VIRTUAL PROTOTYPING OF FAST AREA BASED IMAGE STITCHING ALGORITHM

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Lakshmi Kalyani Mudragada

December 2019

VIRTUAL PROTOTYPING OF FAST AREA BASED IMAGE STITCHING ALGORITHM

Lakshmi Kalyani Mudragada

Thesis

Approved:

Advisor
Dr. Kye-Shin Lee

Committee Member
Dr. Ryan Christopher Toonen

Committee Member
Dr. Nathan Ida

Accepted:

Department Chair
Dr. Robert Veillette

Dean of the College
Dr. Craig Menzemer

Acting Dean of the Graduate School
Dr. Marnie Saunders

Date

ABSTRACT

This work presents a virtual prototyping design approach for a fast area-based image stitching algorithm. The structure of virtual hardware obtained from virtual prototyping corresponds to that of the conceptual algorithm; that is, the actual circuit components including the memory, logic gates, and arithmetic units are linked to the conceptual algorithm blocks. Using the proposed method, the overall structure, hardware size, and computation time of the actual hardware can be estimated at the early design stage. As a result, the virtual hardware facilitates the hardware implementation by eliminating trial design and redundant simulation steps to optimize the hardware performance. To verify the feasibility of the proposed method, the virtual hardware of an image stitching algorithm is realized, and the hardware size and the computation time are estimated. In addition, the image stitching algorithm uses binary image comparisons and image scaling, which can eliminate huge multipliers and adders in the actual hardware. Thus, the proposed algorithm can lead to reduced hardware size and faster computation time. Results show that, with a clock frequency of 250 MHz, the estimated computation time of the proposed virtual hardware is 0.877sec to stitch two 1280×1024 images, which is one-tenth of the time required by the software-based image stitching algorithm realized in MATLAB. The estimated size of the virtual hardware showed memory size of 66.4 Mbits and gate count of 874 gates.

ACKNOWLEDGMENTS

I express my heartfelt gratitude to Dr. Kye-Shin Lee, a wonderful advisor who has been the guiding force behind this research work. I express my sincere gratitude to Dr. Nathan Ida and Dr. Ryan Christopher Toonen for their insightful corrections and suggestions. This research would not have been possible without the help and support from the Department of Electrical and Computer Engineering of the University of Akron. I would like to thank them very much. I wish to dedicate my research to the most important people in my life, my husband Chaitanya Borra and my parents Veerababu Mudragada and Saroja Mudragada and without whose support and encouragement I would not have made it this far. Finally, I want to thank my friends who have always been there for me.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

1.1. Background and motivation

Image stitching is a technique for combining multiple images into one single image captured from different cameras (sources) to generate a panoramic image. Image stitching can reduce redundant information in various sets of images, increase the image storage capability, and enable more effective views of the real world [1]. Due to the numerous advantages, image stitching is widely applicable for smart cars, 3D mapping, defense systems, and medical imaging [2] – [4]. However, due to the intensive computing, most of the image stitching hardware has been implemented with extremely high-cost servers or high-performance multi-GPU/DSP platforms which require enormous computing and power usage [5] – [8].

On the other hand, the demands on customized image processing hardware that can perform more specific tasks are increasing due to emerging applications such as unmanned vehicles, remote sensing, and mobile computing, where the memory size, computational resources, and power are all limited [9]. That is, specific hardware implementation is needed when general purpose computers are not suitable due to constraints on speed, size, and energy consumption. However, while image stitching platforms can be easily implemented on general purpose computers, hardware implementation imposes many

challenges due to the contradiction between limited hardware resources and the requirements for high performance. As a result, it is still challenging to realize image stitching hardware for embedded real-time applications. Another difficulty that limits the hardware implementation is that parameters such as computation speed, power, and area cannot be estimated at the initial design stage. The performance estimation is only possible after the completion of the gate or transistor level implementation. Although there are open source hardware description languages such as VHDL that support general image processing algorithms, it is a time-consuming job to convert the VHDL code into the circuit level and run a series of simulations to figure out the hardware performance. To make matters worse, a series of simulations should be repeated several times to end up with an optimized hardware performance. Therefore, in order to extend the applicability of the hardware-based image stitching platforms, there is an urgent demand for finding new hardware friendly algorithms and new design approaches that can facilitate the hardware implementation.

1.2. Contribution of the thesis

1.2.1. Fast area-based image stitching algorithm

In this work, an algorithm for fast area-based image stitching is proposed. This algorithm uses binary images rather than a RGB or grayscale images. The computation time of the proposed algorithm by using binary images is far less than the algorithm using grayscale images. In addition, the image size is reduced by image scaling to further reduce the computation time while retaining the information of the image.

1.2.2. Virtual prototyping of the proposed algorithm

Furthermore, a virtual prototyping design approach is proposed for the fast area-based image stitching algorithm. With the proposed approach, the overall structure, size, and computation speed of the actual hardware are estimated at the initial design stage. As a result, the optimized virtual hardware facilitates the hardware implementation by eliminating complex design and redundant simulation steps. The performance of the proposed virtual hardware is compared with that of the corresponding software-based image stitching algorithm.

1.3. Organization of the thesis

Chapter 2 reviews the literature on image stitching and hardware implementation approaches. Chapter 3 presents details in the proposed algorithm for fast area-based image stitching. Chapter 4 documents the virtual prototyping of the proposed area-based image stitching algorithm. Chapter 5 compares the computation time between the virtual hardware and the MATLAB simulations. Chapter 6 concludes and discusses future work.

CHAPTER II

LITERATURE REVIEW

This chapter describes previous work on area-based image stitching, feature-based image stitching, and hardware implementation approaches, as well as other approaches that perform image stitching, their applications, and drawbacks.

2.1. Area-based image stitching

2.1.1. Pixel-by-pixel alignment

To use area-based method, a suitable error metric must first be chosen to compare the images. Once this has been established, a suitable search technique must be devised. The simplest technique is to exhaustively try all possible alignments, that is, to do a full search. The main benefit of this technique is that it lessens the sum of absolute differences between overlying pixels. This method is scale variant and rotation variant. Pixel-based image stitching method optimally uses the information obtained from the image alignment. It measures the role of every picture element in the image. The main limitation of this technique is the limited range of convergence. Many approaches to compare two images in pixel-based approach are developed [10]-[14].

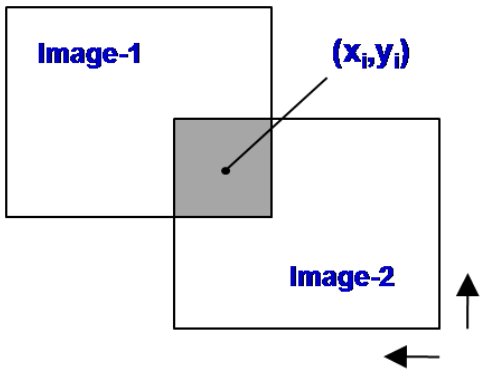


Figure 2.1: Concept of pixel-based image matching.

Figure 2.1 shows the concept of the pixel-based image matching scheme [15]. In order to find the best alignment between the two images, image-2 is shifted relative to image-1 by one pixel in the horizontal and vertical direction, and the average of least square difference for each overlapped region (shaded part) is obtained. That is

$$\text{Average of least square difference} = \frac{\sum_{i,j}\left[I_1(x_{i,}y_{j,}) - I_2(x_{i,}y_j)\right]^2}{\text{Number of Pixels}} \qquad (2.1)$$

where $I_1$and I_2 are the pixel value (luminance) of image-1 and image-2. As a result, the minimum E_{lss} indicates the best matching alignment. The minimum overlap between the two images is found using this error metric.

2.1.2. Correlation method

Template matching is a basic and simple pattern matching technique in digital signal processing, particularly in digital image processing. Template matching is an operation to determine the similarity between two entities, a reference signal, and a target signal.

For some images, template matching uses a reference image (the template), which can be a part of a real image, or for some applications like pattern detection. The pattern is usually smaller than the image. The problem is to find if and where there is an occurrence (or at least a similar enough occurrence) of the template in the target image.

The correlation approach uses the correlation coefficient as a measure of similarity between the reference and the sub-image at each location in the target image. The correlation output is maximum for locations where the template is most similar to the sub-image. If t(x) is the spatial filter or reference, then the correlation at a point x_0 is computed as

$$E_{CC}(x_0) = \sum_i f(x_i)t(x_i - x_0) \tag{2.2}$$

where $f(x_i)$ is the input image and $t(x_i-x_0)$ is the shifted version of the reference image.

If the input f(x) contains a shifted version $t(x-x_0)$ of the reference signal, the correlator exhibits a maximum value at $x=x_0$. Therefore, maximum values occur for the location where the sub-image of f(x) perfectly matches the reference. If the input does not contain the reference t(x), the correlator output is low.

2.1.3. Fourier-based alignment

When the search range corresponds to a significant fraction of the larger image (as is the case in image stitching), a Fourier-based approach may be preferable. The Fast Fourier Transform algorithm can compute the transform of an N × M image in $O(NM \log (NM))$ operations. This can be significantly faster than the $O(N^2M^2)$ operations required to do a full search when the full range of image overlaps is considered. Another

useful property of the Fourier transform is that convolution in the spatial domain corresponds to multiplication in the Fourier domain.

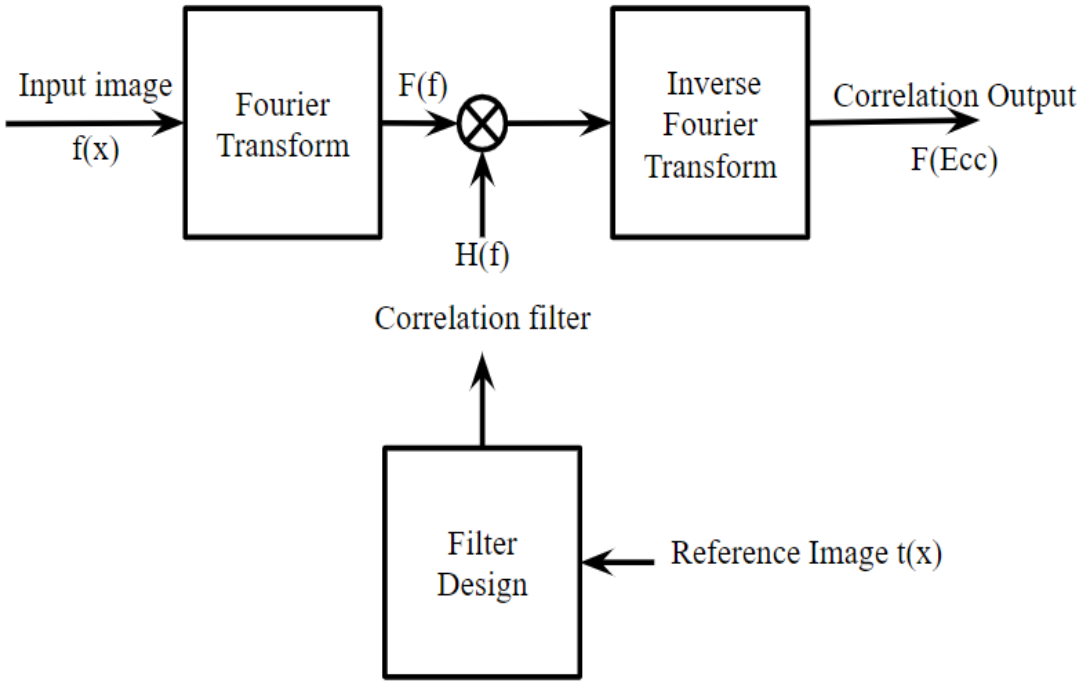


Figure 2.2 Cross-correlation using Fourier transform

Correlation can be also done in the frequency domain as shown in Figure 2.2, as correlation is related to the product of image Fourier transforms by the convolution theorem. Thus, correlation is obtained by multiplying the image Fourier transform with the complex conjugate of the reference Fourier transform. Peak locations (high correlations, which correspond to matches) in the spatial domain are then obtained using the inverse Fourier transform.

2.2. Feature-based image stitching

In the feature-based technique, all main feature points in an image pair are compared with those of every feature in another image by making use of local descriptors. For feature-based image stitching techniques, feature extraction, image registration, and image blending are the various stages followed. Feature-based methods are used by instituting equivalences between points, lines, edges, corners, or other shapes. The main advantages of the feature-based methods include invariance to the noisy image, scale invariance, translation invariance, and invariance to rotation transformations [16]-[17]. Finding the image features is called feature detection. A description of the region around the feature, called a feature description, can then be used to find the same feature in other images. Multiple images that include a common feature can be aligned for stitching or other purposes.

2.2.1. Harris Corner Detector

One early attempt to find the corners was done by Harris and Stephens in their paper “A Combined Corner and Edge Detector” in 1988, so now it is called Harris Corner Detector. The Harris corner method is a mathematical operator that finds features in an image. It is simple to compute and fast [18]. This method is popular as it is rotation, scale and illumination variation independent. In Figure 2.3, even the little movement of the window produces a noticeable difference in the intensity. The blue window shows the flat surface, where any movement does not show any difference in the intensity. The green window shows the edge. For edge the movement in perpendicular direction to the edge is noticeable but the movement along the edge is not noticed. The yellow window shows a

corner. A small movement in any direction can be noticed. Therefore, a corner is said to be a good feature. The process of detecting the corner depends on calculating the sum of the squared differences between the image values in a given window and the corresponding image values in a shifted window [19], as

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \quad (2.3)$$

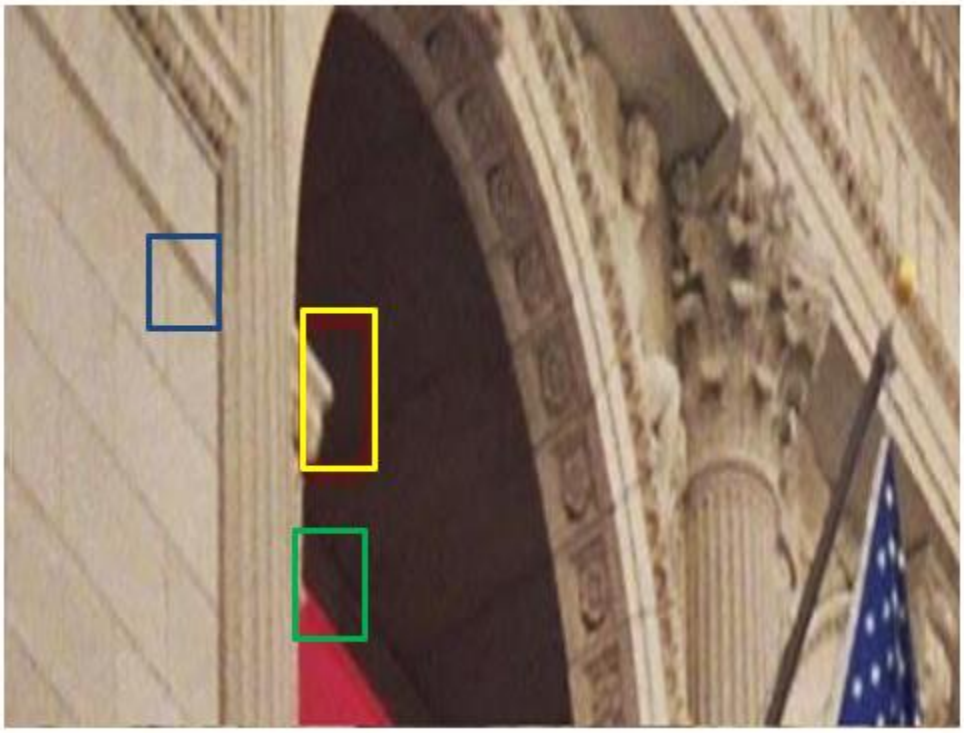


Figure 2.3 Regions with extremely high variation

If a corner is present in the window, then the value of E in equation (2.3) tends to be large. The calculation of E is simplified by using a Taylor series expansion of I. The final equation after applying Taylor series, and ignoring the window function w(u,v), is

$$E(u,v) \approx \sum_{x,y} u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2 \quad (2.4)$$

Writing the above equation into a matrix form

$$E(u,v) \approx [u\ v] \left(\sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.5}$$

It was figured out that eigenvalues of the matrix can help to determine the type of feature seen in a window. For example one may consider the parameter R, defined for a given window as

$$R = det\ M\ -\ k(trace\ M)^2 \tag{2.6}$$

where $\det(\mathrm{M}) = \lambda_1\lambda_2,\ \mathrm{trace}(\mathrm{M}) = \lambda_1 + \lambda_1\ and\ M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$

to decide whether the window has the corner or not. When $|R|$ is small, the window has a flat region. When R < 0, the window has the edge. When the R value is large, the window has the corner.

