

Digital Image Stabilization and Sharpening

Kyle Chavez

Evan Sproul

4/21/2011

EE 552

1. Introduction

One promising method of solar power concentration is the heliostat array. A heliostat consists of a large mirror with the mechanisms and circuitry necessary to actuate it, such that it reflects sunlight onto a given target throughout the day. A heliostat array is a collection of heliostats that focuses sunlight continuously on a central receiver, often called a power tower. Figure 1 shows an example of a heliostat array.



Figure 1: Sandia National Laboratories National Solar Thermal Test Facility heliostat array. This array consists of 222 heliostats and a single 200ft tall power tower.

Optimally concentrating sunlight with heliostats requires that each heliostat mirror is correctly canted (positioned) and focused (shaped). In order to minimize facet canting and focal errors, Sandia National Labs and New Mexico Tech are developing heliostat optical analysis tools. In many cases these tools utilize a camera positioned on top of the power tower (200 feet above ground level). This camera position combined with the large heliostat field size places the camera at a total distance of 300-700 feet away from the heliostat mirror surface being analyzed. Figure 2 displays the camera and heliostat positioning.

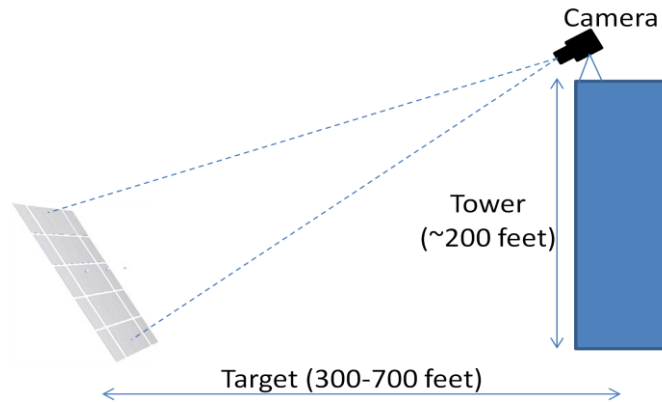


Figure 2: Camera and heliostat positions during optical analysis.

To compensate for the large distance between the heliostat and camera, a long range optical zoom lens (80-400 mm focal length) is used when acquiring images and video of heliostat mirrors. This lens allows the camera to acquire high resolution images at great distances. However, one major drawback of this system is the high sensitivity of the zoom lens and camera to small position disturbances. These disturbances include camera and lens position shift due to wind, structural vibrations, and thermally induced strain. Although these disturbances are small, they can create large shifts in the camera's field of view.

The optical analysis of heliostats relies heavily upon determining the exact positions of specific features in acquired images and video. As a result, any shift in a camera's field of view will have a negative effect on analysis by altering the location of these features within an image. This alteration results in poor analysis and incorrect heliostat canting and focusing.

The following project attempts to implement a software solution to track and correct for shifts in camera position. In the project we reviewed and compared literature on multiple methods for motion correction. We then implemented one method on sample video that mimics the camera position change commonly seen during optical analysis. As a secondary portion of the project, the blur induced by camera motion was investigated and corrected for a single frame using methods previously demonstrated in the EE 552 course.

2. Preexisting Methods for Motion Correction

Our investigation of the preexisting methods for motion correction followed a project paper from a Northwestern University digital image processing course [1]. In the paper, Brooks discusses the various methods for motion tracking and control. These methods include spatio-temporal approaches such as

block matching [2], direct optical flow estimation [3], and least mean-square error matrix inversion [4], as well as, matching methods including bit-plane matching [5], point-to-line correspondence [6], feature tracking [7], pyramidal approaches [8], and block matching [9]. Upon investigating these references we decided to pursue a gray-coded bit-plane matching (GSBPM) method [5]. This method requires minimal computation and is ideal for quickly determining and correcting motion.

3. Implementation of GCBPM Method

In order to implement the selected correction method, we first had to generate a video file that accurately mimicked the sort of motion experienced by the camera on top of the power tower. Our initial attempt to acquire actual tower top video was unsuccessful due to unusually high winds that exceeded the scope of this project. As a result, we instead developed a generic video (acquired in an indoor office setting) that had camera motions similar to that induced by average wind speeds and other disturbances. Using the webcam also allowed us to utilize a slightly lower image resolution. This smaller resolution allowed us to capture a longer uncompressed video segment that we could then attempt to process.

