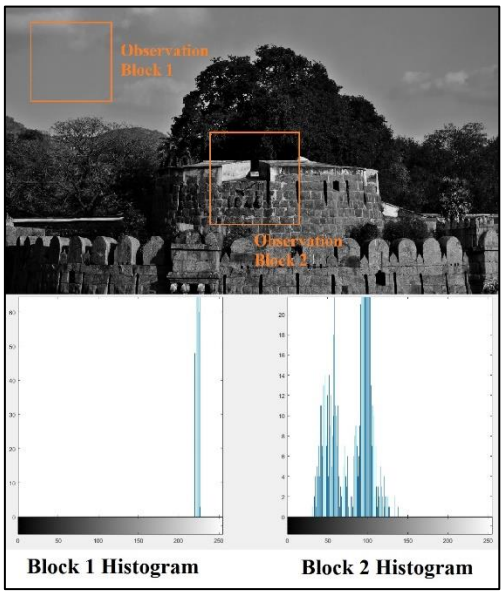

Fig. 4. Two observation blocks in a scene and their corresponding histograms.

Block 2 as shown in the Fig. 4 has more characteristic features and varying pixel values. Block 1, on the other hand, is a plain region. Thus by looking at the histograms, we can arrive at the conclusion that block 2 will be a better candidate to use when trying to estimate the *GMV*. Once we have selected the observation block from the 3 candidate blocks, we try locating the observation block in the previous frame (refer to Fig.3b). It would be a waste of computational resources by keeping the entire of the previous frame as our search space. Thus we limit our search only within a region of interest. It is fairly logical to assume that our region of interest is in the neighborhood of the observation block's original location as it would not have been displaced by a great deal between the two frames. In our study, the region of interest in the previous frame is inclusive of all points the observation block occupied in the current frame and a border surrounding this region which is 5 pixels wide.

After selection of the observation block, we search for the block in the previous frame in a restricted search space as shown in Fig. 3b Mathematically we are performing a similar minimization as shown in (1), the only difference being that the range of values (p, q) can assume are restricted as given by (4) and (5).

Here 'R' defines the width of the boundary of the search space/region of interest around the location of the observation block. This limited search space allows for fast location of the block in the previous frame. Once we have obtained the values of 'p' and 'q' which minimized the *MAD*, we assign these values as the elements of the *GMV*.

$$p\epsilon\{x-R, x-R-1, \dots, x, x+1, \dots, x+M-1+R\}. \quad (5)$$
$$q\epsilon\{y-R, y-R-1, \dots, y, y+1, \dots, y+N-1+R\}. \quad (6)$$

D. Motion Filtering

The *GMV* as predicted by the previous section is nothing but a displacement vector which denotes the direction and magnitude of the camera motion in the 2D plane perpendicular to lens's optical axis. This movement can either be intentional camera movement such as panning or unintentional vibration/jitter. It is required of any stabilization system to discern between intentional camera movements and jitter. Once this has been evaluated to compensate for the movement the current frame is moved in the opposite direction of the vibration at that instant. Intentional camera motion is preserved in the process. Jitter/Vibration manifests itself in the trajectory of the camera as the high-frequency component; thus smoothening of the trajectory is nothing but low-pass filtering this signal. One method is called frame position smoothening [22], wherein the *GMV* is accumulated at every time instant by summing it up. This way the path of the camera is reconstructed in both axes. After the path has been constructed, it can be low pass filtered to smoothen it. However, this low pass filtering has the important drawback of requiring off-line post-processing of sequences. Thus real-time implementation of this technique is not possible. The technique we have used is motion vector integration (MVI) [22] as given by (7).

$$GMV_{integrated}(t) = k \times GMV_{integrated}(t-1) + GMV_{estimated}(t-1) \quad (7)$$

$GMV_{integrated}$ is the motion vector we use to correct the position of the current frame and $GMV_{estimated}$ *is* the motion vector as predicted by the previous stage of the algorithm. The damping parameter is shown by 'k'. The greater the value of k is, the greater the ability of vibration absorption. In practice, the damping factor was between 0.875 and 0.995 according to the required stabilization intensity [23]. The integrated motion vectors calculated for the present frames are used directly as correction vectors to translate image frames into their stabilized position. Thus the motion vector integration system constitutes a first order IIR filter, that directly low-pass filters the differential motion vectors to remove inter-frame jitter.

