GPU Accelerated Elevation Map based Registration of Aerial Images

Joseph French, *Student Member, IEEE*, William Turri, Joseph Fernando, *Member, IEEE and* Eric Balster *Senior Member IEEE*

{joseph.french, William.turri, joseph.fernando}@udri.udayton.edu, ebalster1@notes.udayton.edu

Abstract—This paper proposes a lower latency implementation of the georegistration algorithm proposed by [5]. The algorithm has been modified to mitigate the registration errors and has been parallelized to map to a Graphical Processor Unit (GPU). Also, the target image offset and the painting value computations have been combined to a single loop to eliminate the use of shared memory. Modifications to a current widely used algorithm are proposed. The proposed modified algorithm has been implemented in compute unified device (CUDA) architecture to reduce latency. A fixed coordinate system is used to represent the image, focal, and projection planes. Experimental results show that the proposed algorithm is capable of generating accurate georegistered images for high flying airborne vehicles. While this method has been tested using aerial photographs, it can be extended to Satellite images as well as other image data. A speedup of over 10x has been achieved over the CPU version.

***Index Terms*— Camera model, Digital Elevation Maps, multiple threads, Georegistration, Orthorectification**

I. INTRODUCTION

This paper is based on the methodology proposed by Jovanovic et. al [5]. The original algorithm has been modified to mitigate errors and has been enhanced to map to a Graphical Processor Unit (GPU) to lower latency. Note that registration is the highest latency causing function in most airborne system image processing software chain. It has a higher latency than preprocessing and image compression. The main reasons for selecting this methodology are, firstly, the algorithm could be multi threaded and parallelized, secondly, it could be modified to fit the resource in the GPU namely, the shared memory, and thirdly, it could be changed to mitigate errors to be within the required threshold. In this algorithm, to automatically register aerial images, a camera in which the camera model is known, and a fixed coordinate system independent of the direction of flight to represent the image, focal and projection planes is used. Global position system (GPS)/Instrument navigation system (INS) data in conjunction with digital elevation maps (DEM) are used, to perform accurate orthorectifiaction and geo registration simultaneously, along with the intelligent selection of parameters, to reduce latency and mitigate errors. The required equations are described in detail, and the logic leading to these equations have been explained extensively.

The modified algorithm targeting a GPU has been fully implemented using a Windows based computing system, using the C language in a CUDA environment. This executable program has been used to generate accurate orthorectified and geo registered maps using many input images. A speedup of over 10x has been achieved over the CPU version. While this proposed methodology has been tested using only aerial photographs acquired from high flying airborne vehicles, it can be extended to Satellite images, as well as, other cases, easily.

Many researchers have studied general image registration [1]-[6], [8]-[21] extensively during the past decades with reviews of the methods provided in [4] [19]. These also give the work done in the areas related to registering aerial images. The original algorithm has been used for various applications and is explained in [1],[3] and [5]. Many others researchers have addressed registration of aerial images [8] and have dealt with orthorectification and georegistration [9, 10] separately, as well as, a composite problem. Sheikh et. al. [12] has proposed orthorectification of aerial images using elevation maps and a camera model. Gabor features of the orthorectified images are detected and normalized correlation is obtained between a reference frame and the current frame. Feature linking local registration and direct registration is performed and an adjustment for the camera model has been incorporated in the proposed method. Three processes for data rectification (orthorectification), establishment of correspondence, and model update are used in this method. Sheikh et. al. [13] has proposed somewhat of a similar method to [12], using elevation maps and feature based correspondence. An iterative method has been proposed to optimize global similarities and a gradient based optimization algorithm has been used.

Xiong et. al. [16] has proposed a Harris corner based approach that does not require establishing any correspondence. Yasien [17] has proposed to compare similarities of the reference and a current image to establish a correspondence and register them to each other.

Many have researched the problem of registering an image frame to another image. In [8] a feature matching to a previous frame and linear optimization method based on the Levenberg-Marquardt optimization algorithm has been

978-1-4799-1365-7/13/$31.00 ©2013 IEEE

explained. In [10] a general method of matching geometric features has been explained for any image that could be used for aerial images also.

In [6], a log polar transform based method to register images to register generic images. The advantage of using log-polar over the Cartesian coordinate representation is that any rotation and scale in the Cartesian coordinates is represented as shifting in the angular and the log-radius directions in the log-polar coordinates, respectively. In [11], a similar log polar transform is combined with the phase correlation to register images. In this proposed method, the scale and the rotation angle between the reference and the sensed image is computed and applied to obtain the registered image.

