Aligning Time-Varying Fields of View Using Feature Detection on FPGA

Robert Taglang
Drexel University
Philadelphia, Pennsylvania
rob@taglang.io

Prawat Nagvajara Ph.D
Drexel University
Philadelphia, Pennsylvania
nagvajara@coe.drexel.edu

ABSTRACT

This paper presents a system for detecting features using an FPGA implementation of SURF (Speeded-Up Robust Features), and aligning video streams by applying an adaptive transform generated based on key features. As a result the proposed technique can align time-varying field of view sources. The results include an efficient FPGA implementation of SURF and a real-time affine transform for aligning video sources to achieve minimal latency. The design is based on pipelining the SURF computation and using address mapping for fast transformations. The implementation was performed on a ZedBoard Zynq-7000 ARM/FPGA SoC Development board with two 640 by 480 resolution video streams at 30 frames per second. This approach has potential for future applications including stabilization by matching features between consecutive video frames.

ACM Reference Format:
Robert Taglang and Prawat Nagvajara Ph.D. 2018. Aligning Time-Varying Fields of View Using Feature Detection on FPGA. In *Proceedings of 26th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (ISFPGA'18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The fusion of data from two or more sensors has been well-researched [12] [10], though these approaches typically discuss the process of fusing images which have been pre-aligned. Pre-computed transforms used to align the frames of two cameras are not robust to variations. Some approaches have made use of additional hardware sensors in order to correct against these variations [5]. The approach presented in this thesis seeks to perform this correction in real time, completely in hardware, using feature detection on a Field Programmable Gate Array (FPGA) using the design shown in Figure 1.

In order to align two images, one is treated as the reference image, and the other is the transformed image which needs to be aligned with the reference image. As such the problem can be broken into four major components.

(1) Detecting feature points in the two images (Hessian Determinant, Non-maximum Suppression, Feature Buffer block).

ISFPGA'18, February 2018, Monterey, California, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

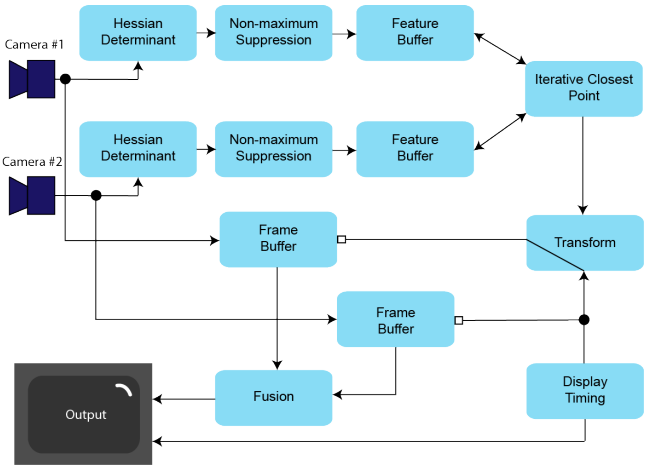


Figure 1: Block diagram of the proposed system for alignment and fusion

(2) Computing the transform which maps the transformed image into the space of the reference image (Iterative Closest Point block).
(3) Applying the computed transform to the transformed image (Transform, Frame Buffer, Display Timing block).
(4) Fusion of the reference and transformed image (Fusion block).

Feature points are detected using some of the techniques from Speeded-up Robust Features (SURF) which makes use of Hessian determinants to detect points of interest in multiple scale spaces. High magnitude features are stored in a buffer to be used for computing the transform.

Once the feature points for the two images have been computed and stored in the buffer, the points from the transform image are mapped to their closest point in terms of euclidean distance from the reference set to create an orthogonal projection from the transform points to the reference points. Singular value decomposition is used to compute the pseudo-inverse of the matrix containing the reference points so it can be premultiplied by the matrix containing the transform points. The result of this product is the transform matrix which maps transform points to reference points with the error minimized in a least-square sense. This process may need to be repeated multiple times to converge to a local error minimum, a process referred to as iterative closest point.

The image data streams into memory. With the transform computed, the desired address for the output image is decomposed into x and y coordinates, which are transformed and then reformed back into an address which is used to select pixels from the data in

memory. In this way, the transform is applied to align the transform image with the reference image.

Finally, the technique of Laplacian fusion is used to combine the aligned images. It effectively selects the highest frequency components from the two images in order to create an output image where the sharpest, focused parts of the two images are combined into one image.