There are different MVI techniques developed for different scenarios. For example, when the scene which is being stabilized has only jitter and no intentional camera movements the following technique is used [24].

$$GMV_{integrated}(t) = k \times GMV_{integrated}(t-1) + \left(\alpha \times GMV_{estimated}(t) + (1-\alpha) \times GMV_{estimated}(t-1)\right). \quad (8)$$

In the above equation α =0.75 and k = 0.95 are used, $GMV_{integrated}$ and $GMV_{estimated}$ are the same as previously defined. This variant works well in cases where the amplitude of vibration of the camera is high. However, when the camera moves at a constant speed (in an intended direction), a larger k will make the $GMV_{integrated}$ value larger. Due to fixed search range of motion estimation, the ability of vibration absorption will be decreased and the compensated trajectory of $GMV_{integrated}$ will lag behind the actual trajectory.

To overcome the issue of constant motion/intentional motion, the vector of constant movement of the camera or *constant motion vector* (CMV) as given by (10) should be subtracted from the expression as given by (9). This will reduce the size of the correction vector as well as increase the ability of compensation. Therefore, Hsu [25] has added an integrator in the formula to improve this lag problem while the camera is panned at a constant speed. The formula is given in (9):

$$GMV_{integrated}(t) = k \times GMV_{integrated}(t-1) + \left(\alpha \times GMV_{estimated}(t) + (1-\alpha) \times GMV_{estimated}(t-1)\right) - \beta \times CMV(t-1). \quad (9)$$

$$CMV(t) = CMV(t-1) + GMV_{integrated}(t). \quad (10)$$

Here the constant k is used to smooth the playback of camera panning, and constant β is used to filter out abnormal vibrations. However, the compensated trajectory will exceed the original path and oscillate around it in some cases which are referred to as the overshooting phenomena. But these type of aberrations is negligible. These two methods have different advantages and disadvantages. The method proposed in [24] has a better stability in reciprocating motion for motion compensation, but on the other hand, Hsu [25] produced a better performance when there was intentional motion of the camera.

E. *Motion Correction*

In our study, we have utilized the method as presented by Hsu. Once $GMV_{integrated}$ is calculated at every time instant, we use this as the correction vector to correct the current frame as given by (11). This process happens continuously as new frames are fed from the acquisition device.

$$I_{\text{corrected frame}}(x, y) = I_{\text{current frame}}\left(x + GMV_{integrated}(1), y + GMV_{integrated}(2)\right). \quad (11)$$

In cases when the amplitude of vibration is too large the corrected frame will require intensity values at undefined regions of the current regions. The extent of these regions varies from frame to frame and presents unacceptable visual artifacts. To our knowledge the solution to this problem has not been discussed in the literature, while available software products use frame trimming and magnification or filling by a constant value. These methods lead to severe quality degradation of the resulting video and limit the range of possible correcting transformations (significant transformations lead to large undefined areas). Litvin et al [14] proposed to use mosaicing for each frame in order to exploit temporal correlations between frames. This however is a time consuming process. In this study we have placed a limit on the magnitude on the elements of $GMV_{integrated}$ (50 in our case). Thus when correcting the current frame it cannot be displaced by more than 50 pixels along either of the two axes. This ensures that the corrected frame does not have undefined regions.

IV. Results

A. *Experiment 1: Study of algorithm performance on synthetic videos*

A number of experiments were conducted to analyze the performance of our version of *motion vector integration*. Initially all our testing was focused on video scenes where vibration was synthetically induced by programmatically shifting the frames in random directions of a stationary scene. When we stabilized the resulting video using our technique, what was observed was that after the first 4 to 5 frames the video was entirely stabilized. This was repeatedly observed as long as the amplitude of the artificial vibration was within the allowable limits of the *search space* of the *observation block*. But these sort of artificial simulations can never truly capture the essence of actual vibrations that a camera has to endure when capturing videos. Thus for a better understanding of how our algorithm performs we have used videos that were captured in real-life scenarios, we also made our own experimental set-up to study the behavior of our stabilization technique while it stabilizes in real-time.