Due to the limited recovery of the rotation angle by the Fast Fourier Transform and Log polar Transform in the frequency domain, the registration method proposed by Zokai et al. [20] is performed entirely in the spatial domain. This work describes a novel technique to recover large similarity transformations (rotation/scale/translation) and moderate perspective deformations among image pairs. It introduces a hybrid algorithm that features log-polar mappings and nonlinear least squares optimization using the Levenberg–Marquardt algorithm. In this method the translation parameter is recovered by using the coarse-to-fine multi-resolution framework, and the scale and rotation parameters are obtained by matching the log-polar transformed images using a cross-correlation function.

In [18] the characteristics of thin-plate spline, multi-quadric, piecewise linear and weighted mean transformation functions are explored and their performances in the registration of images with nonlinear geometric differences are compared for the purpose of registering two images. The effective use of GPS/INS data for image registration has been shown by Shi et. al. in [15] and a histogram based method has been explained in [16].

II. PROPOSED METHODOLOGY

A. CUDA Programming [7]

Compute unified device architecture (CUDA) is a framework that allows users to map algorithms to Nvidia GPUs and accelerate to lower the latency. In this unified architecture, all the hardware functions of the GPU can be accessed through a single programming model. CUDA compatible GPUs are single instruction multiple data (SIMD) machines. They are capable of accelerating loop bodies that do not have any inter-loop data dependencies. Furthermore, algorithms that are order $O(n^3)$ or more are very suitable to accelerate and higher speedup can be achieved. The programming model is based on initiating multiple hardware threads that execute the same source code kernel on the GPU to leverage the benefits of the SIMD architecture. There is a two level hierarchy to specify threads namely the block and the grid. The size of the block, defines the number of threads assigned to single multi processor (MP) and executed in any stream processor (SP) in that MP. The grid defines the number of blocks that are in an algorithm. Lower latency is achieved by computing the threads in parallel in multiple MPs. It is logical to assign a number of threads that is greater than double the number of SPs in a MP to a MP. When a thread is accessing memory, which is high latency operation, it can execute another thread.

At the highest level of the memory hierarchy is global memory, which is used to communicate with the host memory. Shared memory is at the next level of the hierarchy, and facilitates the communication between SP in a single MP. Local memory is at the lowest level of the hierarchy, and is reserved for local variable within a SP. Variable assigned to the local memory of an SP are not accessible from other SPs in a MP. Arrays that are copied from the host memory are loaded to the shared memory in suitable block size and all the SPs can have access to that particular block of the input array. Basically, all the threads assigned to a MP can access the blocks of memory of each array that is assigned in the shared memory. The main reason to load a block to the shared memory is that accessing shared memory causes lower latency than accessing the global memory. This is one of the advantages of programming in CUDA. By storing arrays in shared memory, global memory accesses are reduced, and consequently overall latency is reduced. However, while there is a very large (in G. Bytes) of global memory, the available shared memory is very limited (~2 MB). Threads assigned to a MP can be synchronized to with each other by using a thread barrier mechanism. At present, there is no mechanism to synchronize the thread in different MPs. Therefore, it is essential that there be no data dependency between blocks.

Coalesced memory writing to the global memory is attempted, where ever possible, to reduce latency. Basically, all the threads in an MP should be synchronized, before writing back the outputs to the global memory.

B. Enhancements to the original methodology

The following enhancements were introduced to the original code proposed in [5]. These were done mainly to mitigate registration errors, improve parallelism and reduce latency.

1. Changed the calculation to determine the approximate size of the pixel in the Earth Coordinates (reduced under sampling)
2. Re-arranged the loops to cache the memory more efficiently (lower latency)
3. Removed inter dependency between loops (for improved parallelism)
5. Fixed computations around edges (to mitigate errors)
4. The offset and the painting value computations were combined to be done in one loop. This reduced memory and shared memory required for the CPU and GPU implementations, respectively.

C. Image Projection

A georectified image has been projected onto an Earth coordinate system which gives a geo-location for each pixel. However, the georectified image can either have a perspective that is not orthogonal to the projection plane, or any height associated with the locations obtained from some elevation map. In general, Orthorectification is the process of projecting

an image so that the perspective view angle changes to be orthogonal to the scene. However, when aerial images are orthorectified, they are typically projected onto an Earth based map and associated (DEM), along with removing the perspective distortions. The result is essential for creating more meaningful data from aerial imagery as it can then be tied to a location.