This process is applied continuously in real time. As the data from the camera streams in, it produces a single fused output image for display with minimal latency. Implementing the device in hardware yields substantial benefits in terms of size and power consumption as an embedded solution over a more traditional software based approach to image processing.

2 BACKGROUND

This section presents technical details of the signal processing and algorithms used in the alignment and fusion. The SURF technique is first presented for computing Hessian determinants for use as a feature detector. Iterative closest point is used to compute a least-square fitting between the feature sets, and singular value decomposition is utilized for the computation of the transform. Finally, Laplacian fusion is used to combine the two fields of view.

2.1 Speeded-up Robust Features (SURF)

The generation of features for use as marker points in alignment utilizes the SURF algorithm from Bay et al [2]. SURF is composed of two parts: a discrete approximation for computing Hessian determinants, and the generation of rotation invariant feature descriptors for detected feature points.

SURF is typically used for its applications in object recognition, where the feature descriptor is used to facilitate a match between what is observed and some known set of feature points and descriptors. The descriptor largely serves as a way of discriminating against false positives. The detected feature points will be matched across two images with the assumption that the subject is the same and that the images do contain spatially coherent matches. Given this assumption, it can be concluded that the feature descriptor is not necessary for alignment. The feature points computed using Hessian determinants are sufficient.

2.1.1 Computation of Hessian Determinants. The Hessian determinant is the determinant of a matrix composed of the spatial partial second derivatives of a function $f : \mathbb{R}^n \to \mathbb{R}$. It is of the general form shown in Equation 1. In the image domain, $f : \mathbb{R}^2 \to \mathbb{R}$. The particular form of the Hessian matrix in $\mathbb{R}^2$ with dimensions x_1 and x_2 is shown in Equation 2.

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{1}$$

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} \tag{2}$$

The second order derivative used can be computed via convolution of the Gaussian second order derivative at any point, x, in the image. The formulas for the Gaussian second order derivative for each partial with respect to x_1^2, $x_1 x_2$ and x_2^2 can be seen in Equations 3, 4, and 5 respectively.

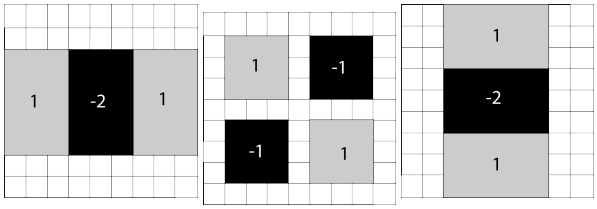
$$\frac{\partial^2 G(x_1, x_2, \sigma)}{\partial x_1^2} = (-1 + \frac{x_1^2}{\sigma^2}) \frac{e^{-\frac{x_1^2 + x_2^2}{2\sigma^2}}}{2\pi\sigma^4} \tag{3}$$

$$\frac{\partial^2 G(x_1, x_2, \sigma)}{\partial x_1 x_2} = \frac{x_1 x_2}{2\pi\sigma^6} e^{-\frac{x_1^2 + x_2^2}{2\sigma^2}} \tag{4}$$

$$\frac{\partial^2 G(x_1, x_2, \sigma)}{\partial x_2^2} = (-1 + \frac{x_2^2}{\sigma^2}) \frac{e^{-\frac{x_1^2 + x_2^2}{2\sigma^2}}}{2\pi\sigma^4} \tag{5}$$

The Gaussian second order derivatives can be approximated as 9×9 kernels with $\sigma = 1.2$. These kernels can be seen in Figure 2. By adjusting σ, Hessian determinants can be computed in different scale spaces. This is a concept that SURF draws from the Scale-Invariant Feature Transform (SIFT) from Lowe et al [11].

By detecting features in different scale spaces, SIFT and by proxy, SURF are robust to changes in scale. SIFT accomplished this by downsampling the image to detect at lower order scale spaces. SURF improved on this approach for speed by instead scaling the kernel.