B. *Experiment 2: Comparison with stabilization algorithm based on feature trajectories*

We used the video sequences which are from http://pages.cs.wisc.edu/~fliu/project/3dstab.htm in order to compare our technique with the standard method given by the MathWorks documentation which is based on Lee *et al* [4]. Their method extracts feature trajectories from the input video. A set of transformations are located by a set of optimizations to smoothen the trajectories and stabilize the video. Their method has the ability to smoothen small rotational variations and translatory vibrations. Their method works exceptionally well except in cases where there is camera rotation and magnification. It produces deformations in the frame as shown in Fig. 6c. Looking at the change in scene between Fig. 6a and Fig. 6b, there is change in scale as well as camera rotation. Their method also fails in cases where there is intentional movement of the camera. The optimization which estimates the geometric transform between every frame is unable to

Fig. 5. The five frames on the top row are from the raw video and the five frames of the bottom row are of the stabilized video.

discern the intentional movement when smoothening the trajectory. This is where our technique performs better and at the same time Lee's technique is time consuming as estimating the geometric transform for every frame is computationally expensive.

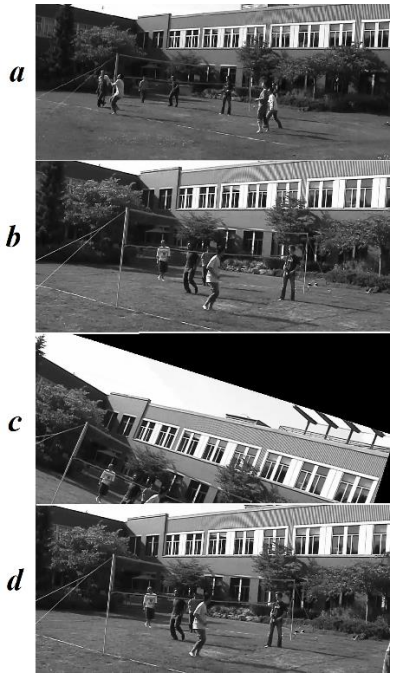

Fig. 6a. The 1st frame in the sequence of the un-stabilized video.
Fig. 6b. The current frame of the un-stabilized video.
Fig. 6c. Depicts the frame as stabilized by Lee's method which is severely distorted.
Fig. 6b. This is the frame stabilized by our method

Another prerequisite for the method presented in [4] is that the video scene must have sufficient number of feature points, in order to estimate the geometric transform. Thus this system will fail in poorly lit scenes and cases where the effect of *motion blur* is too high. Our method will perform better in these scenarios but still has its limitations. If the scene has a lot of repeated textures and features, the *motion estimation* stage of our method will predict the wrong motion vector and this will lead to erroneous results [4].

C. Experiment 3: Real-time stabilization and experimental setup.

For real time implementation we have tested the algorithm on a Windows 7 system running MATLAB 2016 connected to a Logitech C270 web-camera recording at 30 frames per-second. The camera was mounted on a robotic arm which allowed the camera to move along the horizontal axis as shown in Fig 7. Vibrations were induced by means of vibration motors attached on the robotic arm and the incoming raw video feed was stabilized in real time. Fig. 5 shows the sequence of frames as captured by the set-up previously described and its stabilized counterpart.

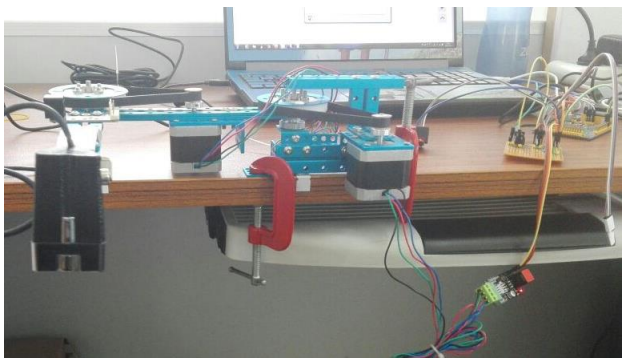

Fig. 7. The experimental setup with the robotic arm on which the video camera was mounted on. The arm could pan along the horizontal axis while vibration motors along the arm would simulate real-life distortions.