There are two types of projection used for orthorectification. The first type is forward projection, where an image coordinate is mapped to a projection plane coordinate. Forward projection isn't used widely for orthorectification, especially with aerial imagery because the resulting pixels are not evenly spaced.

Therefore, back projection is used more frequently. For back projection a world coordinate location is projected onto the image plane. The result of back projection is an evenly spaced image. In photogrammetry, the coordinate system, and the associated rotation angle definitions are shown below in Figure 1. The variables at the origin of the coordinate system are the imaging system center denoted by (X_c, Y_c, Z_c).

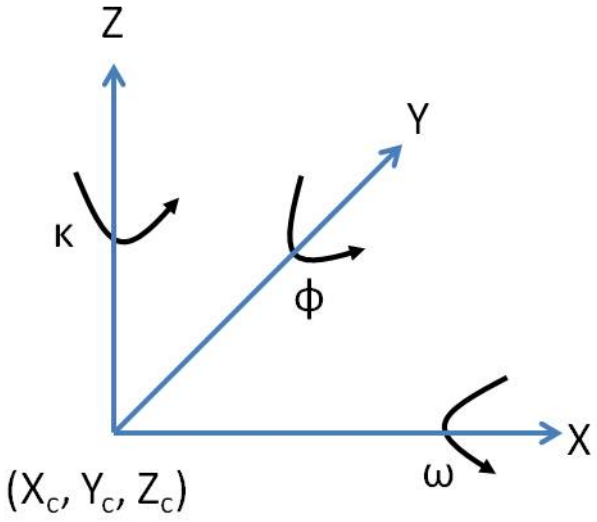

Figure 1. Projection Geometry

The rotations can be multiplied together using the standard rotation matrices, designated at R to form an all encompassing transform matrix, M, and shown in Equation 1. The world coordinate system can be multiplied by M to compute the corresponding image coordinates.

$$M=\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}=R_\omega R_\phi R_\kappa \quad (1)$$

One reason this relationship works is that the imaging system center, the world coordinate, and the corresponding image plane location all are on a line. This relationship can be modeled using the collinearity equations shown in Equations 2 and 3. The image plane indices, i and j, are a function of the camera focal length, f, the world coordinate being projected, (X,Y,Z), the imaging system center, (X_c, Y_c, Z_c), and the individual elements of the transform matrix in Equation 1.

$$i=-f\frac{m_{11}(X-X_c)+m_{12}(Y-Y_c)+m_{13}(Z-Z_c)}{m_{31}(X-X_c)+m_{32}(Y-Y_c)+m_{33}(Z-Z_c)} \quad (2)$$

$$j=-f\frac{m_{21}(X-X_c)+m_{22}(Y-Y_c)+m_{23}(Z-Z_c)}{m_{31}(X-X_c)+m_{32}(Y-Y_c)+m_{33}(Z-Z_c)} \quad (3)$$

C. Iterative Algorithm Description

The following equations are based on the methodology described in [5]. Before the orthorectification begins, the camera is calibrated, which determines the focal length and initial extrinsic camera parameters. Once an image is captured with the corresponding altitude and GPS location, the extrinsic parameters can be computed and correspond to the transform matrix elements.

The first step in the orthorectification process is to determine the region of interest (ROI) of the pre-loaded elevation map. In this step, the corners of the image plane (the image footprint) are forward projected onto the elevation map. The ROI is then used to estimate the required pixel size, to reduce the loss of data during the projection process. The pixel size estimated, determines the interpolation factor required for the rest of the orthophoto creation. As Figure 2 shows, the DEM has a relatively coarse sampling. From the coarse sampling, the ROI requires a finer sampling. The finer sampling is accomplished using a bilinear interpolation method.