2.2.2. SIFT method

The Harris corner detector is rotation-invariant but not scale-invariant. The corner may not remain a corner if the image is scaled. To overcome this problem in Harris corner detector method, in 2004 D. Lowe came up with a new algorithm, Scale Invariant Feature Transform (SIFT) [20]-[21]. There are four steps contained in the SIFT algorithm.

a. Scale-space extrema detection

Scale-space filtering is used to detect the large corners since it is obvious that small corners are detected using same window. Laplacian of Gaussian (LoG) is found for the

image with different σ values. To compute the LoG, an image is blurred by convolving with Gaussian functions having different values of variance σ. Then, second order derivatives (or Laplacian) of the image are calculated. But, calculating second order derivatives is computationally intensive, therefore Difference of Gaussian (DoG) is calculated. The DoG is an approximation of LoG. The DoG is obtained by taking the difference between the Gaussian blurring of an image with two different σ. Now the local maxima across the scale and space are found. It gives a list of (x,y, σ) values which signifies there is a potential keypoint at (x,y) at σ value. Figure 2.4 shows the process of DoG in which it is done for different octaves of the image in Gaussian pyramid.

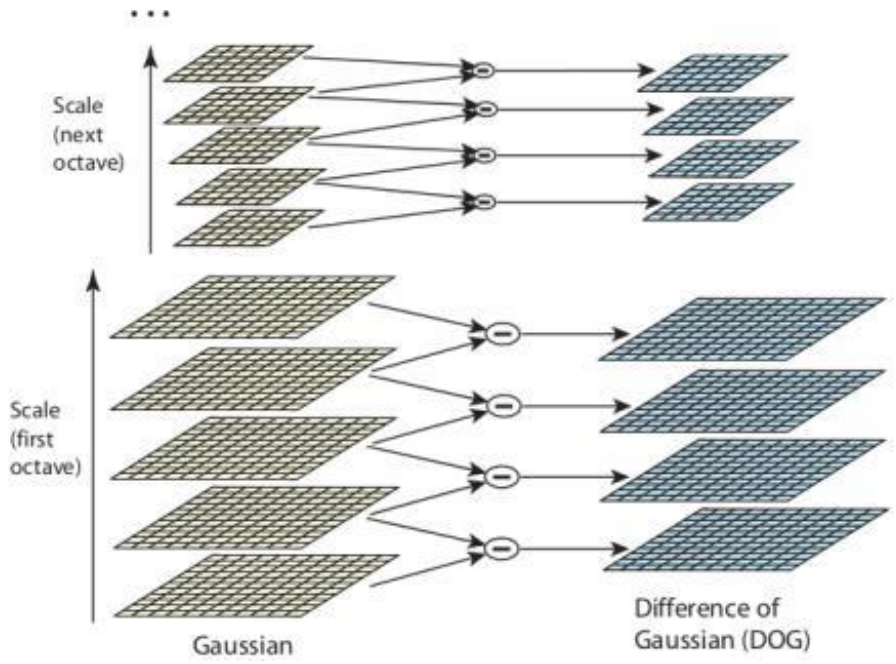


Figure 2.4 Difference of Gaussian (DoG) [21]

After finding the DoG the next step is to locate the local extrema over scale and space. Figure 2.5 shows the process of finding the local maxima/extrema. The symbol X in the middle scale is compared with the 8 neighboring pixels and also with the 9 pixels

in the next and previous scales. It is said to be local maximum if the value of X is greater than all the compared pixels, therefore it is a potential keypoint.

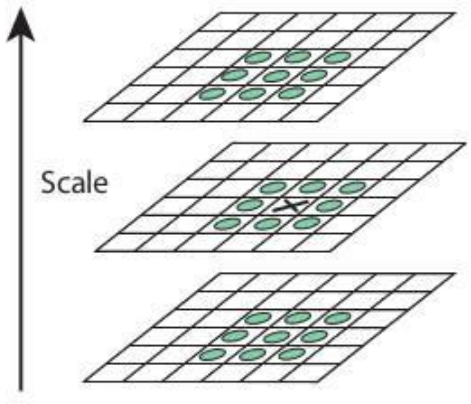


Figure 2.5 Locating maxima [21]

Some typical data can be given as a number of octaves = 4, the number of scale levels = 5, initial σ = 16 and scaling factor k = $\sqrt{2}$.

b. Keypoint localization

After finding the potential keypoint reference, the Taylor series expansion is applied on scale-space to get more exact location of local maxima. If the intensity of this local maxima is less than threshold value, then the potential keypoint is rejected.

The DoG has higher response for edges, therefore edges also need to be removed. To do this, a 2×2 Hessian matrix (H) is used to calculate the principal curvature. So, the images go through two tests: the contrast test and the edge test. Therefore, any low-contrast and edge keypoints are eliminated. A few keypoints are rejected and thus, a lower number of keypoints are left to deal with.

c. Orientation assignment

To achieve invariance to image rotation, an orientation is allocated to each keypoint. To assign orientation, gradient directions and magnitudes are collected around each keypoint. An orientation histogram with 36 bins including 360 degrees is generated. The highest peak in the histogram is considered and any peak above 80% of it is also taken to calculate the orientation. This makes keypoints with the same location and scale but different directions. This contributes to stability of matching.

d. Keypoint descriptor

After the orientation assignment, keypoint descriptor is created. A 16×16 neighborhood around the keypoint is taken. It is divided into 16 4×4 blocks. Now for each sub-block, 8 bin orientation histogram is created; therefore a total of 128 bin values are available.

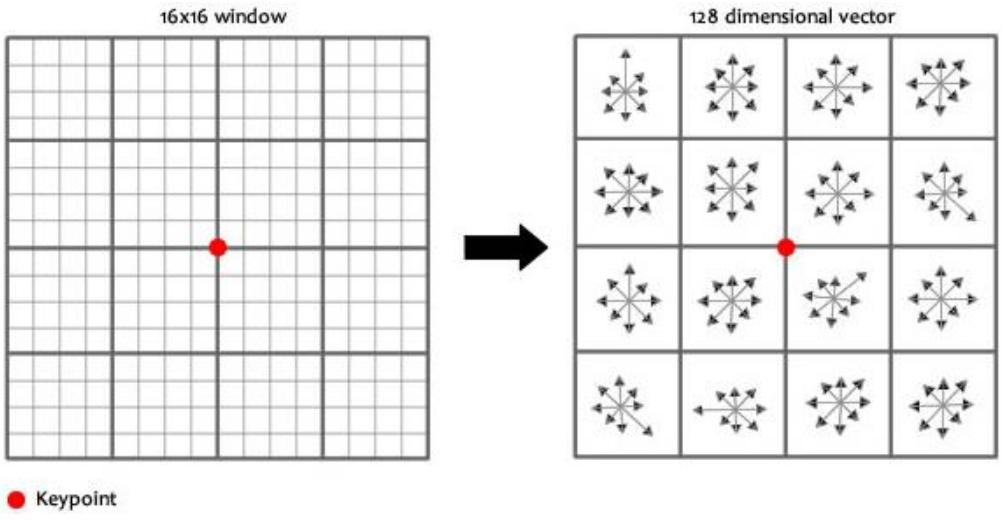


Figure 2.6 16×16 window around the keypoint. This 16×16 window is broken into sixteen 4×4 windows [28].

2.2.3. SURF (Speeded Up Robust Features)

The SIFT method discussed in the last section is comparatively slow, and a faster version is needed. The SURF method goes further and approximates LoG with box filter. Figure 2.7 shows a demonstration of such an approximation. Convolution with box filter can be easy with the help of integral images, which is done in parallel for different scales [22]-[24].

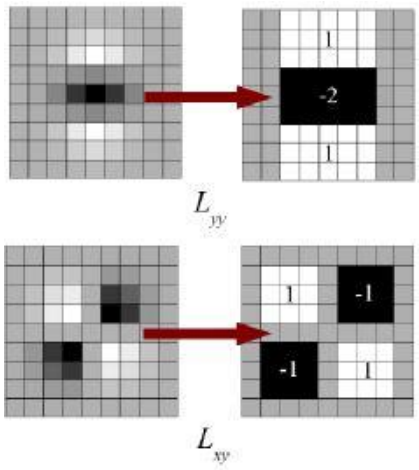


Figure 2.7 Gaussian second derivatives and box filters [22]

For orientation assignment, wavelet responses are used by SURF in both horizontal and vertical directions. Gaussian weights are also applied to it. Figure 2.8 shows how they are plotted in the space. The leading orientation is approximated by calculating the sum of all responses within a sliding orientation window of 60 degrees angle. For many applications, rotation invariance is not required, therefore it is not essential to find this orientation, which speeds up the process. For this process, SURF provides a functionality known as U-SURF or Upright-SURF.

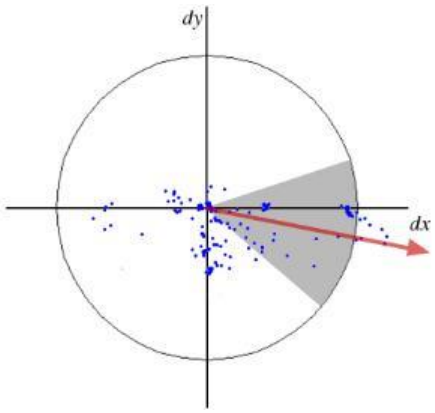


Figure 2.8 Wavelet responses in the horizontal and vertical directions [27]

As it is said SURF uses wavelet responses in the horizontal and vertical direction for the feature description. A neighborhood of size 20s×20s is considered around the keypoint, where 's' is the size. This is further divided into 4×4 sub-regions. As shown in Figure 2.9, for each sub-region horizontal and vertical wavelet responses are considered and vector is formed as

$$v = \sum d_x, \sum d_y, \sum |d_x|, \sum |d_y| \tag{2.7}$$

This vector provides the SURF feature descriptor with total 64 dimensions. If the dimension is low, the speed of computation and matching is high. Therefore, uniqueness of the feature is provided. For more uniqueness, SURF feature descriptor uses and extended 128-dimension version. The sum of d_x and $|d_x|$ are evaluated separately for $d_y < 0$ and $d_y \geq 0$, and same for y-direction. In this approach, the number of features are doubled, and it would not add much computation difficulty.

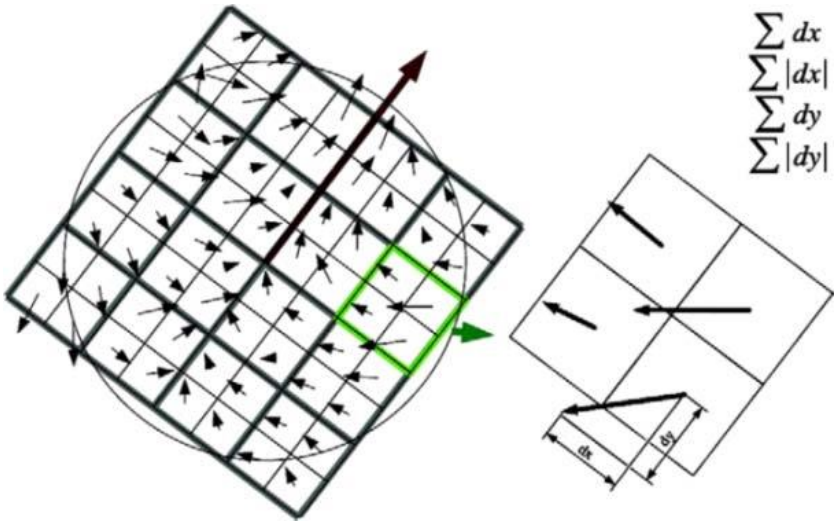


Figure 2.9 Structure breakdown of each feature's neighborhood [24]

Another significant development is the use of the sign of Laplacian for underlying feature point. It does not add computation cost because it is already computed during detection. The sign of Laplacian differentiates light blobs and dark blobs. In matching phase, only features with same type of contrast are compared. Figure 2.10 shows this explanation. The minimal information allows for faster matching without reducing the descriptor's performance.

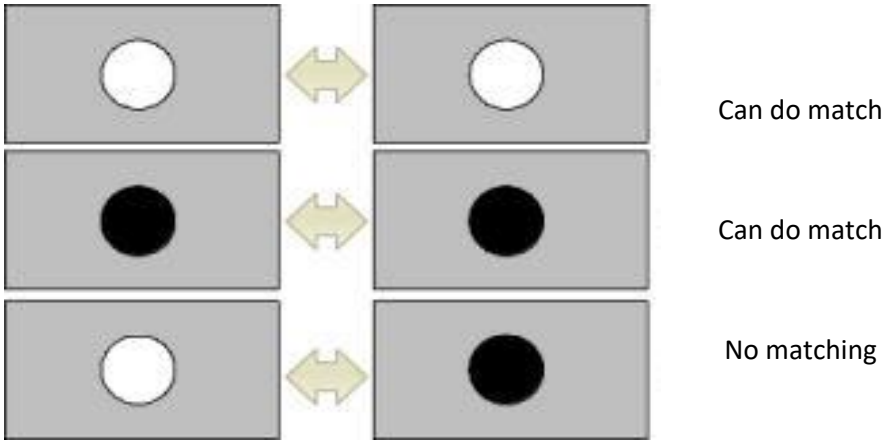


Figure 2.10 Features as light and dark blobs [27]

Therefore, SURF adds a lot of features to improve the speed in every step. It is three times faster than the SIFT algorithm. SURF is suitable at handling images with blurring and rotation, but not suitable at handling viewpoint change and illumination change.