(a) $\frac{\partial^2 G}{\partial x_1^2}$ **(b)** $\frac{\partial^2 G}{\partial x_1 x_2}$ **(c)** $\frac{\partial^2 G}{\partial x_2^2}$

Figure 2: 9×9 **discrete approximations of the Gaussian second order derivative with** $\sigma = 1.2$

Another speed optimization presented in SURF takes advantage of the form of the discrete kernels. Since the approximated kernels are composed of rectangles of constant value, they can be decomposed into a set of box filters. Box filters can be computed quickly with the use of integral images. The general form of the integral image of an $M \times N$ image I is shown in Equation 6.

$$\int I = \begin{bmatrix} I(0,0) & \sum_{m=0}^{1} I(0,m) & \cdots & \sum_{m=0}^{M} I(0,m) \\ \sum_{n=0}^{1} I(n,0) & \sum_{n=0}^{1} \sum_{m=0}^{1} I(n,m) & & \vdots \\ \vdots & & \ddots & \vdots \\ \sum_{n=0}^{N} I(n,0) & \cdots & \cdots & \sum_{n=0}^{N} \sum_{m=0}^{M} I(n,m) \end{bmatrix} \tag{6}$$

Integral images decrease the computational complexity of finding the sum of an area in the input image. The computational

complexity of strict kernel convolution at a point scales with the size of the kernel and is of the order $O(N^2)$ where the kernel is $N \times N$. A box filter can be decomposed into finding the sum of an area in the image and scaling it. Finding the sum from the integral image can be performed in $O(1)$ as shown in Equation 7.

$$\sum_{n=w}^{y}\sum_{m=x}^{z} = \int I(y,z) - \int I(w-1,z) - \int I(y,x-1) + \int I(w-1,x-1) \tag{7}$$

This makes the computation of the Hessian matrix scale only in terms of the image size, yielding no additional penalty for operating on different scale spaces.

2.1.2 SURF Implementations for FPGA Devices. The relatively low computational complexity makes SURF a popular choice for FPGA applications. Battezzati et al. present an architecture using accumulators for computing the integral image pipelined through the Hessian computation and storing detected feature points in a first in, first out (FIFO) cache [1]. These are matched against a stored set of feature points. Chen et al. present improvements on this approach by parallelizing the computation of different scale spaces [6]. The implementation in this thesis follows these approaches with some additional improvements for speed based on the use case of matching against another image rather than a stored set of features.

2.2 Iterative Closest Point Algorithm

The crucial step in aligning the two images is the computation of an affine transform mapping one image into the space of the other. Once the images are aligned, they can be fused. A combination of SURF and the iterative closest point algorithm are used to compute this transform. Iterative closest point was designed as a method for aligning 3-D point cloud data [7]. In its simplest form, the algorithm follows the following steps:

(1) Each point in the set of points to be transformed is matched against the closest (usually Euclidean distance) point in the reference set of points.
(2) A transformation is estimated to minimize the distance between the transform set and their matches in the reference set of points.
(3) The transform is applied to the points.

This process is repeated, converging to the local minimum that is the match between the two point sets.

Let $X_r(i)$ be the i^{th} point in the set of reference points onto which the transform points, $X(j)$, where j is the index of the closest point to $X_r(i)$ in X, will be projected. The general form for this transformation M is shown in Equation 8, and the expanded matrix form can be seen in Equation 9. In the expanded matrix form M is decomposed into R, a 2×2 rotation matrix, and T, a translation offset.

$$X_r(i) = X(j) \cdot M \tag{8}$$

$$\begin{bmatrix} X_r(i)_1 \\ X_r(i)_2 \\ 1 \end{bmatrix} = \begin{bmatrix} X(j)_1 & X(j)_2 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & R_{12} & T_1 \\ R_{21} & R_{22} & T_2 \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

This equation only solves for M for a single point relation, but can be restructured to contain the whole set of N points as shown in Equation 10. In this equation P is a function $P : i \rightarrow j$ mapping the closest points in each set.

$$\begin{bmatrix} X_r(0)_1 \\ X_r(0)_2 \\ X_r(1)_1 \\ X_r(1)_2 \\ \vdots \\ X_r(N-1)_1 \\ X_r(N-1)_2 \end{bmatrix} = \begin{bmatrix} X(P(0))_1 & X(P(0))_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & X(P(0))_1 & X(P(0))_2 & 1 \\ X(P(1))_1 & X(P(1))_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & X(P(1))_1 & X(P(1))_2 & 1 \\ \vdots & & & & & \vdots \\ X(P(N))_1 & X(P(N))_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & X(P(N))_1 & X(P(N))_2 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{11} \\ R_{12} \\ T_1 \\ R_{21} \\ R_{22} \\ T_2 \end{bmatrix} \tag{10}$$

Solving for M in this way maps all points in X to their closest points in X_r based on an orthogonal projection with the error minimized in a least-square sense.