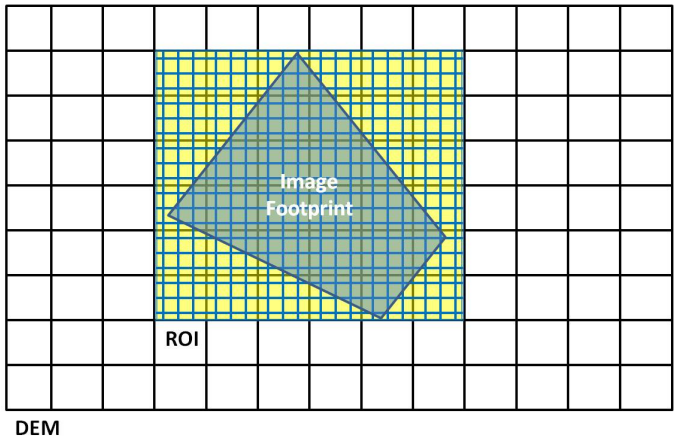

Figure 2. Relationship between the Elevation map, Region of Interest (ROI) and Image footprint.

Figure 2 shows the relationship between DEM, the ROI, and the image footprint. The equation used for the computation of the interpolation factor, I_f, is shown in Equation 4. The side of the image footprint nearest the imaging system is measured in terms of the DEM sampling, giving a distance, r, in meters. The distance is then divided by the number of image pixels along the corresponding side, N_r. One issue is that the interpolation factor determined from the nearest side can lead to over sampling and aliasing artifacts along the furthest footprint side. To mitigate the aliasing

artifacts at the opposite side of the footprint, is to scale the interpolation factor to a slightly higher value, in this case a factor 1.2 is empirically determined to be the most suitable.

$$I_f = 1.2\frac{r}{N_r} \quad (4)$$

The collinearity equations shown in Equations 1 and 2 are ideal for parallelization as each resulting pixel location is independent of the other pixel locations. However, when performing the computations, lower memory accesses typically reduce the computational latency. To reduce the amount of memory accesses, all of the interpolated pixels, within a single DEM pixel are projected, before a new set of elevation map pixels are loaded.

The collinearity equations are modified as shown in Equations 17 and 18. The distances are calculated for each interpolated location within the ROI in all three directions, D_x for the distance from the imaging system in the horizontal direction, D_y for the distance in the vertical direction, and D_z for the change in altitude. The interpolated indices are indicated using the subscript *i*.

$$i[x_i, y_i] = -f\frac{m_{11}D_x[x_i] + m_{12}D_y[y_i] + m_{13}D_z[x_i, y_i]}{m_{31}D_x[x_i] + m_{32}D_y[y_i] + m_{33}D_z[x_i, y_i]} \quad (17)$$

$$j[x_i, y_i] = -f\frac{m_{21}D_x[x_i] + m_{22}D_y[y_i] + m_{23}D_z[x_i, y_i]}{m_{31}D_x[x_i] + m_{32}D_y[y_i] + m_{33}D_z[x_i, y_i]} \quad (18)$$

The input image, camera model, the entire elevation maps and ROI coordinates are passed as parameters. The output is the orthorectified image. The psuedocode for the current implementation on the CPU is shown below.

```
for y_north ← 0, max_pixels_North -1 //of the elevation map ROI

  Calculate initial D_y
 for x_east ← 0: max_pixels_East -1 //of the elevation map ROI
  - Compute the offset of the starting pixel of each block
  Reset D_y
  Calculate initial D_x
  Load DEM pixels
 for γ_north ← 0:INTERPOLATION_FACTOR -1
    Reset D_x
    Calculate initial D_z
  for χ_East ← 0: INTERPOLATION_FACTOR -1
       Calculate i
       Calculate j
       Compute the corresponding intensity value for painting
       Increment D_x, D_z
  endfor χ;
  Increment D_y
  endfor γ;
 endfor x;
 endfor y;
```

IV Implementation and Experimental results

A. Multi threaded algorithm

The iterative algorithm given above was multi threaded targeting a CUDA environment. Note that the inter loop data dependencies have been removed and the code is parallelizable using multiple threads.

The computations for each DEM location can be parallelized for a square window of INTERPOLATION_FACTOR size on each side (width and height). This can be used as the block size to spawn that many threads in the CUDA environment. The resources on the GPU do not support a very large INTERPOLATION_FACTOR if it is used to generate the number of threads. A constant multiplier factor was considered. This was also not attractive as the INTERPOLATION_FACTOR was a variable and could be any value upto 255. Therefore, (INTERPOLATION_FACTOR x1) threads were used in the CUDA environment, instead of the possible (INTERPOLATION_FACTOR x INTERPOLATION_FACTOR). This results in a loop in the kernel, which executes INTERPOLATION_FACTOR number of loops. Note that the maximum number of thread on a processor has been increased to 32678 from 768, for GTX 6xx series (Kepler) GPUs and later. However, to ensure compatibility with older cards this condition has been maintained in the present implementation. The grid size was set to the size of the ROI of the elevation map in the East and North direction respectively. The input image, and the elevation map were copied to the GPU (global) and memory was allocated to return the resulting image. Some other required parameters, such as, the camera model, the GPS coordinates of the camera, were also passed to the GPU.