Upon creating the video, development of the bit-plane algorithm began. During development we followed the methods presented by Gonzalez [10] and Ko [5]. Initially, we used the dollar bill image presented in Figure 3.14 of Gonzalez to ensure our algorithm's accuracy. Figure 3 displays our algorithm's results for comparison purposes.

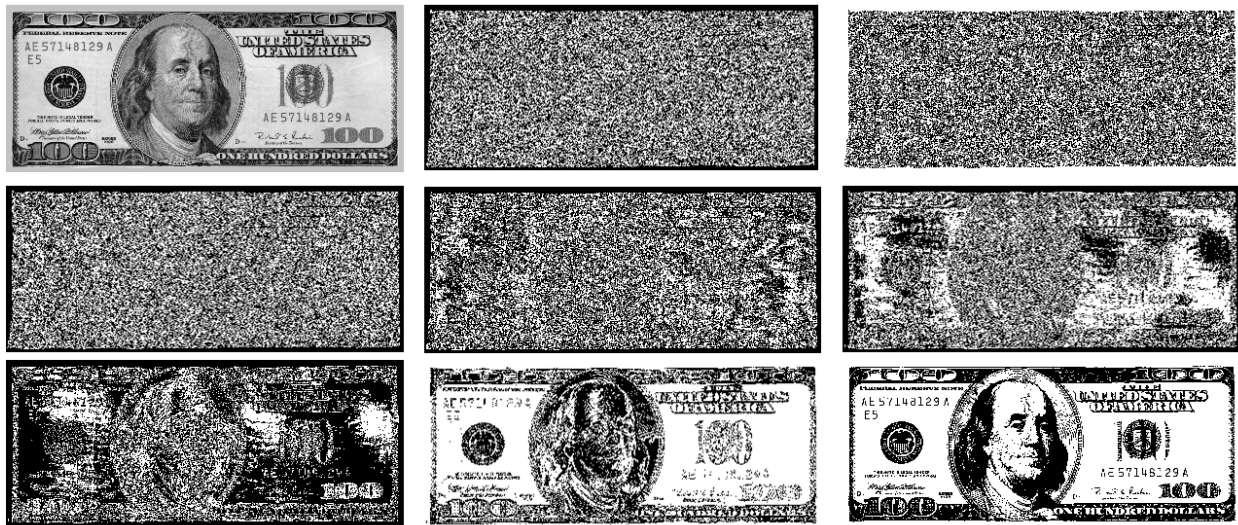


Figure 3: Bit-plane sliced of 100 dollar bill image.

After successful implementation of the initial bit-plane algorithm, we moved towards adapting the algorithm for gray-coded bit-plane separation as discussed in Ko's paper. Equations 1 and 2 were used to generate the final gray-coded bit-plane.

$$f^t(x,y) = a_{k-1}2^{k-1} + a_{k-2}2^{k-2} + \dots + a_02^0 \quad (1)$$

$$g_{k-1} = a_{k-1}$$

$$g_k = a_k \oplus a_{k+1}, \quad 0 \leq k \leq K-2 \quad (2)$$

In the Equation 1 $f^t(x,y)$ represents the gray level of the t^{th} image frame at coordinates x and y . In Equation 2 the g_k represents the eight-bit gray code. In both equations a_k are the standard binary coefficients.

Our results after applying Equations 1 and 2 to the initial frame of our video segment are shown in Figure 4. From the results of Figure 4 we chose bit-planes four and five as good candidates for analysis. These layers showed a large amount of detail making them ideal for detecting changes in the image.