We have tracked one particular feature-point in both the sequences and it is seen that the point is displaced less erratically in the stabilized frame. If the right most region of corresponding frames are obsereved (refer to Fig.5), it is seen that in the raw video the second laptop screen goes out of the frame whereas in the stabilized video the algorithm is able to retain that region in the frame.

This is not that evident in Fig.5 but by tracking the location of that feature in both the axes as each frame passes we get a better estimate of the stabilization performance. The X and Y locations of this feature point is graphically represented in Fig. 8.

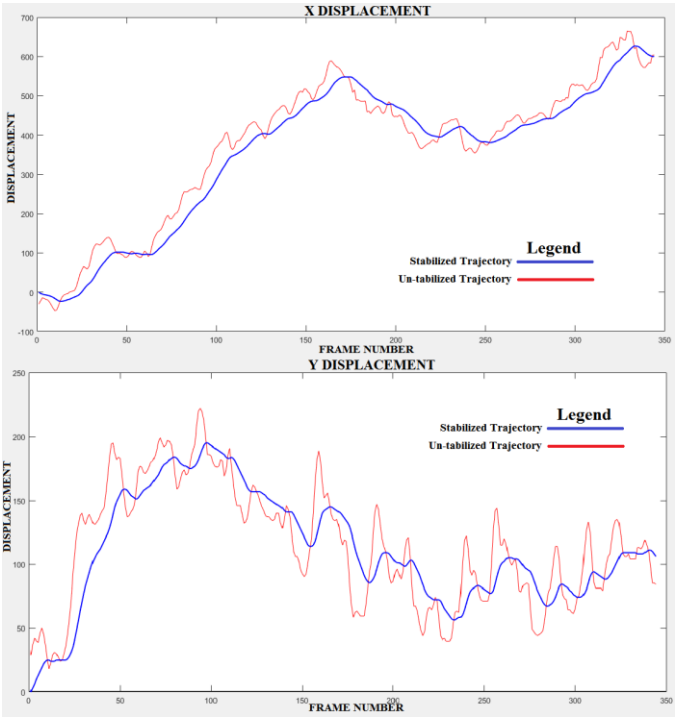

Fig. 8. The red line denotes the path of the feature point in the un-stabilized video whereas the blue line is the stabilized path.

From the graphs in Fig.8 it's is worth noting that the *Motion Filtering* stage has retained the intentional camera movements and has simple smoothened out the high amplitude aberrations.

D. *Experiment 4: Effect of damping factor on stabilization performance*

In Fig. 9 stabilization algorithm was with different values of the 'damping factor'. If we look at Fig 9.a, k=0.70 was used in (9) during the stabilization process. The value of the 'k' in this case is too small as the stabilized trajectory almost follows the original trajectory of the tracked feature point. But as we progressively go on to increase the value of the 'damping factor' we notice that stabilization performance increases as the path become more stabilized. But if we observe in Fig. 9.d with k=0.99 the smoothening of the path was overdone and a significant part of the original trajectory was distorted and the intentional camera movement was not retained. This manifests itself in the stabilized video as unsynchronized frames and is not visually appealing. Based on experimental observation we chose an optimum value of k=0.90.

E. *Experiment 5: : Effect of block size on stabilization performance*

Table I shows how varying the size of the square observation blocks affects performance. In an m × m pixel dimension block, we see how performance varies with different values of m. What can be inferred from the above table is that as the size of the block is increased the motion estimation system is able to predict motion with increasing