The strategy that has been used to map the code on the GPU is to use the shared memory and the local memory, as much as, possible. The 2x2 array for the localized DEM values, the D_E are allocated in shared memory. D_N is allocated an array of size INTERPOLATION_FACTOR in shared memory. All the other variables are assigned to local memory. This memory allocation enables registration with even larger interpolation factors to be computed on the GPU. These variables are accessed by all the threads assigned to a MP. The pseudo code of the kernel running on the GPU is given next.

```
int k= threadIdx.x; //range  0:interpolation_factor-1

 if(k==0)
    {
   Reset Dy
   Calculate Dx and save in shared memory
   Load DEM pixels and save in shared memory
    }
__syncthread(); //Synchronize all the threads
      Reset Dx
      Calculate initial Dz
for  χEast  ← 0: INTERPOLATION_FACTOR -1
         Calculate i;
         Calculate j;
         Compute the corresponding intensity value for painting
         Increment Dx, Dz;
endfor χ;
```

The offset of the resultant image where the painted value is store is computed in the kernel. This offset is checked if it is within the borders of the resultant image. If it is within, the paining value is stored. Since every MP computes different offsets to the resulting image there is no possibility of coalesced memory writes. Basically, there is not need for all the threads to be synchronized before image saving.

The calling routine running on the CPU is as follow. First, the memory is allocated on the device for the image data, the elevation map and the results. The elevation map is copied to the device only once in many registration image calls. Second, the image and the elevation map are copied to the device. Third, the number of threads that must be initialized on the block and the grid are set and the device based registration function is called. Fourth on the return, the results are copied from the device the host. Note that the size of the memory of the image, the elevation map and the results are different. They are not the same size in any give scenario.

```
//Allocate memory for the image, elevation map
//and the results and copy the arrays.

cudaMalloc((void**) &d_img_data, img_mem_size);
cudaMalloc((void**) &d_terrain_data,terrain_mem_size);
cudaMalloc((void**) &d_results, results_mem_size);

cudaMemcpy(d_img_data, img_data, img_mem_size,
cudaMemcpyHostToDevice);
cudaMemcpy(d_terrain_data, terrain_data, terrain_mem_size,
cudaMemcpyHostToDevice);

dim3 block_threads(interpolation_factor, 1);
dim3 grid(max_pixelsEast, max_pixelsNorth);
//Call the GPU routine
 registration_gpu<<<grid, block_threads>>> (d_img_data,
d_terrain_data, d_results, other_params);
//Copy the results to host
cudaMemcpy(gpu_results, d_results, results_mem_size,
cudaMemcpyDeviceToHost);
```

B. Experimental Results

Various tests were conducted using different input images and were registered. In one test, an image of size 9000x9000 of characters and a terrain of size 7202x4501 floats were used. Using a ROI of 162x127 and an INTERPOLATION_FACTOR of 81, resulted in an image of size (162x81)x(127x81), which is 13122x10287. This implementation and the CPU based implementation were compared for latency. The latency for the CPU version was 4.15 seconds and the GTX 580 GPU version, the latency was 0.4 seconds. This gave an average speedup of 10.3x over the CPU version running on a 3.4 GHz Quad core Xeon based machine.

V Conclusion

This paper gives extensive details of the proposed implementation. The tests conducted seem to prove that the GPU version is well suited for robust aerial image registration. The proposed GPU targeted implementation of the methodology of this paper is effective in reducing the latency for aerial image registration function. More testing is required to ascertain the robustness for various input images, and another version should be developed targeting the Kepler card and in OpenCL as future works.

REFERENCES

[1] Douglass A. Alexander, Robert G. Deen, Paul M. Andres, Payam Zamani, Helen B. Mortensen, Amy C. Chen, Michael K. Cayanan, Jeffrey R. Hall, Vadim S. Klochko, Oleg Pariser, Carol L. Stanley, Charles K. Thompson, and Gary M. Yagi. "Processing of Mars Exploration Rover imagery for science and operations planning," *Journal of Geophysical Research*, vol. 111, 2006.