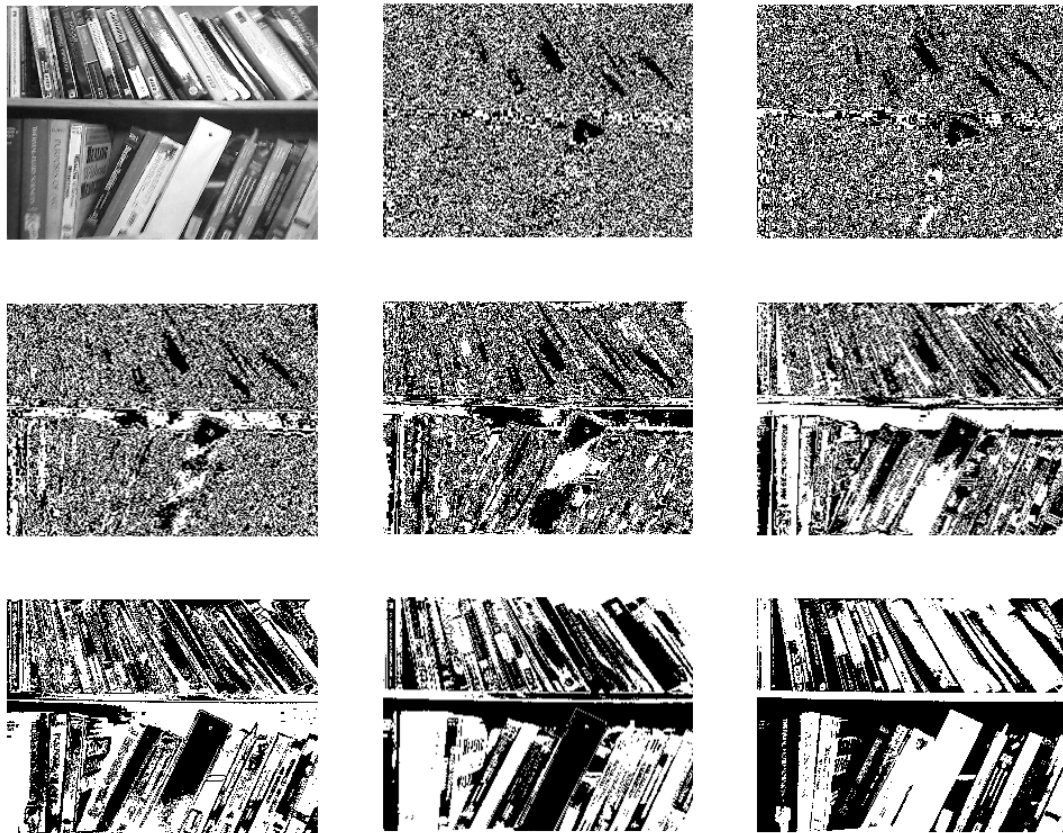


Figure 4: Gray-coded bit-planes of initial video frame.

After determining an ideal bit-plane we created the appropriate code to calculate the approximate motion vector of successive video frames. The code analyzes four small sub-images of the single frame bit-plane image. The sub-images of the bit-plane image are shown in Figure 5.

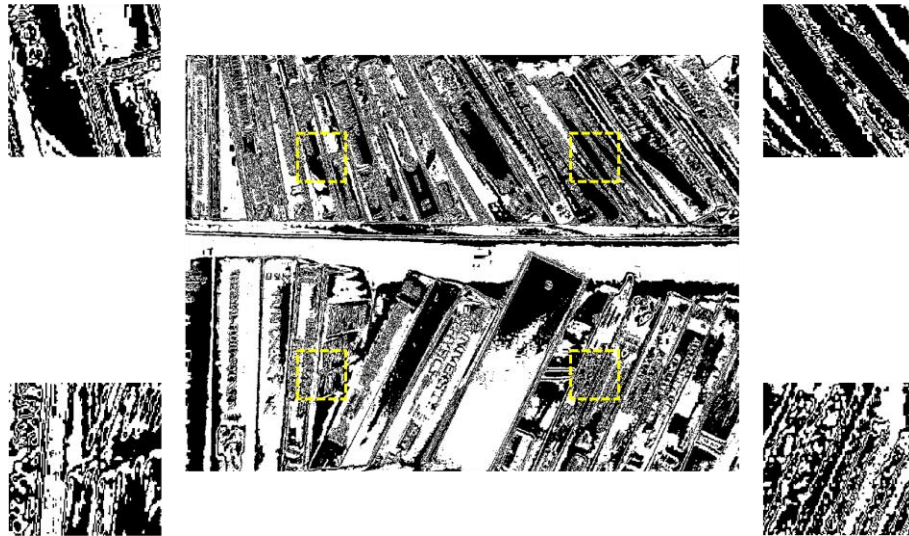


Figure 5: Single bit-plane with highlighted and zoomed sub-images.