2.3. Hardware description

The FPGA implementation of image registration algorithms is a difficult problem due to the limited sources of the hardware and the necessity for real-time processing speeds. In this section, as a hardware implementation example, the FPGA based I2A (image interpolation approach) algorithm, and the challenges of FPGA-based prototyping are discussed [25].

2.3.1. Example of hardware implementation

For implementing a real-time optical flow sensor an image interpolation approach (I2A) is proposed. The application for this work is the development of a small and lightweight sensor suitable for use on a small unmanned flight vehicle and eliminating affine motion between two images. This can be done by using the translation information between two matching sub-regions of those two images. The translation information between the sub-regions is handled using the image interpolation algorithm.

Figure 2.11 shows the FPGA implementation of the I2A algorithm. Data input is camera data or frames from the previous FPGA. Line and pixel FIFOs consists of external FIFOs, which are used to create the frames. Binary conversion converts 8-bit data into the binary data using the threshold from the adaptive thresholding block. SM (Saliency Model) performs AND, OR and NOT single bit logic operations. Accumulators perform addition

of each SM frame. Accumulator produces six parameters for each patch of maximum of 64×64 sized images. MSD (Mie Scattering Diffusion) performs the optical flow based on the six parameters. The average block calculates the average of the optic flows in x and y directions. These averages of first FPGA which corresponds to S_{x2}, S_{y2} in Figure 2.12 are used to control the external FIFOs in the second FPGA in connection mode. X_{OF} and Y_{OF} of 8-bit width are sent to PC for post processing through RS232 [26].

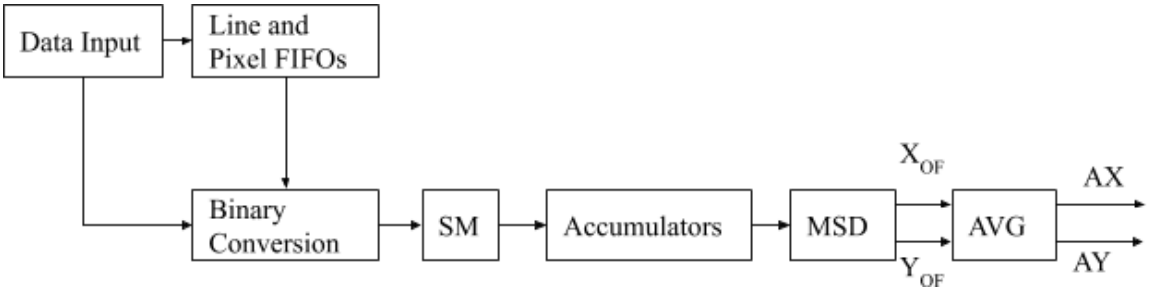


Figure 2.11 Image interpolation algorithm (I2A)

Now an image registration algorithm using binary images used I2A in it as shown in Figure 2.12. It is built on two FPGAs, in which each FPGA is programmed to run the I2A algorithm. The first FPGA is connected to an image sensor. The sensor captures the image and runs the image back to the FPGA. The first FPGA is used to implement the first iteration and second FPGA for second iteration. S_{x1}, S_{y1}, S_{x2}, S_{y2} are inputs of two FPGAs. The four inputs are set to zero when operated individually. S_{x1}, S_{y1} are still zero and S_{x2}, S_{y2} are average optic flows calculated by first FPGA. These inputs are given through extFIFO control block. The extFIFO control block controls the extFIFO block delaying its input by one frame when S_x and S_y are zero. The adaptive threshold block calculates the threshold used to create binary images.

In the I2A block, data input is taken from the camera. There are line and pixel FIFOs used to create the reference frames. There is binary convertor that converts 8-bit data into binary data using the threshold from the block adaptive thresholding. And further a single bit logic operation AND, XOR, and NOT are performed. The accumulators compute six parameters for each patch (64×64 size). And next MSD computes the optic flow based on the six parameters X_{OF} and Y_{OF}. Further average values are taken for these optic flows in x and y directions. These average values are used to control the external FIFO in the second FPGA. X_{OF} and Y_{OF} are 8-bit width which are sent to PC for post-processing thorough RS-232.

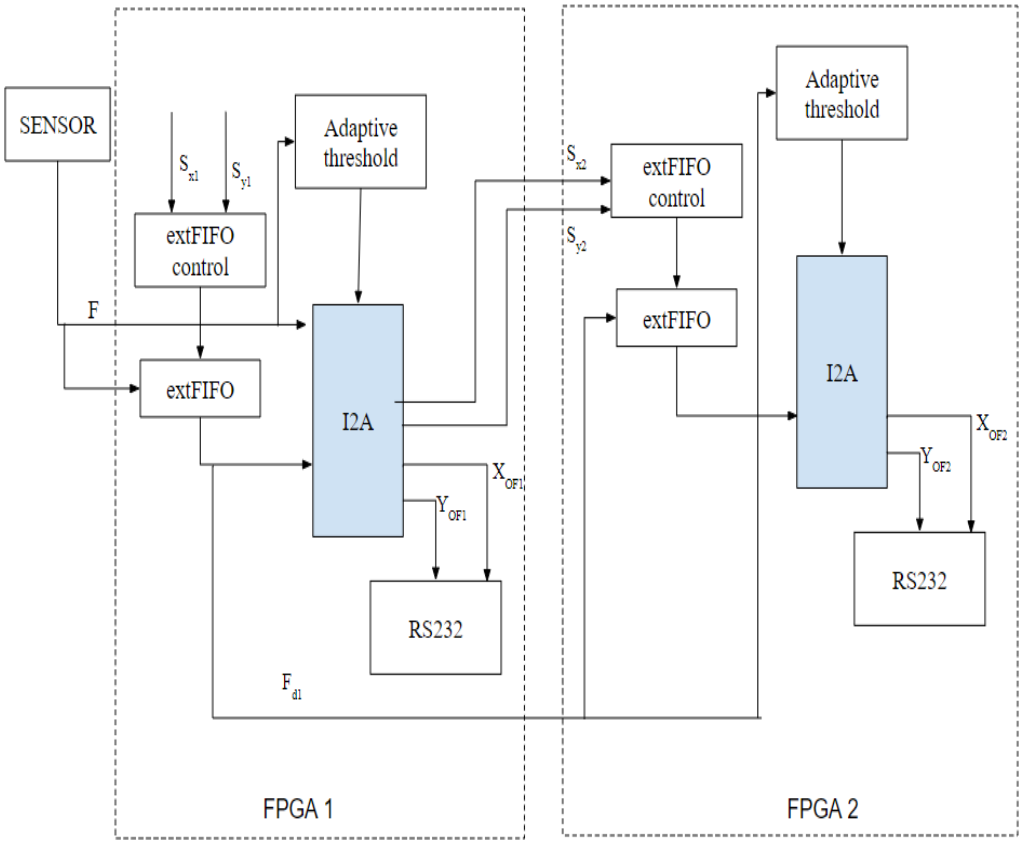


Figure 2.12 The block diagram of multiple FPGA system.

2.3.2. Challenges for FPGA-based prototyping

It is hard to estimate the hardware performance including computation time hardware size and power consumption in the early design stage. This can be figured out after the design optimization stage. The proposed virtual prototyping approach can solve this problem, since the hardware performance can be estimated at the early design stage.

CHAPTER III

PROPOSED AREA-BASED FAST IMAGE STITCHING

This chapter describes the proposed area-based image stitching algorithm using binary images (1-bit pixels), which can speed up the computation by simplifying the operations. The realization of pixel comparison between the two images can be realized with a simple logical operation instead of subtraction and addition.

3.1. Algorithm

Figure 3.1 shows the proposed area-based image stitching algorithm. First, two input images (24bit color) have an overlapping area. Conversion of a color image to a grayscale image requires more knowledge about the color image. A pixel color in an image is a combination of three colors Red, Green, and Blue (RGB). The RGB color values represent in three dimensions XYZ, and they are illustrated by the attributes of lightness, chroma, and hue. The quality of a color image depends on the color represented by the number of bits the digital device could support. The primary color image is represented by 8 bits, the high color image is represented using 16 bits, the actual color image is represented by 24 bits, and the deep color image is represented by 32 bits. The number of bits determines the maximum number of different colors supported by the digital device. If each Red, Green, and Blue occupies 8 bits, then the combination of RGB occupies 24 bit and supports 16,777,216 different colors. The 24 bit represents the color of a pixel in

the color image. Luminance using 8 bits represents the grayscale image. The luminance of a grayscale image pixel value ranges from 0 to 255. The conversion of a color image into a grayscale image is converting the RGB values (24 bits) into a grayscale value (8 bits).

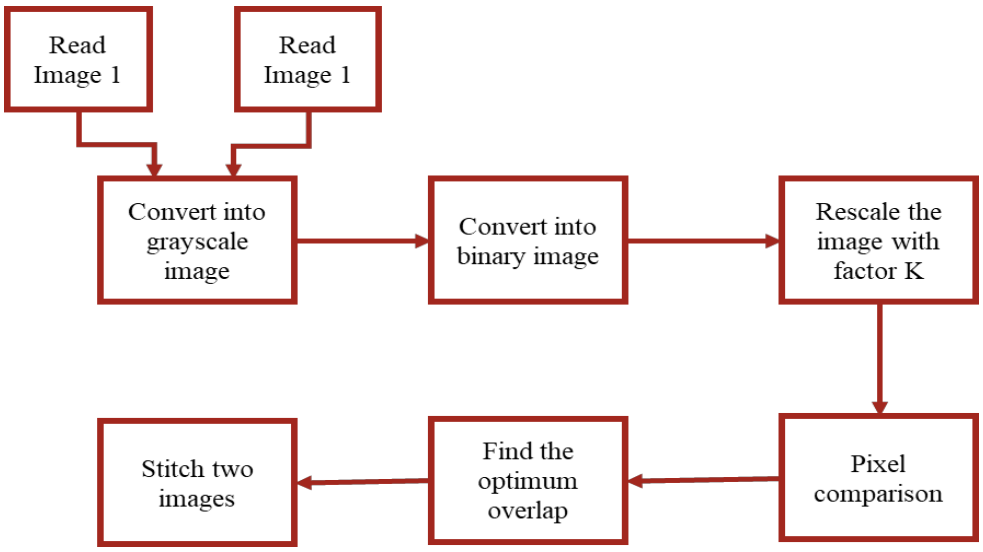


Figure 3.1 Area-based image stitching algorithm

Furthermore, the grayscale (8 bit) image is converted into a binary image. The threshold level – usually the mid-level of the minimum and maximum pixel value and representing the pixel value with either 255 (pixel intensity higher than a threshold) or 0 (pixel intensity less than a threshold), where 255 and 0 stands for the white and black level, leads to the binary image. Later, 255 is replaced with 1 to reduce the image data further. As shown in Figure 3.2, if the threshold is too low (th=20), it eliminates the blob-like objects (some boxes in the top left corner of the image). If the threshold is too high (th=200), a large number of essential information is eliminated (image information at the top of the image and the bottom of the image).

In this case, the mid-level value 128 is optimum for the threshold, since the binary image preserves the basic features of the original image. The dotted red boxes indicate the area of the image that is off-track or redundant data, where for threshold 20 some pixel data is removed in the dotted box and for threshold 200 some extra pixel data is added into top-left corner and some pixel data is missing at the bottom. The threshold level of 128 retains all the information present in the original image. Also, by using binary images, the number of computations to compare every pixel of the two images can be significantly reduced, which enables a fast image stitching.

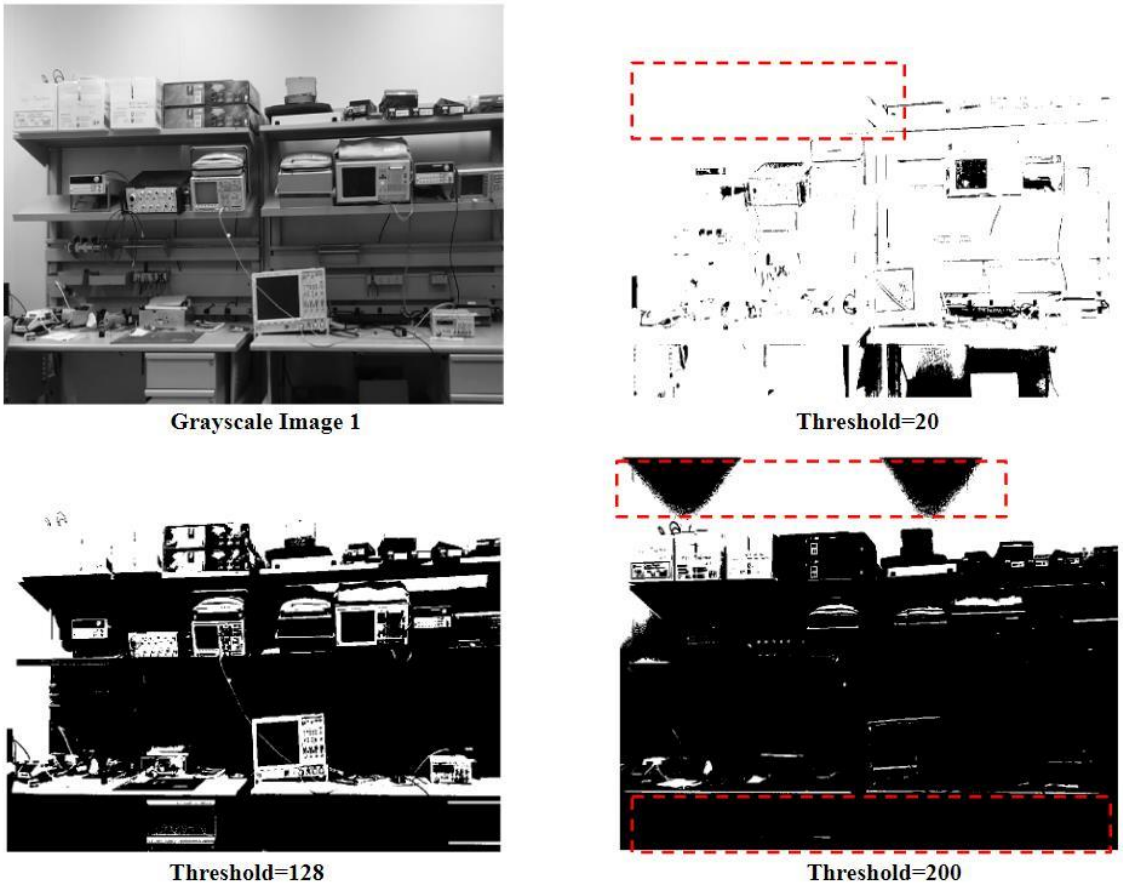


Figure 3.2 Binary images with different threshold levels

Next, rescale the image to reduce the number of computations during the comparison function. In general, image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is the removal of redundant data. The transformation is applied prior to storage or transmission of the image. At some time later, the compressed image is decompressed to reconstruct the original image.

A scaling factor K, which is an integer greater than 1, is used to rescale or compress the image. Different values, like K=2, 4, 8, 16, 32 are been experimented to observe the time taken stitch two images. The main advantage of image rescaling is, as the K value increases, the computation time decreases. Figure 3.3 shows the basic idea of image rescaling. A K×K size pixels are replaced with a single pixel.