[2] Emmanuel Christophe, Julien Michel, and Jordi Inglada. Remote Sensing Processing: From Multicore to GPU. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol 4, issue 3:643–652, September 2011.

[3] Kaichang Di and Rongxing Li. CAVHOR camera model and its photogrammetric conversion for planetary applications. *Journal of Geophysical Research*, vol. 109, 2004.

[4] A. Goshtasby. *2-D and 3-D Image Registration for Medical, Remote Sensing, and Industrial Applications,* New York, Wiley Press, 2005.

[5] Veljko M. Jovanovic, Michael M. Smyth, Jia Zong, Robert Ando and Graham W. Bothwell, "MISR Photogrammetric data reduction for geophysical retrievals," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 36, issue 4, pp 1290-1301, July 1998.

[6]Rittavee Matungka, Yuan F. Zheng, Robert L. Ewing, "Image Registration Using Adaptive Polar Transform," *IEEE Transactions on Image Processing*, Vol. 18, no. 10, pp 2340-2354. Oct. 2009.

[7] Nvidia Corporation. "CUDA C Programming Guide." http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html [May 12, 2013].

[8] Clark F. Olson, Adnan I. Ansar, Curtis W. Padgett. "Robust Registration of Aerial Image Sequences," in *Advances*

in Visual Computing Lecture Notes in Computer Science, Volume 5876, G Bebis et. al.(Eds), Berlin Heideleberg: Springer, 2009, pp 325-334.
[9] Clark F. Olson. "Image registration by aligning entropies." in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Volume 2. pp 331-336, Dec. 2001.
[10] Clark F. Olson. "A general method for geometric feature matching and model extraction." *International Journal of Computer Vision*. Volume 45, Springer, 39-54, 2001.
[11] Jignesh N. Sarvaiya, Suprava Patnaik, Kajal Kothari. Image Registration Using Log Polar Transform and Phase Correlation to Recover Higher Scale, *Journal of Pattern Recognition Research* 7, 2012, pp. 90-105.
[12] Yaser Sheikh, Sohaib Kahn, Mubarak Shah. "Feature-based geo registration of aerial images", Anthony Stefanidis, Silvia Nittel (Eds), in *Geosensor Networks*, Boca Raton, Florida, USA, CRC Press, 2004, pp 112-138.
[13] Yaser Sheikh, Sohaib Kahn, Mubarak Shah, Richard Cannata. Geodetic Alignment of Aerial Video Frames, Mubarak Shah Rakesh Kumar (Eds), in *Video Registration, Video Computing Series*, Kluwer Academic Publishers, 2003. pp 144-179.
[14] Juan Shi, Jinling Wang, Yaming Xu, ″Use of GPS/INS observations for efficient matching of UAV images", In *Proceedings of the International Global Navigation Satellite Systems Society IGNSS Symposium*, 2011.
[15] H. Tuo, L. Zhang, Y. Liu. Multi sensor aerial image registration using direct histogram specification. in *Proceedings of the IEEE International Conference on Networking, Sensing and Control.* 2004, pp 807-812.
[16] Y. Xiong, F. Quek. Automatic aerial image registration without correspondence. in *Proceedings of the 4th International Conference on Computer Vision Systems.* 2006.
[17] M. S. Yasein, P. Agathoklis. "A robust, feature-based algorithm for aerial image registration," in *Proceedings of the IEEE International Symposium on Industrial Electronics*. 1731-1736, 2007.
[18] L. Zagorchev and A. Goshtasby, "A comparative study of transformation functions for nonrigid image registration," *IEEE Trans. Image Processing,* vol. 15, no. 3, pp. 529-538, 2006.
[19] Zitova, J. Flusser. "Image registration methods: A survey," in *Image and Vision Computing*, Volume 21, J. M. Frahm, M. Pantic et. al.(Eds), Elsivier, 2003, pp 977-1000.
[20] S. Zokai and G.Wolberg, "Image registration using log-polar mappings for recovery of large-scale similarity and projective transformations," *IEEE Trans. Image Processing*, vol. 14, no. 10, pp. 1422–1434, Oct. 2005.
[21] S. Zokai and G.Wolberg, "Robust Image Registration Using Log-polar Transform," In *Proceedings of the IEEE International Conference on Image Processing*, September, vol. 1, pp 493-496, 2005.