During analysis, the code uses Equation 3 to find the error function of each sub-image between successive frames. In this equation W is the width of the sliding block and p is the range of pixels being analyzed.

$$\varepsilon(m,n) = 1/(Wz) \sum \sum g_k^t(x,y) \oplus g_{k-1}^t(x+m,y+n), \quad -p \leq m,n \leq p \quad (3)$$

Once the error function for each sub-image is known, the (m, n) location of the error minimum in each error function is found. Furthermore, the algorithm takes the median value of the four error function minimums and uses this value to approximate a global motion vector for the entire image. This process is repeated for each pair of successive frames and outputs the motion vector of each frame. Knowing these vectors the code can then alter each original video frame to keep the objects of interest in the correct centered location.

In order to speed up the overall processing time of our algorithm we implemented a graphical processing unit (GPU). In order to run the code on the GPU we first developed the code in MATLAB to run on a computer central processing unit (CPU). Once full operation was verified, we adapted the code using the Accelereyes Jacket software program. This allowed us to maintain the general MATLAB code structure, but implement a GPU as desired.

Lastly, we addressed the issue of sharpening a single video frame. Although blurring is not a large issue when viewing video clips, there are instances where single frames are captured and observed. In these cases, blurring can be a significant problem. To address this problem we applied the Weiner filter code developed previously in the course. To correctly adjust the filter for the specific application, input constants used to generate an appropriate $H(u,v)$ were determined iteratively by viewing the output results until suitable inputs were found.

4. Results

The results of the stabilization portion of this project are best showcased in the output video file. Figure 6 shows three video frames that demonstrate the results shown in the video.



Figure 6: Frame 1 of original video (left), frame 80 of original video (center) and frame 80 of corrected video (right).

By looking at a specific feature of Figure 6 we can see the overall correction applied by the stabilization algorithm. The corner of the rightmost used sticker in frame 80 of the original video is located at (1492, 441). The same feature in the corrected version of frame 80 is located at (1060, 214). By comparing these positions to the original feature position in frame one at (1032, 197) we compare the relative motion of the different video frames. From this comparison we find that frame 80 of the original video moved 360 pixels horizontally and 244 pixels vertically. Furthermore, frame 80 of the corrected video moved only 28 pixels horizontally and 17 pixels vertically.

The use of a GPU decreased processing time of the video segment. In this particular case the video segment had a resolution of 1600x1200 and GCBPM was performed with a W of 49 and a p value equal to 100. When using a CPU the overall runtime for ninety frames with these settings was 563.64 seconds, giving a processing speed of 0.16 frames per second. Using a GPU the overall runtime for ninety frames with the same settings was 159.24 seconds, yielding a processing speed of 0.57 frames per second. All processing times include the necessary steps of reading and writing video files.

The sharpening results are shown in Figure 7. Because image sharpening was a secondary result, the GPU was not used during execution of the sharpening algorithm.