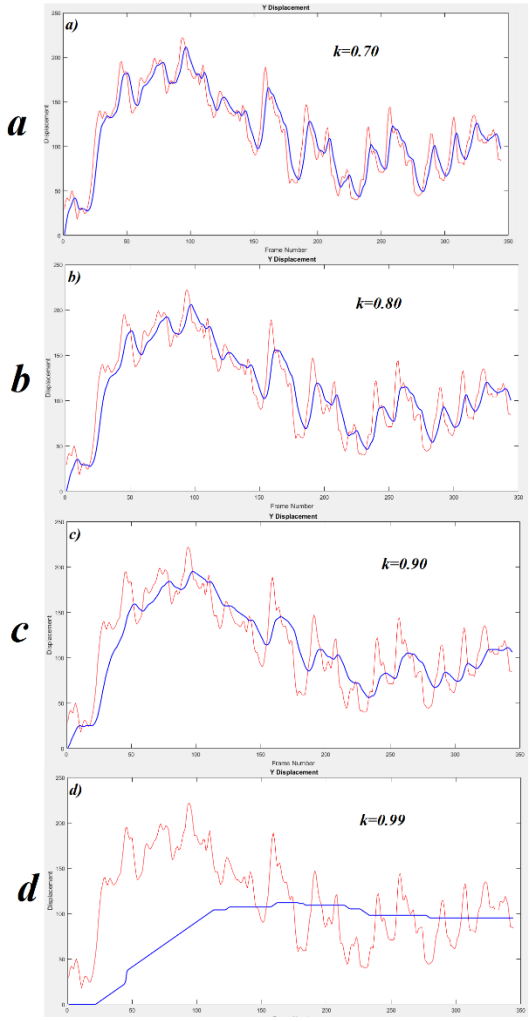

Fig. 9. Here we have the same video sequence stabilized with different values of the damping factor (k). We have traced the vertical position of a particular feature point in the above figure. Fig. 9a. Stabilized with k=0.70. Fig. 9b. Stabilized with k=0.80. Fig. 9c. Stabilized with k=0.90. Fig. 9d. Stabilized with k=0.99.

TABLE I. EFFECT OF BLOCK SIZE

Size of the observation block.(m × m pixels)	Mean Squared Error (MSE) between original trajectory and predicted trajectory (pixel2) along X-axis	Mean Squared Error (MSE) between original trajectory and predicted trajectory (pixel2) along Y-axis	Average mean squared error of both axes	Processing time for a 3 second video (s).
20	15.1414	5.0000	10.0707	2.345
26	3.6566	10.5960	7.1263	2.897
30	1.6566	23.3333	12.49495	3.121
36	2.2525	22.7273	12.4899	4.300
40	3.8384	2.8182	3.3283	4.678
46	0.7879	0.7677	0.7778	6.133
50	1.7576	0.6364	1.197	7.643
56	0.1313	0.3232	0.22725	10.311
60	2.9899	4.5354	3.76265	12.614

accuracy as the mean squared error (MSE) drops (for both the axes) barring some anomalies. It is also observed that as the block size increases the processing time also increases. This is because computing the histogram of larger blocks will progressively take more time. Thus for our real-time implementation, it is essential that a balance between stabilization performance and processing time is found. Based on this experiment a block size of m=46 was selected for the proposed method.

V. Conclusion and Future Work

In its current form, with the system being implemented on MATLAB we were able to implement video stabilization in real-time when we selected a block size of m=46. This, resulted in the motion estimation subsystem to give an average MSE of approximately 0.7778 as given in Table I. This performance can be improved if the proposed algorithm was implemented on dedicated hardware with scope for parallelism. FPGA's are best candidates for our proposed algorithm to be deployed on, with their inherent parallel nature of computation.

Another area of improvement is to equip the algorithm to perform better in cases where there is a change in scale of the scene (magnification) as a result of movement of the camera in the third degree of freedom. The proposed algorithm in its current for is unable to suppress minor rotational movements of the camera. This is another issue that will be addressed in the further study of the topic.

Apart from the visual data being used as cue's to stabilize the raw video, sensor inputs from gyroscope and accelerometer can be integrated into the digital image stabilization algorithm; this can result in improved stabilization results.

Thus in this paper, we present a robust stabilization technique that has the ability to work in different situations with relatively acceptable stabilization performance. It has been shown how our proposed method performs in comparison with a contemporary standard stabilization technique.

References

[1] Barnes, Graham R. "Vestibulo-ocular function during co-ordinated head and eye movements to acquire visual targets." The Journal of Physiology 287 (1979): 127.

[2] Holder, Donald W., and William R. Phillips. "Electronic image stabilization." U.S. Patent No. 4,637,571. 20 Jan. 1987.

[3] Mcleod, Stuart, and Ewan Findlay. "Optical image stabilization." U.S. Patent No. 8,436,908. 7 May 2013.

[4] Lee, Ken-Yi, et al. "Video stabilization using robust feature trajectories." Computer Vision, 2009 IEEE 12th International Conference on. IEEE, 2009.