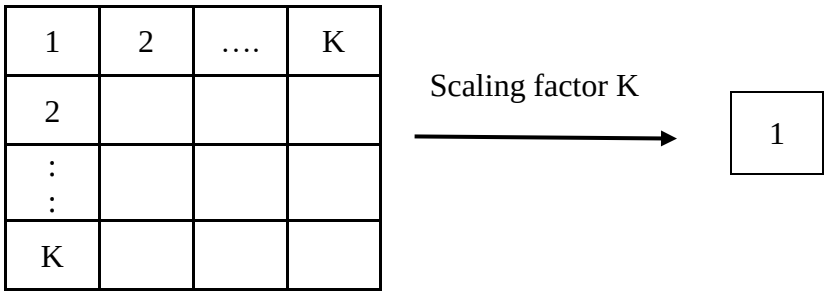


Figure 3.3 Image rescaling

When K=2, 2×2 pixels are replaced by 1 pixel. Figure 3.4 shows an example of image rescaling for K=2. Here 4 pixels are replaced by the most frequently occurring value. When the number of 1's and the number of 0's in the 2×2 pixels are equal as shown in leftmost 2×2 image the value 0 is considered as most occurring pixel. As shown in Figure 3.5 a 4×4 size image is resized to 2×2 where K=2.

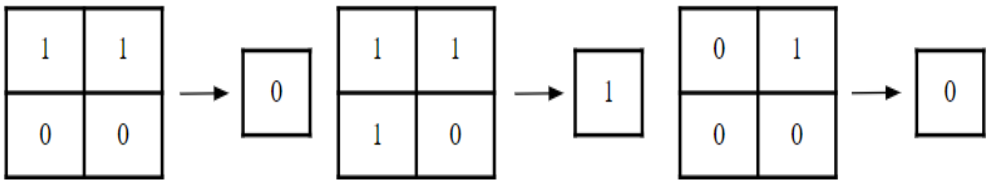


Figure 3.4 Image rescaling concept

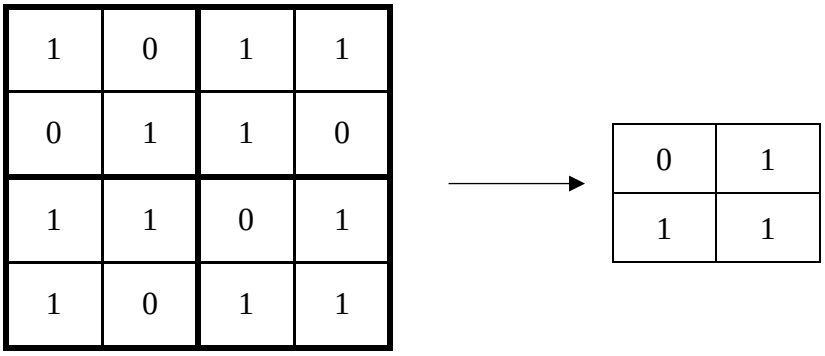


Figure 3.5 Image rescaling for K=2

Figure 3.6 shows the results of a binary image for different K values. It is observed that when K value increases the image size is reduced. For K = 2 and K = 4 the computation time is higher than K = 8. For K = 16 the computation time is very less but the image data is missing due to high K value. Since 16×16 pixels are replaced by 1 pixel, more image information is lost, for which expected results are not achieved. Therefore K=8 is chosen as conclusive value.

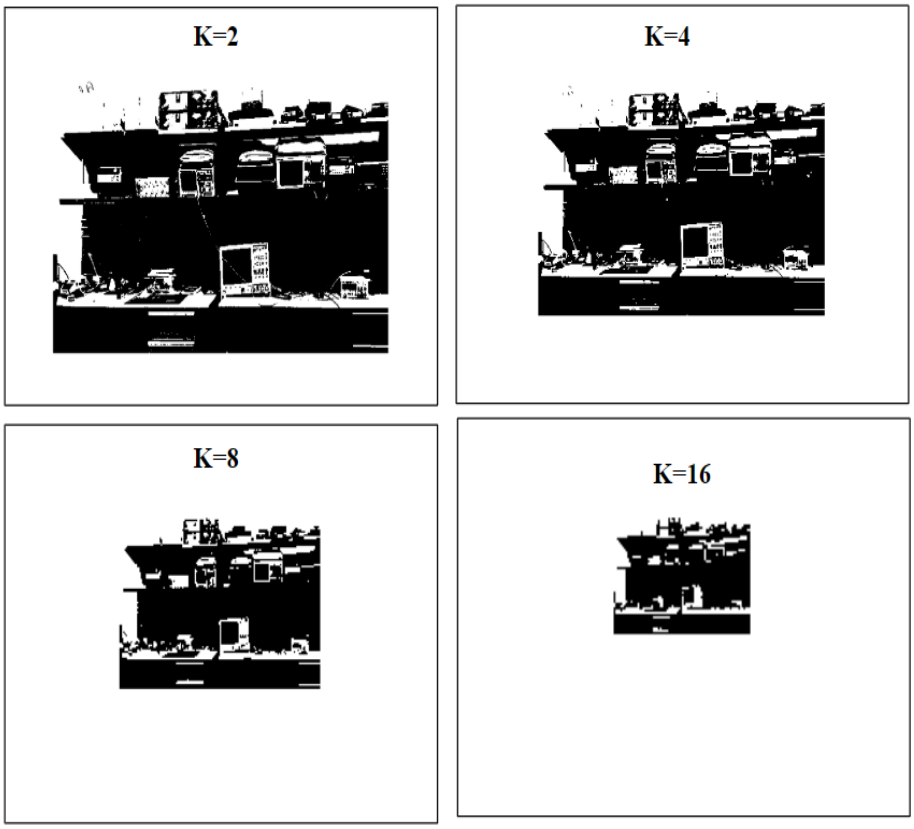
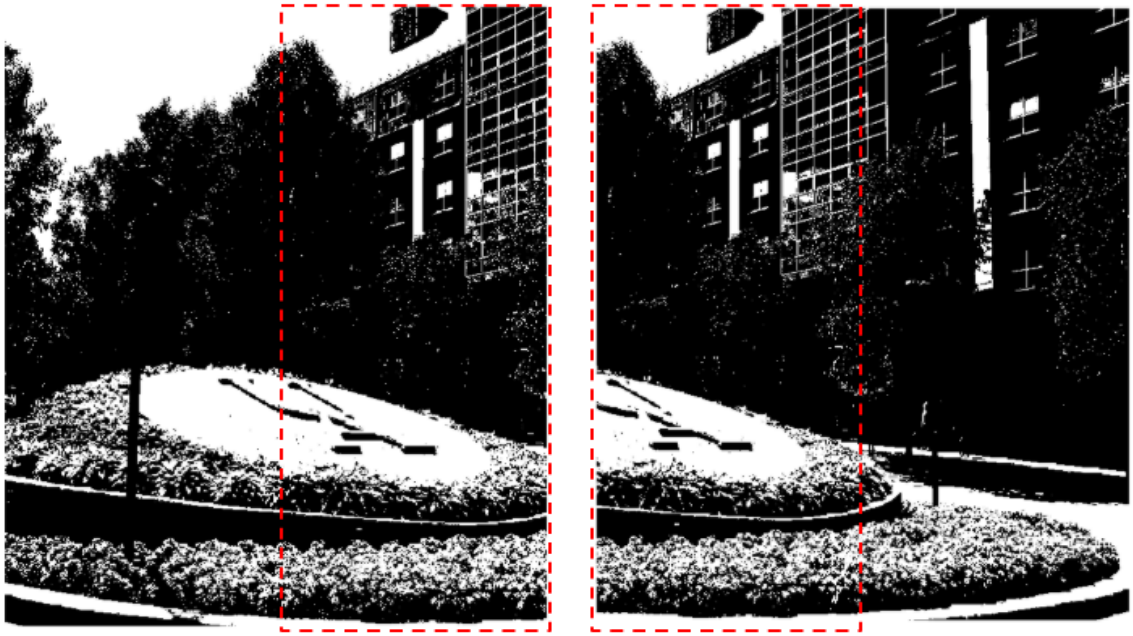


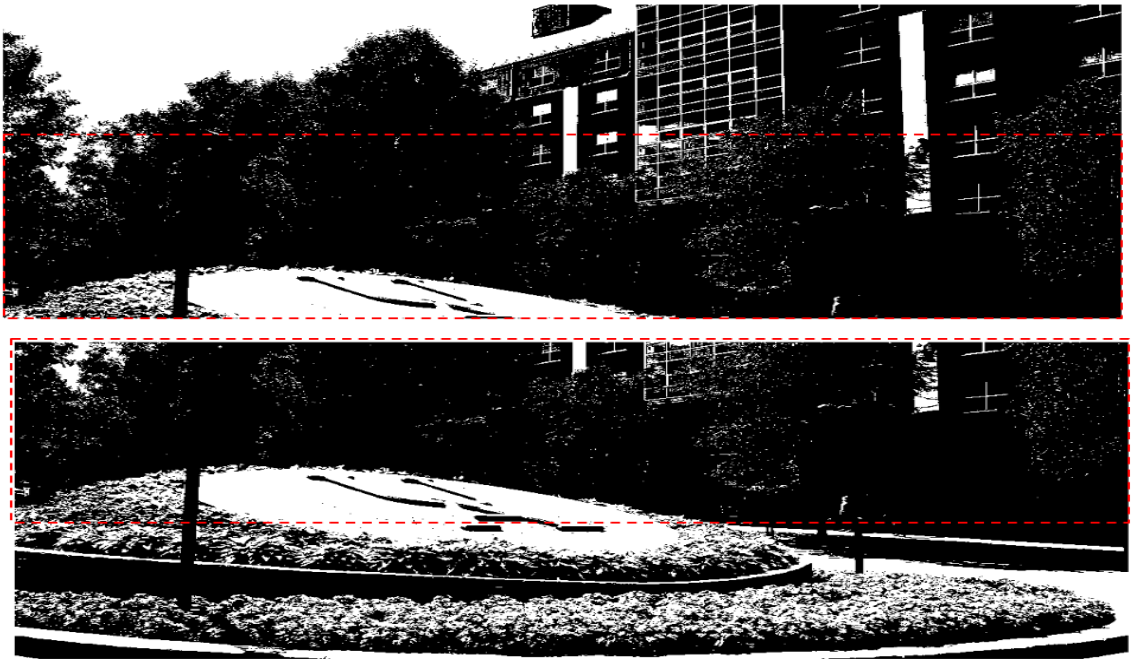
Figure 3.6 Binary images for different K values

3.2. One-dimensional image stitching

Area-based image stitching can be performed in either one dimension or two dimensions. In the one-dimensional approach, one image is being shifted only in the horizontal or the vertical direction to compare the two images. Figure 3.7 shows one-dimensional comparing approaches of two images, where the overlapped areas are indicated by the dotted box. Figure 3.7 (a) shows an example of two images where the comparison of the overlapped area is performed in the horizontal direction and Figure 3.7(b) shows an example of two images where the comparison of the overlapped area is in vertical direction.



(a)



(b)

Figure 3.7 Example of two images with overlapped areas. (a) The overlapped area in the horizontal direction, (b) Overlapped area in the vertical direction

Figure 3.8 shows an example of a one-dimensional approach, where a 3×3 binary image is considered for simplicity. In this example, image-2 is shifted horizontally and it can also shift vertically. During the first iteration, column-3 of image-1 and column-1 in the image-2 are compared. The colored pixels indicate the pixels being compared. The two pixels in the shaded area (each from image-1 and image-2) are compared, which is realized by the XOR operation. The comparison result is 0 when the two pixels match; otherwise, the result is 1. Next, the comparison results are all added up and divided by the number of pixels being compared for the 3 iterations. This value corresponds to the average of least square difference of each iteration, as mentioned in equation 2.1. It is observed that from Figure 3.5, the first iteration average of least square difference is 0.

In the second and third iteration, 6 and 9 pixels (each from image-1 and image-2) are compared. The average of least square difference values of the second and third iteration are 0.66 and 0.55, respectively. After comparing all the pixels, the average of least square difference value for each iteration is again compared, where the iteration with the minimum average of least square difference value leads to the optimum overlap. Therefore, the first iteration leads to the optimum overlap. Figure 3.6 shows the one-dimensional image stitching results based on the optimum overlap. The overlapped area is highlighted in red color. Here the stitching is performed by removing the overlapped area from image-2 (first column) and concatenate the two images.

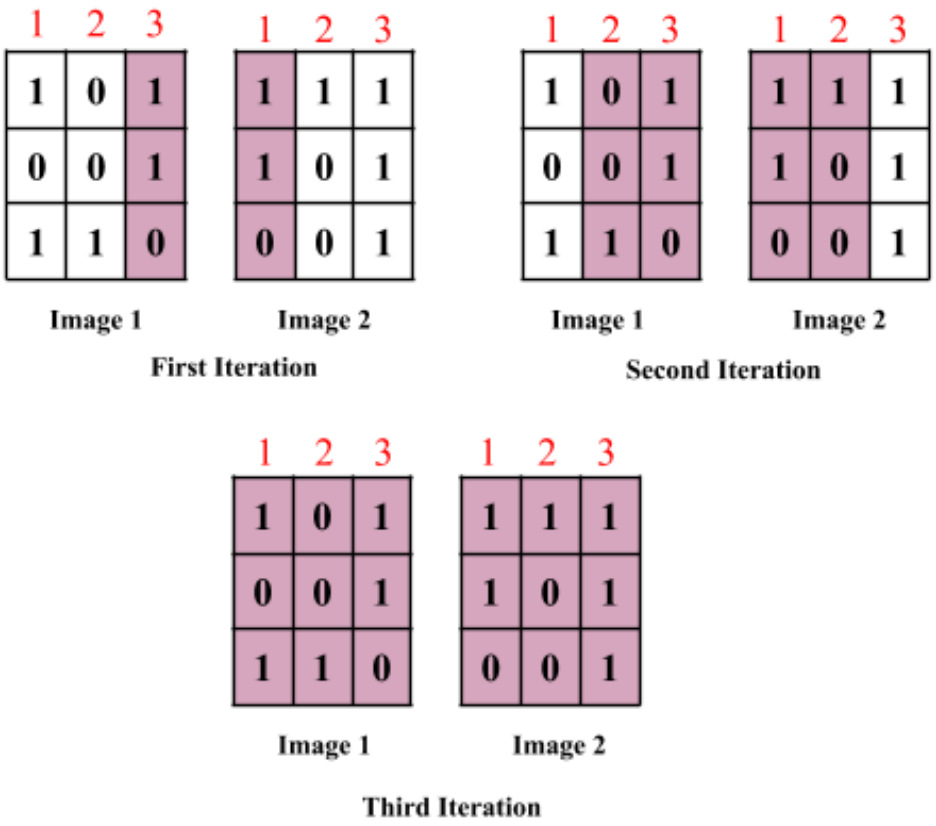


Figure 3.8 One-dimensional image stitching example

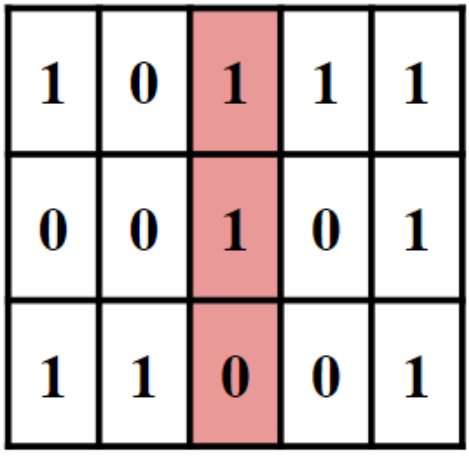


Figure 3.9 One-dimensional image stitching result

3.3. Two-dimensional image stitching

For the one-dimensional image stitching, image-2 is shifted only in the horizontal or vertical direction to vary pixels overlapped area. However, in the two-dimensional image stitching, image-2 is shifted both in the horizontal and the vertical direction to find the optimum overlap. Figure 3.10 shows an example of two-dimensional image stitching. The shaded area in the figure is the overlapped area of image-1 and image-2.