In this form, the transform M will include shear transformations and non-uniform scaling. The computation can be simplified by forcing the second basis vector in R to be orthogonal to the first. The two cameras are expected to be be physically in the same plane, and as such, non-uniform scaling and shear transformations are not expected for alignment. By setting $R_{21} = -R_{12}$ and $R_{22} = R11$, the computation of M can be reduced as shown in Equation 11.

$$\begin{bmatrix} X_r(0)_1 \\ X_r(0)_2 \\ X_r(1)_1 \\ X_r(1)_2 \\ \vdots \\ X_r(N-1)_1 \\ X_r(N-1)_2 \end{bmatrix} = \begin{bmatrix} X(P(0))_1 & X(P(0))_2 & 1 & 0 \\ X(P(0))_2 & -X(P(0))_1 & 0 & 1 \\ X(P(1))_1 & X(P(1))_2 & 1 & 0 \\ X(P(1))_2 & -X(P(1))_1 & 0 & 1 \\ \vdots & & & \vdots \\ X(P(N-1))_1 & X(P(N-1))_2 & 1 & 0 \\ X(P(N-1))_2 & -X(P(N-1))_1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{11} \\ R_{12} \\ T_1 \\ T_2 \end{bmatrix} \tag{11}$$

If scaling is disallowed, and the transformation consists of only a translation and a rotation, this can be reduced further. Consider first computing the centroids of the point sets as in Equation 12.

$$C = \frac{1}{N}\sum_{i=0}^{N-1} X(i) \tag{12}$$

Based on the closest point matching, the covariance matrix H can be computed as in Equation 13.

$$H = \sum_{i=0}^{N-1} (X(P(i)) - C) \cdot (X_r(i) - C_r)^T \tag{13}$$

The singular value decomposition $U\Sigma V = SVD(H)$ can be used to compute the rotation $R = VU^T$, where the translation is the distance between the centroids.

The performance of iterative closest point can be further improved by making it more sensitive to errors. Chetverikov introduced a variant of iterative closest point referred to as trimmed iterative closest point (TrICP) [8]. TrICP is more robust to errors by eliminating points that introduce error into the matching. Some detected features points will not have correspondences between images. By eliminating these points, the overall error can be reduced to compute a more accurate transform.

2.3 Singular Value Decomposition (SVD)

Let Equation 11 be of the form $X = Q \cdot M$. In order to compute the transform M, the equation must be restructured as in Equation 14.

$$Q^{-1}X = M \tag{14}$$

Q is not a square matrix, and as such is not invertible, but its pseudoinverse can be used in this instance. The pseudoinverse is computed through the use of singular value decomposition. The use of singular value decomposition to compute the transform is the source of the least squares fitting achieved by iterative closest point.

This is necessary for computing transforms with more degrees of freedom, however for rigid body transformations, it is sufficient to find the singular value decomposition of the covariance matrix H, a 2×2 matrix for which a closed form solution does exist[3] and is shown in Equation 15.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} cos\beta & sin\beta \\ -sin\beta & cos\beta \end{bmatrix} \begin{bmatrix} w_1 & 0 \\ 0 & w_2 \end{bmatrix} \begin{bmatrix} cos\gamma & sin\gamma \\ -sin\gamma & cos\gamma \end{bmatrix} = \begin{bmatrix} E & H \\ -H & E \end{bmatrix} + \begin{bmatrix} F & G \\ G & -F \end{bmatrix}$$
$$\frac{w_1 + w_2}{2} = \sqrt{E^2 + H^2}$$
$$\frac{w_1 - w_2}{2} = \sqrt{F^2 + G^2}$$
$$\gamma - \beta = tan^{-1}(G/F)$$
$$\gamma + \beta = tan^{-1}(H/E) \tag{15}$$

2.3.1 SVD Implementations for FPGA Devices. Singular value decomposition can be performed on FPGAs by cascading a set of 2×2 cells [13]. Ledesma-Carillo et al. present a hardware efficient algorithm for computing singular value decompositions on large matrices using one-sided Jacobi rotations for computing SVD on arbitrary $M \times N$ matrices [9]. One of these approaches is necessary if the sensor fusion must correct for scale or shear. It is worth noting that these approaches both require high utilization of the FPGA, and as such may be difficult to implement for large numbers of feature points.