[5] Yeh, YeouMin, Sheng-Jyh Wang, and Hwang-Cheng Chiang. "Digital camcorder image stabilizer based on gray-coded bit-plane block matching." Photonics Taiwan. International Society for Optics and Photonics, 2000.

[6] Matsushita, Yasuyuki, et al. "Full-frame video stabilization with motion inpainting." IEEE Transactions on Pattern Analysis and Machine Intelligence 28.7 (2006): 1150-1163.

[7] Ratakonda, Krishna. "Real-time digital video stabilization for multi-media applications." Circuits and Systems, 1998. ISCAS'98. Proceedings of the 1998 IEEE International Symposium on. Vol. 4. IEEE, 1998.

[8] S. Battiato, G. Gallo, G. Puglisi, S. Scellato, SIFT features tracking for video stabilization, in: Proceeding IEEE International Conference on Image Analysis and Processing, 2007, pp. 825–830.

[9] Liu, Feng, et al. "Content-preserving warps for 3D video stabilization." ACM Transactions on Graphics (TOG) 28.3 (2009): 44.

[10] Wang, Jung Ming, et al. "Video stabilization for a hand-held camera based on 3D motion model." Image Processing (ICIP), 2009 16th IEEE International Conference on. IEEE, 2009.

[11] Zhang, Guofeng, et al. "Video stabilization based on a 3D perspective camera model." The Visual Computer 25.11 (2009): 997-1008.

[12] Liu, Feng, et al. "Subspace video stabilization." ACM Transactions on Graphics (TOG) 30.1 (2011): 4.

[13] Lee, Tae Hwan, Yun-gu Lee, and Byung Cheol Song. "Fast 3D video stabilization using ROI-based warping." Journal of Visual Communication and Image Representation 25.5 (2014): 943-950.

[14] Litvin, Andrey, Janusz Konrad, and William C. Karl. "Probabilistic video stabilization using Kalman filtering and mosaicing." Electronic Imaging 2003. International Society for Optics and Photonics, 2003.

[15] Cain, Stephen C., Majeed M. Hayat, and Ernest E. Armstrong. "Projection-based image registration in the presence of fixed-pattern noise." IEEE transactions on image processing 10.12 (2001): 1860-1872.

[16] Bin, Yin, and Duan Hui-chuan. "Image stabilization by combining gray-scale projection and block matching algorithm." IT in Medicine & Education, 2009. ITIME'09. IEEE International Symposium on. Vol. 1. IEEE, 2009.

[17] Kumar, Sanjeev, Mainak Biswas, and Truong Q. Nguyen. "Efficient phase correlation motion estimation using approximate normalization." Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on. Vol. 2. IEEE, 2004.

[18] Auberger, Stéphane, and Carolina Miro. "Digital video stabilization architecture for low cost devices." Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on. IEEE, 2005.

[19] Battiato, Sebastiano, et al. "SIFT features tracking for video stabilization." Image Analysis and Processing, 2007. ICIAP 2007. 14th International Conference on. IEEE, 2007.

[20] Sledevič, T., and A. Serackis. "SURF algorithm implementation on FPGA." Electronics Conference (BEC), 2012 13th Biennial Baltic. IEEE, 2012.

[21] Zhu, Shan, and Kai-Kuang Ma. "A new diamond search algorithm for fast block matching motion estimation." Information, Communications and Signal Processing, 1997. ICICS., Proceedings of 1997 International Conference on. Vol. 1. IEEE, 1997.

[22] Erturk, Sarp. "Image sequence stabilisation: Motion vector integration (MVI) versus frame position smoothing (FPS)." Image and Signal Processing and Analysis, 2001. ISPA 2001. Proceedings of the 2nd International Symposium on. IEEE, 2001.

[23] Uomori, Kenya, Atsushi Morimura, and Hirofumi Ishii. "Electronic image stabilization system for video cameras and VCRs." SMPTE journal 101.2 (1992): 66-75.

[24] ki Paik, Joon, Yong Chul Park, and Dong Wook Kim. "An adaptive motion decision system for digital image stabilizer based on edge pattern matching." IEEE Transactions on Consumer Electronics 38.3 (1992): 607-616.

[25] Hsu, Sheng-Che, et al. "A robust in-car digital image stabilization technique." IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 37.2 (2007): 234-247.