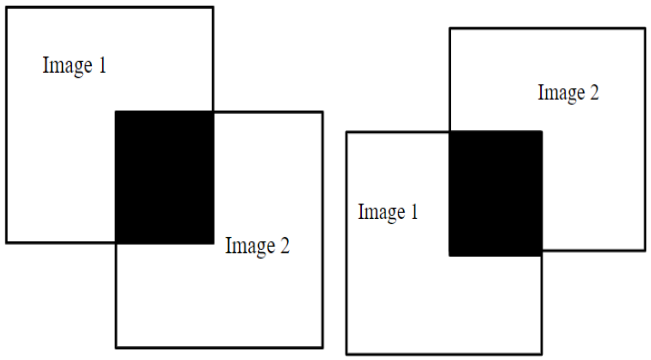


Figure 3.10 Two-dimensional image stitching example

The two-dimensional image stitching is more accurate than the one-dimensional case, though this leads to more complicated computations. The number of pixels compared is more compared to the one-dimensional image stitching approach since pixels are compared in one-to-one. Figure 3.11 shows a two-dimensional image stitching algorithm example using two binary images, where the shaded pixels indicate the pixels being compared. Similar to the one-dimensional case, there is horizontal shift for 3 iterations. In horizontal shift 1 column-1 of image-2 is compared column-3 of image-1, in horizontal shift 2 column-1 and column-2 of image-1 are compared with column-2 and column-3 of image-2, and finally in horizontal shift 3 column-1 to column-3 of image-1 are compared

with column-1 to column-3 of image-2. In this way, the shaded area is compared using the vertical and horizontal shifts.

The average of least square difference values for each area compared for horizontal shift 1 are 0, 1, 0.33, 0.5 and 1, for horizontal shift 2 the values are 1, 0, 0.66, 0.5 and 0 and for horizontal shift 3 the values are 0.33, 0.66, 0.44, 0.5 and 0.66. It is observed that the minimum average of least square difference values is '0' in three places, which corresponds to the vertical shift 1 of horizontal shift 1, the vertical shift 2 of horizontal shift 2 and the vertical shift 5 of horizontal shift 2. However, the optimum overlapped area is the area with the maximum number of pixels that have been compared. Therefore, the vertical shift 2 of horizontal shift 2 shows the minimum average of the least square difference of pixel comparison among the other overlap cases.

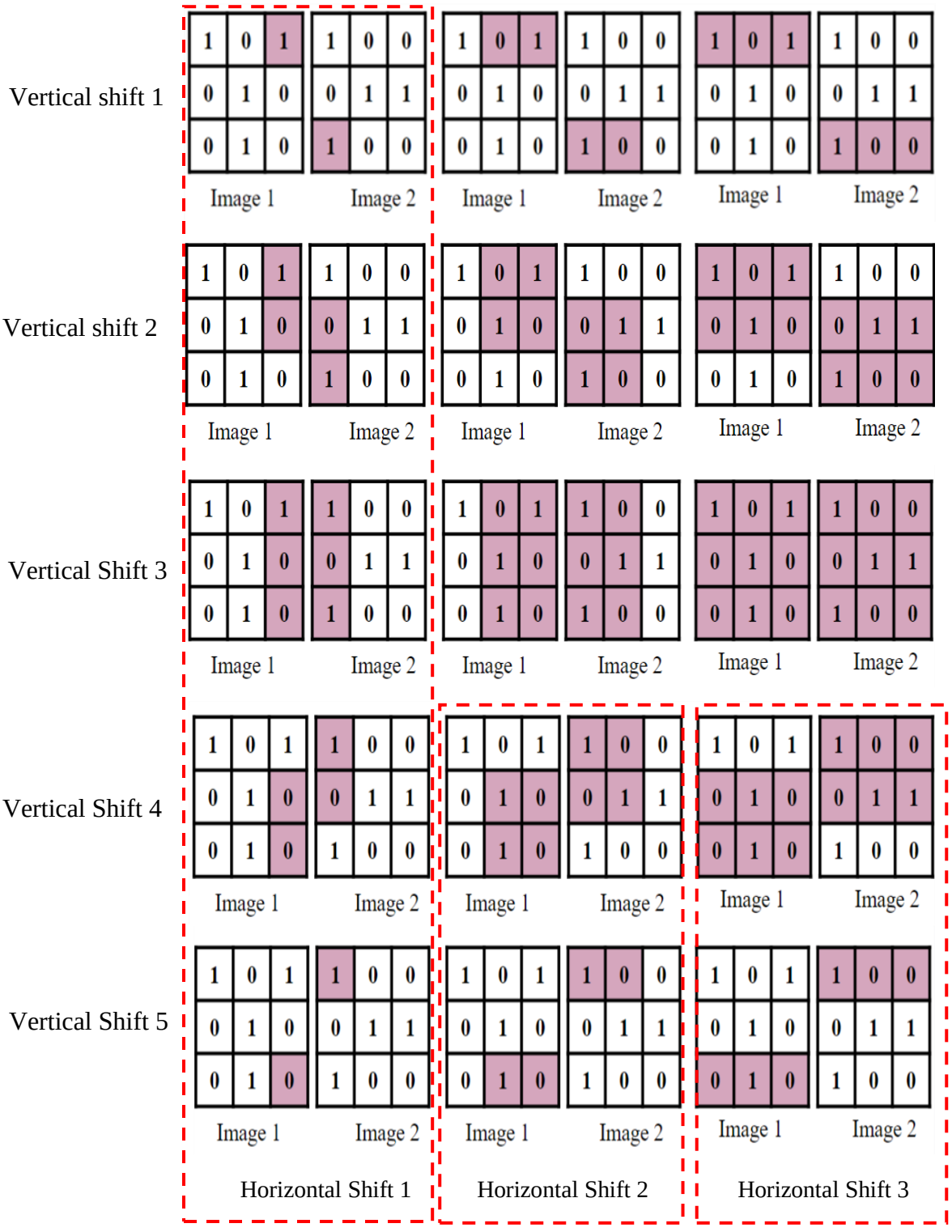


Figure 3.11 Two- dimensional image stitching example

As shown in Figure 3.9 an image stitching is performed for the one-dimensional image stitching. However, in the case of the two-dimensional image stitching, since the overlapped area is in two directions, as shown in Figure 3.10, it requires extra steps to stitch the two images.

Figure 3.12 explains the stitching procedure after finding the overlapped area.

Step 1: Replace the pixels in the overlapped area of image-2 with 0's.

Step 2: Add a row of 0's below the overlapped area of image 2 and above the overlapped area of image 1. The rows with 0's are used to be filled the blank area that is generated after combing the two images. Here the highlighted columns of image-1 and image-2 with green color are used for stitching two images.

Step 3: Circular shift the image 2 such that the overlapped columns in both the images are in parallel. And then add two images;

The result of step 3 contains the overlapped area. The highlighted green colored columns are considered as the image data for stitching two images. The final stitching image is obtained by concatenating the highlighted green color of image-1from step 2, next from the resulting image data from step 3 and then from image-2 of step 2. Figure 3.13 shows the two-dimensional image stitching result.

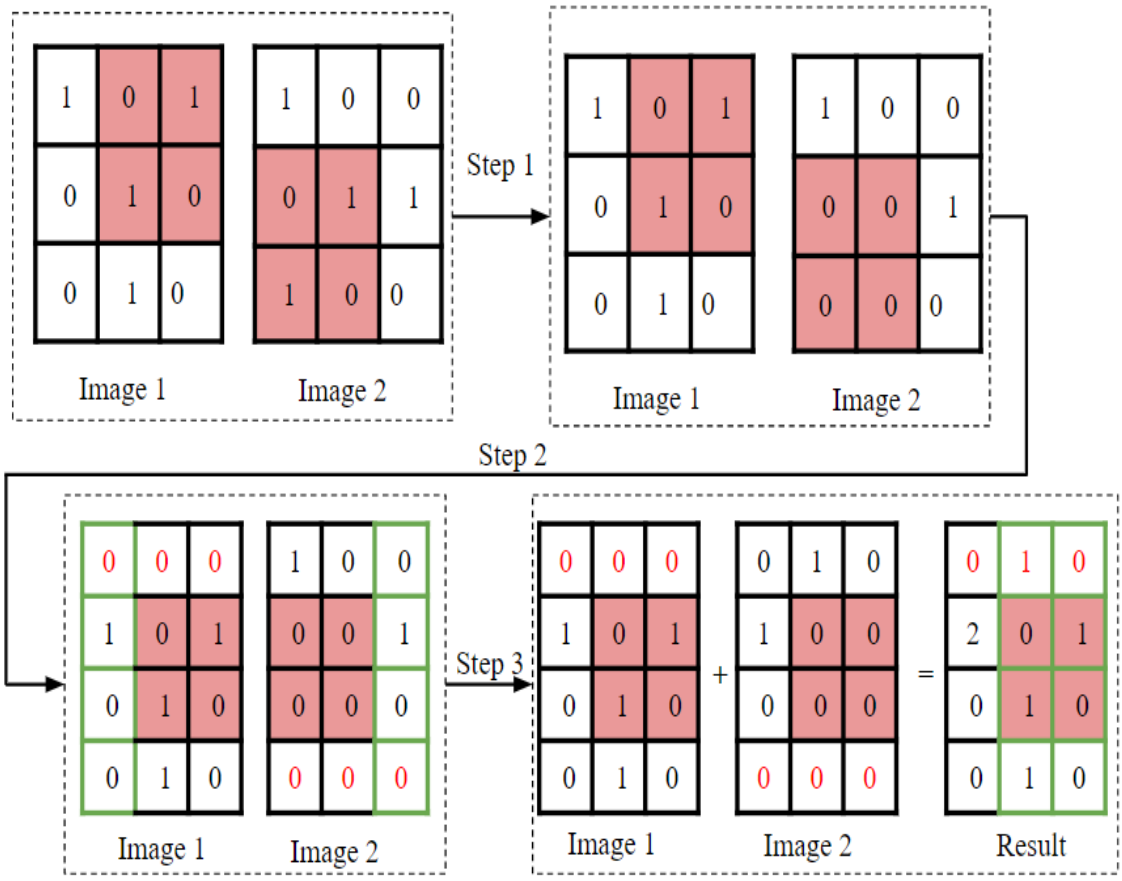


Figure 3.12 Two-dimensional image stitching procedure

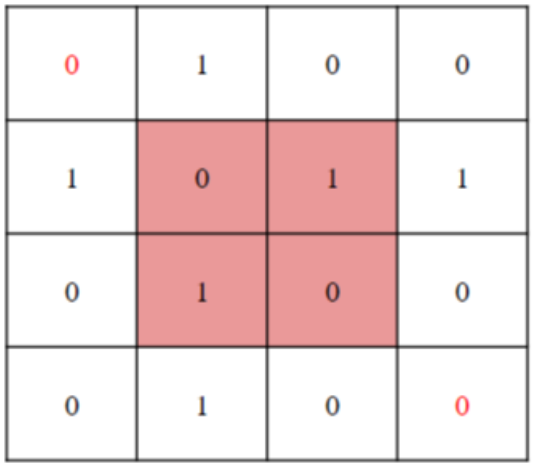


Figure 3.13 Final stitched image

CHAPTER IV

VIRTUAL PROTOTYPE OF AREA-BASED IMAGE STITCHING ALGORITHM

This chapter describes the virtual prototype of the area-based image stitching algorithm. A virtual hardware is implemented for the proposed area-based image stitching algorithm. Each component in the image stitching algorithm is converted into a corresponding hardware block. This approach allows estimating the structure and computation cost of the actual hardware.

4.1. Concept of virtual prototyping

The basic concept of the proposed virtual prototyping is converting the image stitching concept or algorithm into an actual hardware that consists of the memory, logic, or arithmetic unit. Also, this approach allows estimating the structure and computation cost of the actual hardware.

Figure 4.1 shows the concept of the proposed virtual prototyping, which is applied to an area-based image stitching algorithm, where Figure 4.1 (a) shows the conceptual representation of the algorithm. Image 1 and image 2 are taken to perform the comparison. Images are shifted in such a way that each pixel is compared with others. In the process of finding the overlap, the higher number of pixels or the maximum area with a minimum average of least square differences is considered as an optimum overlapped area. After selecting the best overlap in the images, the stitching process is performed.

Figure 4.1 (b) shows the resulting virtual hardware obtained through virtual prototyping. The virtual hardware is equivalent to the area-based image stitching algorithm which performs the same operation. However, the conceptual blocks are replaced with actual circuit components such as the memory, logic gates, and arithmetic unit. As a result, from the virtual hardware, the size and computation time of the actual hardware can be estimated. The virtual prototyping approach facilitates the hardware design of image processing circuits by enabling performance estimation at the early design stage.

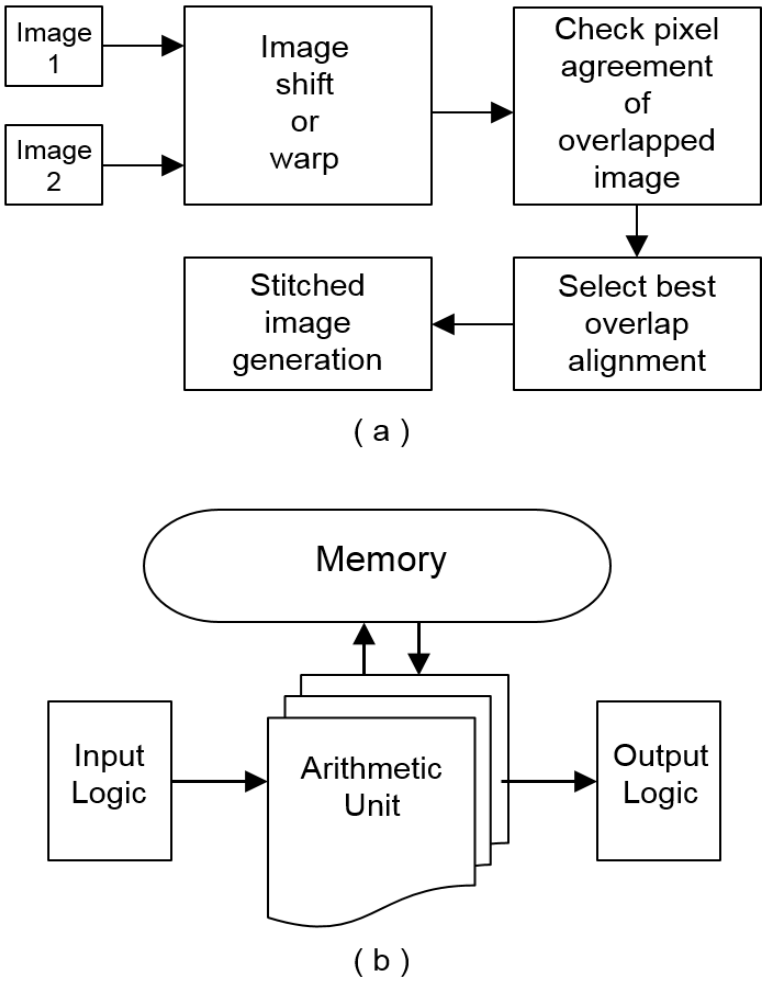


Figure 4.1 Concept of virtual prototyping (a) Area-based mage stitching algorithm (b) Virtual hardware.