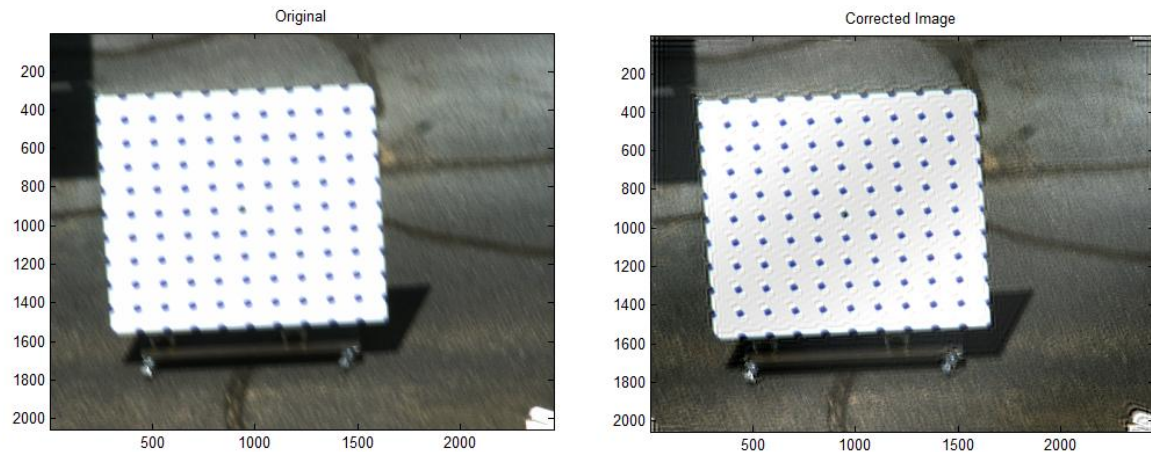


Figure 7: Original blurred frame (left) and sharpened frame (right).

5. Conclusions and Future Work

The methods used in this project were successful in stabilizing a moving video segment that mimics video commonly acquired using heliostat optical analysis tools. These results confirm that it is possible to implement GCBPM to accurately approximate the motion of successive video frames. Additionally GCBPM can be adapted to a GPU, which in this case resulted in observed processing speed increase of 354%. In addition to stabilization, image processing techniques can be used to sharpen individual image frames and improve the clarity of image features.

With these conclusions established it is now possible to integrate the same stabilization techniques into actual heliostat mirror analysis video. In doing so, additional processing improvement steps should be considered to increase the overall quality of output video. Some techniques to be considered are motion dampening to minimize small motion and further integration of the GPU to allow for efficient processing of high resolution video. Furthermore, it is important to note that the quality of stabilization was highly dependent upon the size of the sub-image being analyzed. To guarantee the highest stability, different sub-image sizes should be utilized until an optimum balance between processing time and stabilization quality is found.

6. References:

- [1] A. Brooks, Real-Time Digital Image Stabilization, Image Processing Report, Department of Electrical Engineering, Northwestern University, Evanston, IL, 2003.
- [2] T. Chen. Video Stabilization Using a Block-Based Parametric Motion

Model. Technical report, Stanford University, Information Systems Laboratory, Dept. of Electrical Engineering, Winter 2000.

[3] J. Chang, W. Hu, M. Cheng, and B. Chang. Digital image translational and rotational motion stabilization using optical flow technique. IEEE Transactions on Consumer Electronics, vol. 48, no. 1, pp. 108-115, Feb. 2002.

[4] C. Erdem and A. Erdem. An illumination invariant algorithm for subpixel accuracy image stabilization and its effect on MPEG-2 video compression. Elsevier Signal Processing: Image Communication, vol. 16, pp. 837-857, 2001.

[5] S. Ko, S. Lee, S. Jeon, and E. Kang. Fast digital image stabilizer based on gray-coded bit-plane matching. IEEE Transactions on Consumer Electronics, vol. 45, no. 3, pp. 598-603, Aug. 1999.

[6] M. Ben-Ezra, S. Peleg, and M. Werman. A Real-Time Video Stabilizer Based on Linear Programming.

[7] A. Censi, A. Fusiello, and V. Roberto. Image Stabilization by Features Tracking. Technical report, University of Udine, Machine Vision Laboratory, Dept. of Mathematics and Informatics, 1998.

[8] J. Jin, Z. Zhu, and G. Xu. Digital Video Sequence Stabilization Based on 2.5D Motion Estimation and Inertial Motion Filtering. Real-Time Imaging, vol. 7, pp. 357-365, 2001.

[9] J. Jin, Z. Zhu, and G. Xu. A Stable Vision System for Moving Vehicles. IEEE Transactions on Intelligent Transportation Systems, vol. 1, no. 1, pp. 32-39, Mar. 2000.

[10] R. Gonzalez, and R. Woods. Digital Image Processing Third Edition. Upper Saddle River, NJ: Pearson Prentice Hall, 2008, pp. 118.