In contrast, if only a rigid body transformation is required, then Equation 15 can be implemented trivially using CORDIC approximations for the square root and arctangent functions.

2.4 Laplacian Fusion

Laplacian pyramids of images have their origin as a strategy for image encoding [4]. A Gaussian blur is applied to the image, and the image is downsampled to half of its original size. At each level above the lowest level of the Gaussian pyramid, the level below is upsampled to match the scale of the current level. The difference between the upsampled image and the current scale level image is the Laplacian of the image. At a single level, the Laplacian can be thought of as the error introduced by applying a Gaussian and Box filter.

The property of the Laplacian that makes it ideal for fusion is its ability to capture the high frequency components in an image through the use of simple kernel operators that are easily implemented in hardware. The difference between a blurred image and the original will have higher magnitude in the areas where the image was sharpest.

The fusion of two images can be thought of as a function of the two images X and Y of dimension $M \times N$ where $Z = f(X, Y)$, a single image of dimension $M \times N$. A naive approach to fusion would be to compute the Laplacians and use their magnitudes to select a pixel from either X or Y as shown in Equation 16. In this context, $|L(x)|$ represents the absolute value of the Laplacian of the input.

$$Z(i,j) = \begin{cases} X(i,j) & |L(X(i,j))| \geq |L(Y(i,j))| \\ Y(i,j) & otherwise \end{cases} \tag{16}$$

This approach does not account for variations in colorspace between the two images. More saturated images would likely have a higher valued Laplacian simply because it is brighter, therefore having higher magnitudes at individual pixels. This approach also will not facilitate smooth stitching of the images. Contiguous regions of selection from one image will be adjacent to regions from the other with no transition, producing a grainy effect at areas of high frequency.

A more correct approach would involve using the Laplacian in a weighted sum to combine the pixels of the images, rather than simply selecting them, as shown in Equation 17.

$$Z(i,j) = \frac{|L(X(i,j))|}{|L(X(i,j))|+|L(Y(i,j))|} \cdot X(i,j) + \frac{|L(Y(i,j))|}{|L(X(i,j))|+|L(Y(i,j))|} \cdot Y(i,j) \tag{17}$$

3 PROPOSED METHOD

The implementation of this design was performed on a ZedBoard Zynq-7000 ARM/FPGA SoC Development Board.

3.1 Camera Interface

The design was implemented using a pair of OV7670 VGA cameras. These cameras feature an I^2C interface for configuration, and generate hsync and vsync VGA timing signals along with 8 bits of data. They can be configured to output 16-bit RGB(565) with half of the RGB signal sent on each clock pulse. Configured this way, the camera outputs a resolution of 640 by 480 pixels at $30Hz$.

3.2 Streaming Kernel Operators

Performing kernel convolution in real time is complicated by the fact that a pixel has data dependencies on its neighbors. As such, the input data must be buffered until all of the necessary data is available to perform convolution at a given point. The buffer size must be at least $N \times W + M$ where the kernel is $M \times N$ and the image is of width W. On FPGAs, this minimally sized buffer can be implemented using LUTRAM which also has the advantage of being able to act as a set of shift registers. The convolution multipliers and adders can be attached to a single set of cells, and the data can be shifted through the array as it streams. The architecture for this scan chain approach for computing 3×3 kernel convolution can be seen in Figure 3.

This scan chain approach to kernel convolution is a fast and utilization efficient approach to performing kernel convolution on streamed data. However, LUTRAM on most FPGAs is a limited resource, and as such, large kernels and image widths will make it difficult to implement a design. Implementing an LUTRAM scan