4.2. Virtual hardware implementation

A virtual hardware platform is designed for the proposed area-based image stitching algorithm shown in Figure 3.1. Figure 4.2 shows the virtual hardware where there is a one-to-one correspondence between the area-based image stitching algorithm in Figure 3.1 and Figure 4.2. This figure is explained from top to bottom and is related to each block in Figure 3.1. Subsections in section 4.2 refers both Figure 4.2 and Figure 3.1 for better understanding.

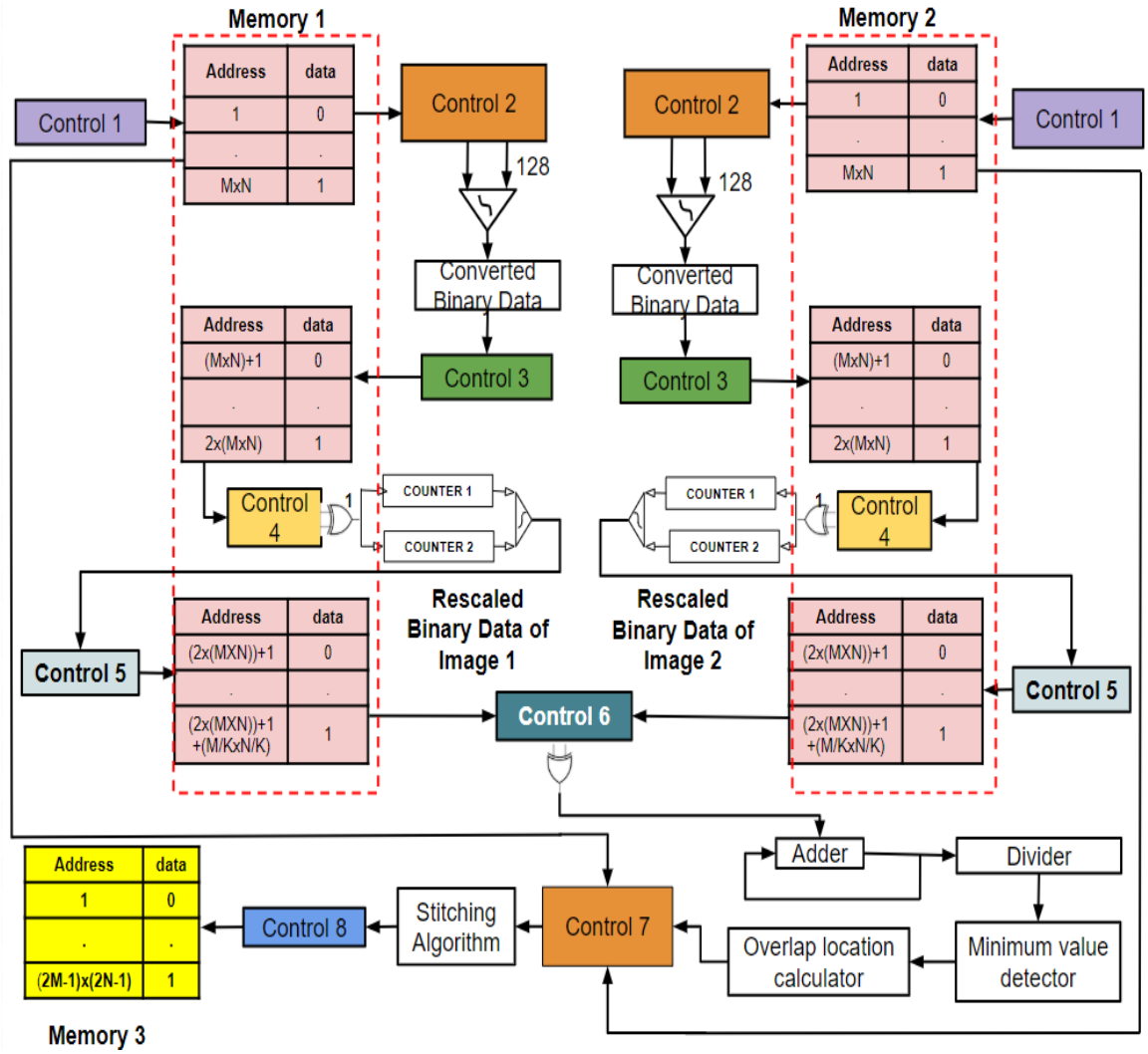


Figure 4.2 Virtual hardware of area-based image stitching platform

Memory-1 is used to store the image-1 data throughout the process in the algorithm. Similarly, memory-2 is used to store image-2 data. The final stitched image data is stored in memory-3. Memory-1 and memory-2 are indicated in pink color. Also, memory-3 is indicated in yellow color at the bottom left corner. Writing and reading for memory-1 and memory2 are performed simultaneously. 8 different controls (control-1 to control-8) are used to read and write the image data.

Control-1 block is used to write the grayscale image data (0 to 255) into memory-1 and memory-2. Since the size of the image is M-by-N, it takes **M×N** cycles to write the grayscale data into the memories. It takes 1 clock cycle for one-pixel data to be written in the memory. The grayscale image converted from RGB is used in the virtual hardware.

4.2.1. Converting grayscale image into binary image

Control-2 block is used to read the grayscale image data from memory-1 and memory-2 to convert the grayscale image data into binary data (1 or 0), for which it takes **M×N** cycles. It takes each pixel and compare with the threshold to generate a binary output.

A comparator is used to compare each grayscale pixel in the image-1 and image-2 with 128. Figure 4.3 shows the virtual hardware of converting a grayscale image into a binary image. If the value of the grayscale image data is less than 128, the output of the comparator is 0, if the value of the grayscale image data is higher than 128, the output of the comparator is 1. A combinational logic is used to compare the grayscale data with the threshold 128.

Control-3 block which is represented as the green color, is used to write the converted binary data into the memory-1 and memory-2 in Figure 4.2. Writing the binary

data into the memories is realized in a pipelining process, that is, one grayscale pixel data is read by control-2 in 1st clock cycle, then converted and then written into the respective memory in 2nd clock cycle. Now 2nd grayscale pixel data is read by control-2 in the 2nd clock cycle and then converted and then 2nd binary pixel data is written into respective memory in 3rd clock cycle and so forth. It takes only 3 clock cycles to read convert and write the first two-pixel data into the memories. Therefore, it requires 2((M×N)+1) cycles to complete the binary image conversion.

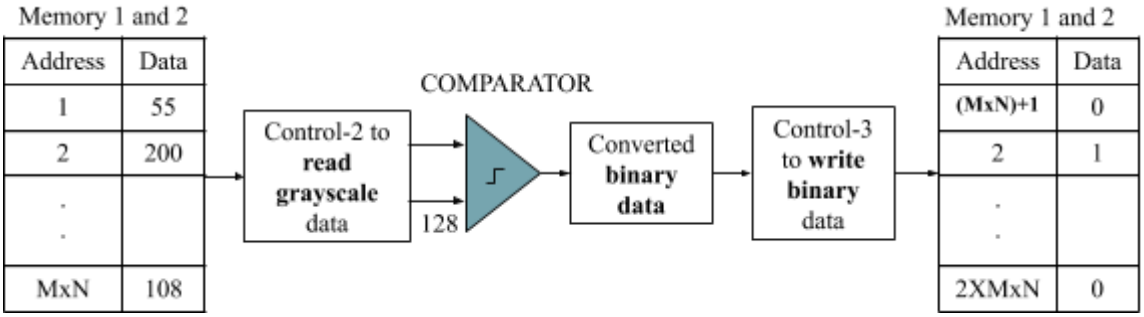


Figure 4.3 Virtual hardware for grayscale to binary conversion

4.2.2. Image size rescaling

Figure 4.4 shows the virtual hardware to scale the binary image with scaling factor K (2, 4, 8, 16). To rescale the image, an XOR gate, two counters, and a comparator are used. Since K=8 is being fixed by the proposed algorithm it needs to count a maximum of 8 (1000), therefore MOD-4 counters are used in this virtual hardware, thus 4 flipflops are needed. The counter is reset after processing K×K data. The rescaling mentioned in virtual hardware is obtained from rescaling algorithm mentioned in the Figure 3.1. From the memory of size M×N, the size K×K (2×2 or 4×4 or 8×8) binary pixel data is read, and each bit is compared with 1 using the XOR gate. If the XOR gate output is 1 counter 1 is

incremented by 1 and if the XOR gate output is 0 counter 2 is incremented by 1. Following, the total number of 1's, and 0's are compared using a comparator. The K×K pixel data is replaced with the highest occurrence of data (1 or 0) in the memory 1 and 2. In this manner, the M×N image is mapped to M/K×N/K image.

Here M×N clock cycles are required to read the image data for processing K-by-K data. One clock cycle to generate the rescaled data and another clock cycle to write the rescaled data into memories. For example, if K = 8 it takes 64 cycles to read the binary data from memory 1 and 2 using control 4. It takes 1 cycle to rescale the data (65 cycles for now), and one cycle to write the rescaled data into the memories 1 and 2 using control 5. Next, consider reading another K×K which corresponds to, 64 pixels from 66th cycles. So, it took 66 cycles to write 1 rescaled data into the memory. And next 64 bits are read from memory 1 and 2 using control 4 so it took 130 cycles and cycles for writing. So, it consumes 131 clock cycles. Therefore, it is expected to take (M×N)+(M/K×M/K) +1 cycles to complete the rescaling operation.

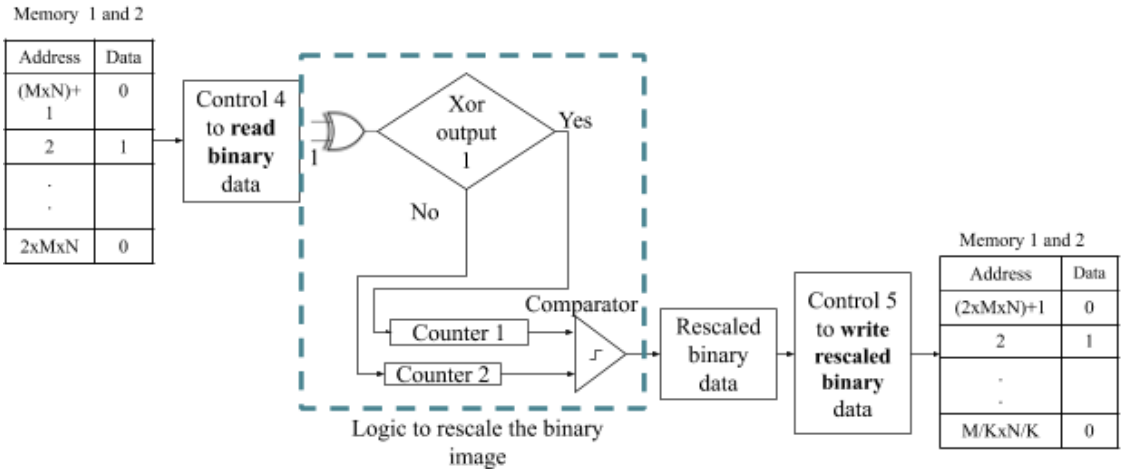


Figure 4.4 Virtual hardware to rescale the binary image

4.2.3. Two-dimensional pixel comparison

After rescaling the binary images, the two rescaled images are compared to find the optimum overlap between the two images. Each pixel in the image 1 is compared with each pixel in the image 2. The virtual hardware performs a 2-dimensional pixel comparison based on the proposed algorithm.

As shown in Figure 4.2 control-6 is used to read the required image data from memory 1 and memory 2. The image data is compared as explained in chapter 3 section 3.3. The image data from memory 1 and memory 2 is read in a format which corresponds to, the shaded part read from the memories as shown in Figure 3.11. Comparing the pixels in hardware is realized by a combinational logic with XOR gate, adder, and divider. Area or pixel data read for each iteration from memory 1 and memory 2 are compared with the XOR gate, and the results are summed using an adder. Next, a divider is used to divide the final sum (pixel comparison result) by the number of comparisons performed. Figure 4.5 shows an example of reading the rescaled image data from the corresponding address of memory 1 and memory 2 to perform the comparison between two images. In this iteration, only the shaded pixels in image 1 and image 2 are read and used to compare the difference.

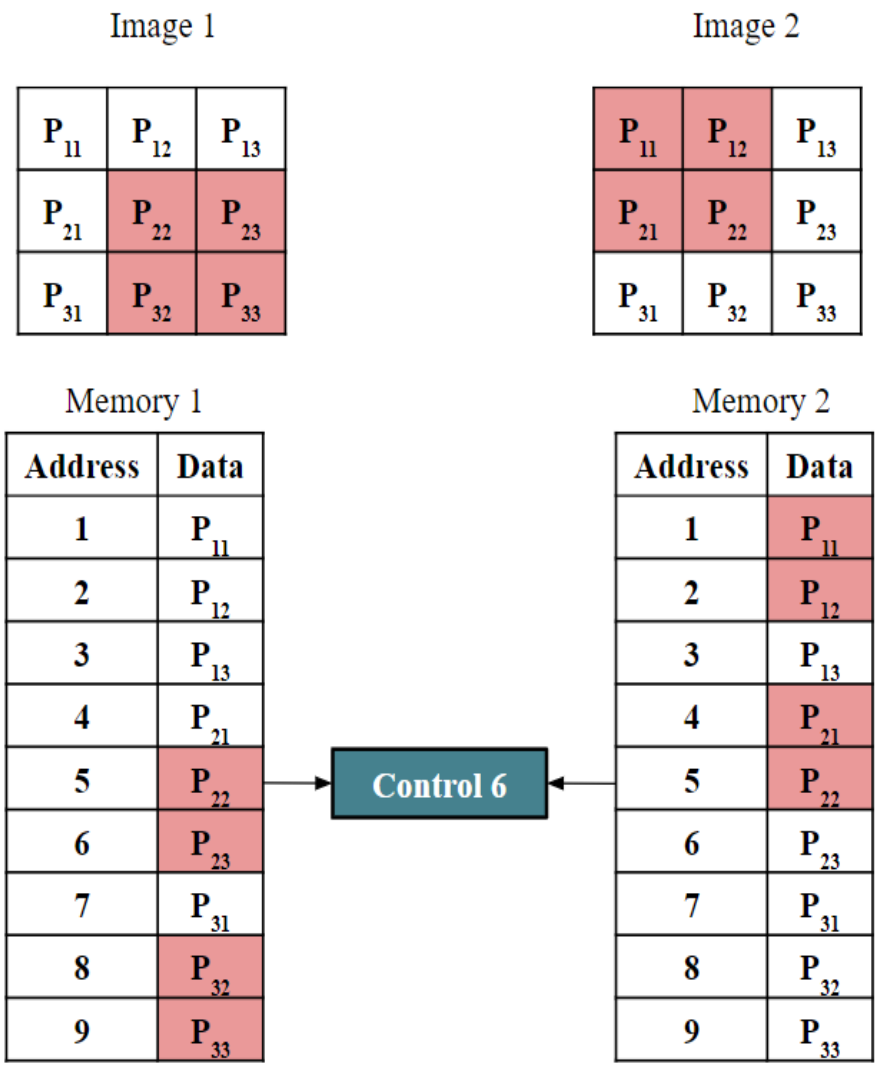


Figure 4.5 Example of reading rescaled data from the memories for pixel comparison

4.2.4. Optimum overlap detector

Figure 4.6 shows the logic for the optimum overlap detector. After reading the required image pixel data, each pixel is compared using an XOR gate. If four pixels are compared four XOR gate outputs are added using an ADDER. Further, the sum is divided by the number of pixels being compared using a DIVIDER. Next a comparator is used to find the optimum overlap. One input of the comparator is connected to the divider output,

where another input is initially set to 2. The first comparison is only 1 pixel (either 1 or 0), as explained in chapter 3 section 3.3. it is observed from Figure 3.11, in vertical shift 1 and horizontal shift 1. So, the first compared average is either 1 or 0. Next, comparator output is given to the optimum overlap detector.

The divider output is compared with the other input of the comparator. One input of the comparator is divider output and another input is initially 2. Since the divider output of the first cycle is 0 or 1 it is compared with number 2 and the second input is updated with the divider output value. If the divider output is smaller than the second input the present divider output is given as feedback to the second input of the comparator. If the divider output is greater than the second input (previous divider output), no operation is performed. If the divider output is equal to the second input (previous divider value), the number of comparisons is stored in temporary register. Furthermore, the maximum number of comparisons is used to decide the position of the optimum overlapped area.

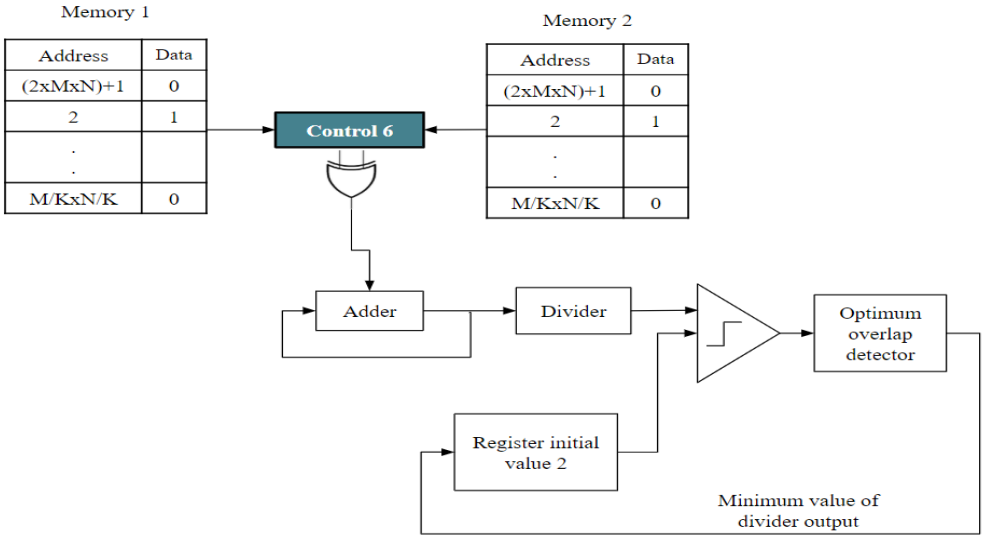


Figure 4.6 Optimum overlap detector

After finding the overlapped area in two images it's time to stitching two images. Figure 4.7 shows the outline of the stitching procedure. As shown in Figure 4.2, the control-7 block is used to read the grayscale image data to perform the stitching operation. Since the binary image is not clear for the naked eye, grayscale image data is used instead of binary data to stitch two images. The grayscale image data is stored in memory 1 and memory 2 from address 1 to M×N. Optimum overlap detector is given as input to the stitching algorithm to get the optimum overlap. After stitching control 8 block is used, which represents the control to write the stitched data into memory 3.

The stitched image size is increased depending on the size of the overlap between the two images. If the overlapped area is small the stitched image is big. If the overlapped area is big the stitched image is small. The size of memory 3 is considered based on the smallest overlapped area between the two images. If the overlapped area is only 1 pixel the number of pixels of the stitched image is (2M-1) × (2N-1). Therefore, it takes (2M-1) × (2N-1) cycles to write the stitched M×N image data into the memory 3.

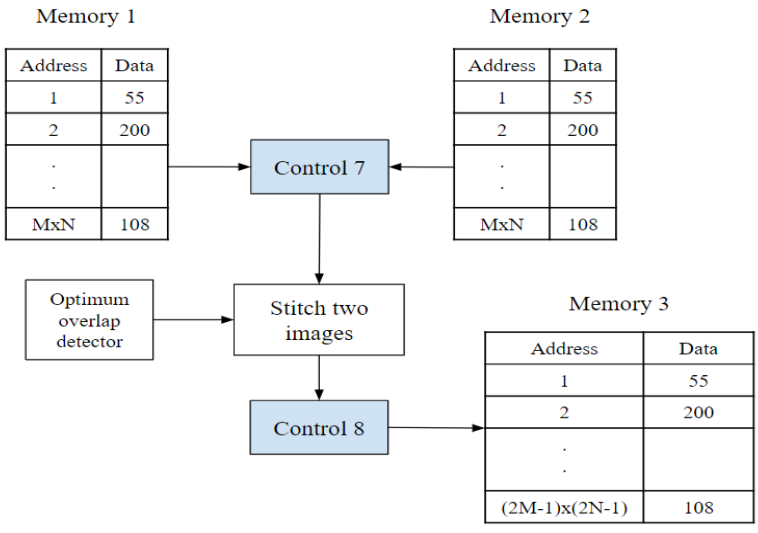


Figure 4.7 Stitching two images using grayscale pixel data.

CHAPTER V

IMAGE STITCHING RESULTS AND COMPUTATIONAL COST

5.1. Two-dimensional image stitching results

Figure 5.1 shows the result of two-dimensional area-based image stitching based on the proposed algorithm obtained using MATLAB. The two grayscale input images of a laboratory are shown in Figure 5.1(a), and the stitched images for scaling factor K=1, K=8 and K=16 are shown in Figure 5.1 (b), (c) and (d). Using the scaling factor K=1 is like applying the image stitching algorithm without image rescaling. It is observed that the stitching operation is implemented precisely, and there is no disruption near the overlapped area in the stitched image for K=1. However, the MATLAB computation time for K=1 took 1489.34 seconds. To reduce the computation time, the rescaling factor K is increased to 8. It is observed that there is no difference between Figure 5.1 (b) and 5.1(c), the overlapped area is the same for both the images. Similarly, Figure 5.2 the college campus photo is used as input images. The stitched images for scaling factor K=1, K=8 and K=16 are shown in Figure 5.2 (b), (c) and (d). The output image is the same for K=1 and K=8, shown in Figure 5.2 (b) and (c), whereas, Figure 5.2 (d) shows the distorted output for K=16.

It took 8.2 seconds in MATLAB to execute the image stitching algorithm. Further scaling factor K is increased to 16; it is observed in Figure 5.1 (d) that there is some

distortion, the overlapped area is not accurate. The algorithm took 3.3 seconds in MATLAB. The final stitched images are incorrect for K=16 because 16×16 data is replaced with 1 or 0 during rescaling. Therefore, more image data is lost. The purpose of this algorithm is to perform fast area-based image stitching. Therefore, the scaling factor K=8 is considered, which is not losing the image information and size of the image is reduced, and therefore, the computation cost is also reduced when compared to K=4 or 2. After stitching the leftover area is filled with 0 values to make it a complete image (left bottom and right top corners).

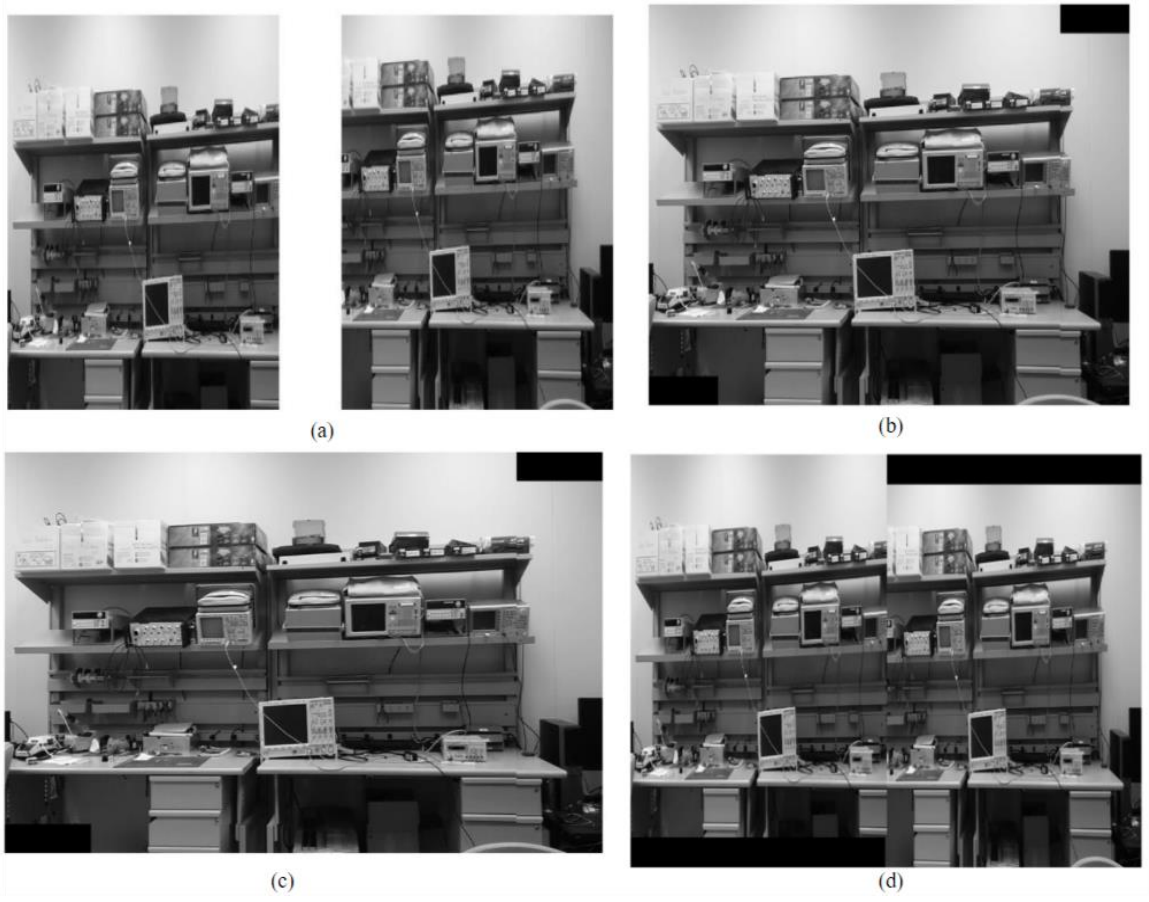


Figure 5.1 Image stitching result of lab photo (a) Input images of size 1280×1024 (b) Output image for K=1 (c) Output image for K=8 (d) Output image for K=16

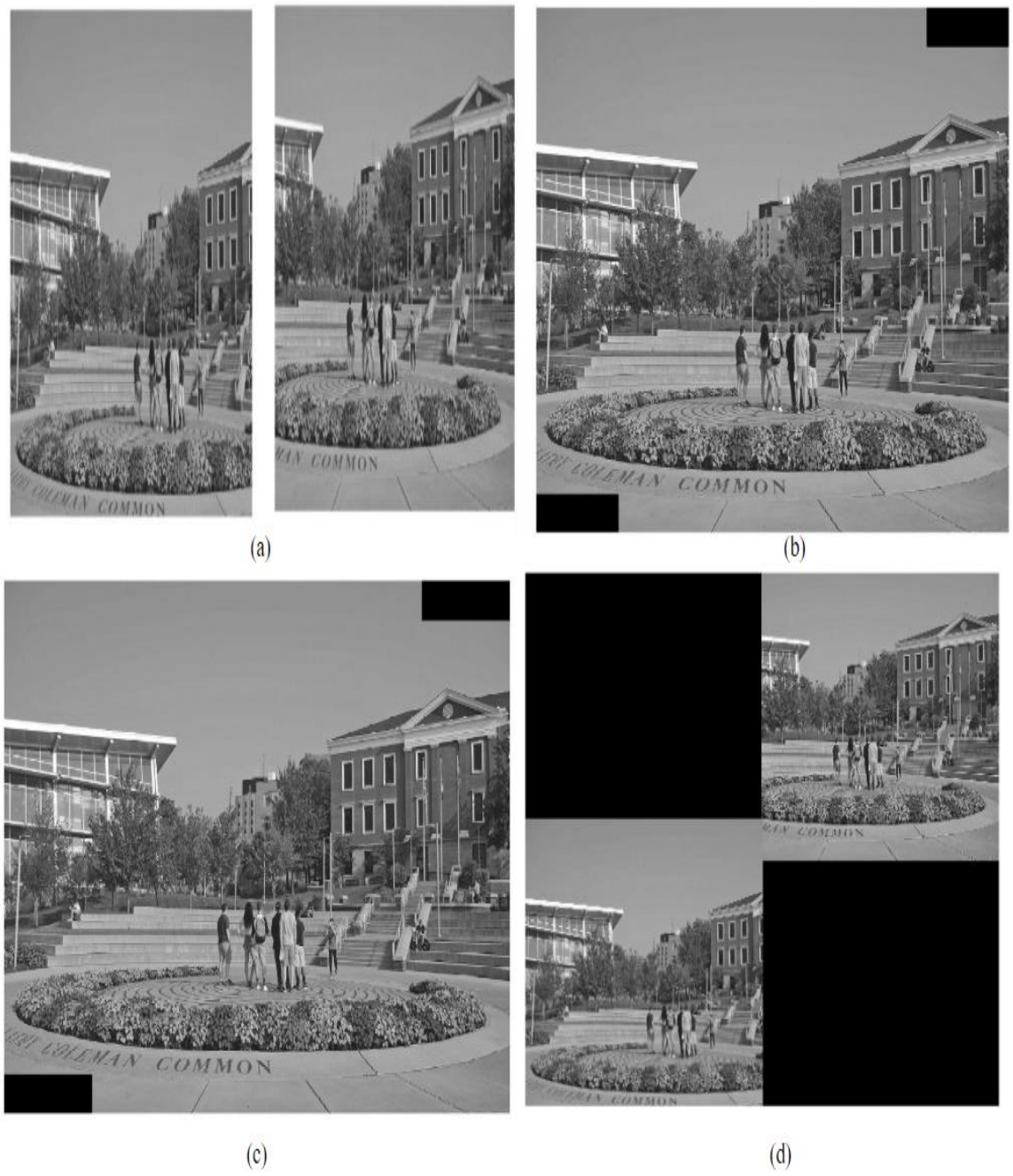


Fig 5.2 Image stitching result of college campus photo (a) input images of size 1280×1024 (b) output image for K=1 (c) output image for K=8 (d) output image for K=16

5.2. Computation time

The computation time of virtual hardware can be calculated from the number of cycles taken to perform the image stitching operation that is shown in Figure 5.1 and Figure 5.2. However, to end up with a general expression, a M×N image is used to find the computation time.

Accessing memory-1 and memory-2 can be performed simultaneously. Thus it takes M×N cycles to write the grayscale image data into the memories, to read the grayscale image data from the memories to convert into binary data, reading the grayscale data (M×N), converting grayscale data to binary data and writing the binary data (+1) into the memories can be realized using pipelining. It takes M×N+1 cycles to read and write the converted binary data into the memories. Reading K×K binary data bits to rescale the binary image also takes M×N cycles. Again, write the rescaled bit into the memory using pipelining. Writing the rescaled data also takes M×N+1 cycle.

The number of cycles to read the binary data from the two images to perform the comparison (XOR operation, addition, and division) is given by

$$\frac{\frac{M^2}{K}\times\frac{N}{K}\times\left(\frac{N}{K}+1\right)}{2} \tag{5.1}$$

After comparison and finding the optimum overlap, to stitch the images, read the grayscale image, which takes M×N cycles. To write the final stitched image data it requires (2M-1)×(2N-1) cycles. This equation gives the worst-case scenario of the overlapped area (when the overlapped area is only one pixel). By adding the number of

cycles for each operation, the total number of cycles required to perform image stitching is given

$$4 \times (M \times N) + 2 + \frac{\frac{M^2}{K}+\frac{N}{K}+\left(\frac{N}{K}+1\right)}{2} + ((2M-1) \times (2N-1)) \quad (5.2)$$

Figure 5.3 shows the number of cycles required for each operation for stitching 1280×1024 (M=1280, N=1024) sized image for K=8.

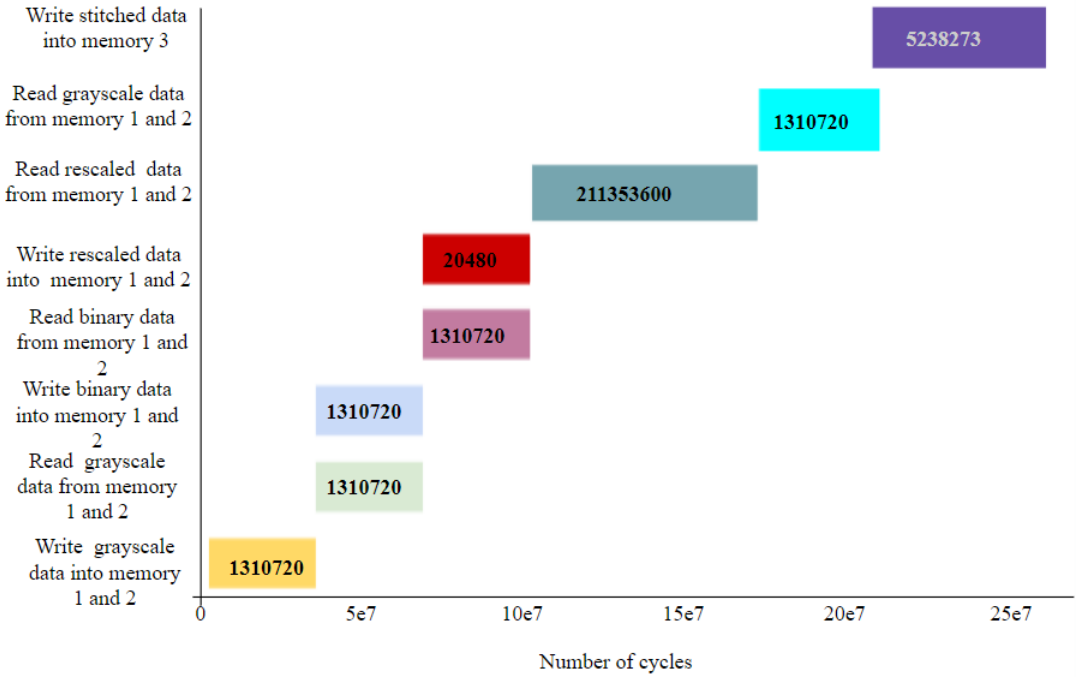


Figure 5.3 The number of cycles required for each operation stitch two 1280×1024 Images with K=8.

The actual size and computation time of the area-based image stitching hardware can be estimated from the virtual hardware. Table 1 shows the estimated computation time of the area-based image stitching hardware compared with the software algorithm realized using MATLAB. The computation time of the virtual hardware is obtained by using equation 2 with different clock frequencies (250MHz, 500MHz, 750MHz, and 1GHz) and

image scaling factor K (2, 4, 8, and 16). The computation time decreases as the clock frequencies and scaling factor K increases. However, increasing K has the risk of losing the useful information contained in the image; this can lead to poor stitching results.

Based on the simulation results, K=8 still gives good stitching quality with relatively fast computation time. With K=8 and clock frequency of 250 MHz, the computation time of the virtual hardware shows 0.887 Sec which is one-tenth the time required by MATLAB

Table 5. 1 The computation time of MATLAB and virtual hardware platform for stitching two 1280×1024 images.

K value	MATLAB (sec)	Vitrual hardware (Sec)			
		Clock Frequency			
		250 MHz	500 MHz	750 MHz	1GHz
2	599.06	215.2	107.6	71.736	53.802
4	51.77	13.516	6.758	4.503	3.379
8	8.2	**0.887**	0.443	0.295	0.221
16	3.351	0.0951	0.047	0.0317	0.0239

5.3. Hardware Size

Three memory blocks are used in the virtual hardware, where memory-1 is used for image 1 data and memory-2 is used for image 2 data storage. As discussed in section 4.2 and shown in Figure 4.2, the maximum size of the two memories is (2× (M×N)) + (M/K× N/K). For having considered M=1280 and N=1024, the size of memory-1 and memory 2 is nearly 12 megabits. Memory-3 is used to store the stitched image data. It is estimated that the maximum size is up to 5.3 MB for the worst-case overlapped images (overlap in only one pixel) stitched image.

Control 1 to control 8 in Figure 4.2 in chapter 4 are used to write and read the image data to perform the required operation. For example, control-1 is used to write the grayscale image data into the memories. To write the data into the memory the counter is used to count the address and write the corresponding image data into the memory. Therefore, it takes 1280×1042 = 131720 (21bits) cycles to write the image data. It takes 21 flipflops to count up to 131720. Assuming each flip-flop requires 4 gates it is concluded that control-1 takes 84 gates to perform the write operation.

Similarly, the number of gates for control-2, control-3, control-4 takes 84 gates. Control-5 is used to write the rescaled image data with size (M/K × N/K). Thus, it takes 60 gates. Control-6 is used to read the image data to perform the comparison operation $\left(\frac{\frac{M^2}{K}+\frac{N}{K}+\left(\frac{N}{K}+1\right)}{2}\right)$ cycles it requires 124 gates. Control-7 is used to read the grayscale data, therefore it takes 84 gates and finally, control-8 is used to write the stitched image data with the size of (2M-1) × (2N-1), it takes 92 gates. The number of gates used in the virtual

hardware is shown in table 5.2. Thus, the total gate count of the virtual hardware is estimated to be 874.

Table 5. 2 Gate count of virtual hardware

Block	**Number of Gate/ Memory size**
Control 1	84
Control 2	84
Control 3	84
Control 4	84
Control 5	60
Control 6	124
Control 7	84
Control 8	92
Comparators	90
Counters	64
XOR gates	24
Memory 1	12 Mbits
Memory 2	12 Mbits
Memory 3	5.3 MB

CHAPTER VI

CONCLUSION AND FUTURE WORK

6.1. Conclusion

A virtual prototyping design approach for a fast area-based image stitching algorithm is proposed. The virtual hardware obtained from virtual prototyping is equivalent to the conceptual algorithm, thus actual circuit components including the memory, logic gates, and arithmetic units are linked to the conceptual blocks in the proposed algorithm. Using the proposed method, the overall structure, hardware size, and computation time of the actual hardware are estimated. In addition, the image stitching algorithm using binary image comparisons and image scaling eliminates the need for the huge multipliers and adders in the virtual hardware. Thus, the proposed algorithm leads to reduced hardware size and faster computation time. Results show, assuming the clock frequency of 250 MHz, the estimated computation time for the proposed virtual hardware to stitch two 1280×1024 images was 0.877 sec, which is one-tenth the time required by the software-based image stitching algorithm realized in MATLAB. The estimated size of the virtual hardware was memory size of 66.4 Mbits and gate count of 874 gates.

6.2. Future work

This design has demonstrated that the area-based image stitching can decrease the time taken to stitch two images and the computation cost is reduced in virtual hardware. There are a few verifications and optimization that can be done in the future:

- The virtual hardware can be further implemented on actual hardware. The TRDB_D5M Kit provides everything that needs to develop a camera on the Altera DE2_115 board.
- Research can be conducted on virtual hardware to estimate the power consumption.

REFERENCES

[1] J. H. Chen and C. M. Huang, "Image stitching on the unmanned air vehicle in the indoor environment," in *Proc.SICE Annual Conf.,* Aug. 2012, pp. 402-406.

[2] S. Chen, "QuickTime VR – An image based approach to virtual environmental navigation," *SIGGRAPH 95,* vol. 29, pp. 29-38, 1995.

[3] R. Szeliski and H. Shum, "Creating full view panoramic image mosaics and environmental maps," *SIGGRAPH 97,* vol. 31, pp. 251-258, 1999.

[4] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," *Int. J. Computer Vision,* vol. 74, pp. 59-73, 2007.

[5] M. Qasaimeh and A. Sagahyroon, "FPGA-Based Parallel Hardware Architecture for Real-Time Image Classification," *IEEE Transactions on Computational Imaging*, vol. 1, no. 1, pp, 56-70, March 2015.

[6] J. H. Wang and S. Zhong, "An Embedded System-onChip Architecture for Real-time Visual Detection and Matching," *IEEE Transaction on circuits and systems for video technology*, vol.24, no.3, pp.525-538, march 2014.

[7] S. Zhong and J. H. Wang, "A real-time embedded architecture for SIFT," *Journal of Systems Architecture,* vol.59, pp. 16–29, 2013.

[8] V. Bonato and E. Marques, "A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection," *IEEE Transactions on Circuits And systems For Video Technology*, vol. 18, no. 12, December 2008.

[9] T. V. Huynh, “Deep neural network accelerator based on FPGA,” in *Proc. IEEE Information and Computer Science,* Nov. 2017, pp. 254-257.

[10] S. Pravenaa and R. Menaka, “A methodical review on image stitching and video stitching techniques,” *Int. J. Appl. Eng. Res.*, vol. 11, no. 5, pp. 3442–3448, 2016.

[11] R. Szeliski, “Image Alignment and Stitching: A Tutorial,” *Found. Trends® Comput. Graph. Vis.*, vol. 2, no. 1, pp. 1–104, 2006.

[12] D. Bhadane and K. N. Pawar, “A Review Paper on Various Approaches for Image Mosaicing,” *Int. J. Eng. Manag. Res. Page Number*, vol. 62, pp. 193–195, 2013.

[13] S. Arya, “A review on image stitching and its different methods,” *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 5, no. 5, pp. 299–303, 2015.

[14] B. Zitová and J. Flusser, “Image registration methods: A survey,” *Image Vis. Comput.*, vol. 21, no. 11, pp. 977–1000, 2003.

[15] P. F. Felzenszwalb and R. Zabih, “Dynamic programming and graph algorithms in computer vision,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 4, p. 721—740, Apr. 2011.

[16] M. Brown and D. G. Lowe, “Automatic panoramic image stitching using invariant features,” *Int. J. Comput. Vis.*, vol. 74, no. 1, pp. 59–73, 2007.

[17] M. Wang, S. Niu, and X. Yang, “A novel panoramic image stitching algorithm based on ORB,” *Proc. 2017 IEEE Int. Conf. Appl. Syst. Innov. Appl. Syst. Innov. Mod. Technol. ICASI 2017*, pp. 818–821, 2017.

[18] K. Chen and M. Wang, “Image stitching algorithm research based on OpenCV,”

Proc. 33rd Chinese Control Conf. CCC 2014, no. 1, pp. 7292–7297, 2014.

[19] J. Li and J. Du, "Study on panoramic image stitching algorithm," *2010 2nd Pacific-Asia Conf. Circuits, Commun. Syst. PACCS 2010*, vol. 1, no. 408202, pp. 417–420, 2010.

[20] K. S. Meshram and A. M. Agarkar, "Content Based Image Retrieval Systems using SIFT : A Survey," *Int. J. Electron. Commun. Eng.*, vol. 2, no. 10, pp. 18–25, 2015.

[21] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," in *International Journal of Computer Vision*, 2004, pp. 91–110.

[22] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3951 LNCS, pp. 404–417, 2006.

[23] B. B. Swapnali and K. S. Vijay, "Feature Extraction Using Surf Algorithm for Object Recognition," *Int. J. Tech. Res. Appl.*, vol. 2, no. 4, pp. 2320–8163, 2014.

[24] A. Xu and G. Namit, "SURF: Speeded-Up Robust Features COMP 558-Project Report," McGill University, pp. 2–29, 2008.

[25] A. H. Nguyen, M. Pickering, and A. Lambert, "The FPGA implementation of an image registration algorithm using binary images," *2014 Int. Conf. Reconfigurable Comput. FPGAs, ReConFig 2014*, no. May 2015, 2014.

[26] M. N. Haque, M. Biswas, and M. R. Pickering, "Computationally efficient global motion estimation using a multi-pass image interpolation algorithm," *2012 Pict.*

Coding Symp. PCS 2012, Proc., pp. 349–352, 2012.

[27] A. Mordvintsev and K. Abid, “Introduction to SURF (Speeded-Up Robust Features) — OpenCV-Python Tutorials 1 documentation,” 2013. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html. [Accessed: 03-Dec-2019].

[28] U. Sinha, “SIFT: Theory and Practice: Generating a feature - AI Shack,” 2017. [Online]. Available: http://aishack.in/tutorials/sift-scale-invariant-feature-transform-features/. [Accessed: 03-Dec-2019].