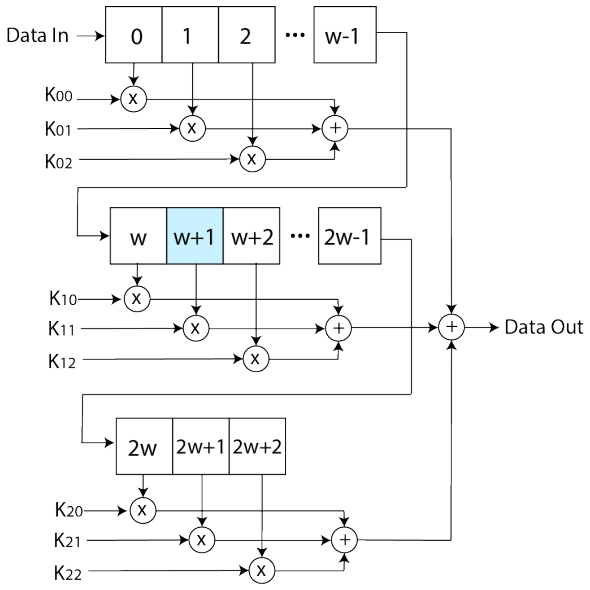
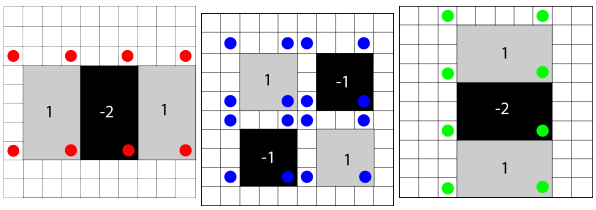


Figure 3: Block diagram for computing 3×3 kernel convolution on a data stream using a scan chain

chain for performing operations on 9×9 kernels with an image width of 640 pixels on the chipset used for this design consumed 96% of the available LUTRAM. Since it was necessary to implement two hessian operators, one for each camera, an approach was devised to utilize the slower, but more plentiful block RAM.

3.2.1 Hessian Kernel Operators. The kernels discussed in this section are the 9×9 discrete approximations shown in Figure 2. If straightforward kernel convolution were to be implemented, it would require sampling all points with non-zero kernel values which would not be realistic for performing these operators in real time. Instead, the integral image is used to reduce the number of required sampling points. Recall that the integral image can be used to compute the area in a rectangle by sampling the corners, adding the bottom-right corner and top-left corner, and subtracting the bottom-left corner and top-right corner. As such, the required sampling points for the SURF kernels are highlighted in Figure 4.



(a) $\frac{\partial^2 G}{\partial x_1^2}$ (b) $\frac{\partial^2 G}{\partial x_1 x_2}$ (c) $\frac{\partial^2 G}{\partial x_2^2}$

Figure 4: 9×9 SURF kernels with marked integral sampling points

It can be observed in Figure 4 that the worst-case for the data dependencies is the bottom-right point in $\frac{\partial^2 G}{\partial x_2^2}$. If the Hessian determinant is to be computed with minimal latency, it should be computed when that point becomes available. In order to compute the integral at a point, the integral above, to the left, and to the top-left of the desired point must be sampled. From this, it can be concluded that thirty-four points must be sampled to compute the Hessian determinant (eight from $\frac{\partial^2 G}{\partial x_1^2}$, sixteen from $\frac{\partial^2 G}{\partial x_1 x_2}$, seven from $\frac{\partial^2 G}{\partial x_2^2}$, and three to compute the integral).

In order to do this in real time, the Hessian determinant must be computable within a pixel clock cycle. Given a resolution of 640 by 480 at 30 frames per second, this is a pixel clock speed of $12.5MHz$. In this implementation, block RAM has a latency of three clock cycles. In order to compute the determinant in a single pixel clock cycle, the Hessian block is clocked at $200MHz$, giving it 16 clock cycles for every pixel clock cycle. A $200MHz$ clock has a period of only $5ns$ which makes pipelining of instructions important in order to meet timing. A dual port block RAM was used, with addressing using the lower bits of y to map into a modular address space of 16 rows. The dual port RAM effectively allows for 32 read/write operations within the pixel clock period. An LUTRAM cache containing the integral values for the last row up to the top-left point required for the integral image reduces the number of memory reads for required points to 29, along with a single memory write to place the last value in the cache into the block RAM.

The process is pipelined by splitting up the determinant computation into small operations that execute as the data becomes available from the block RAM. It is worth noting that this implementation does not need to be modified to compute Hessian determinants in different scale spaces. Larger kernels still have the same number of points that must be sampled, but the required block RAM for the design does grow with the kernel size.

3.2.2 Average Filter Approximation of Single Level Laplacian Pyramid. In this design, only a single level of the Laplacian pyramid is used for fusion. In a single level, a Gaussian blur is applied to the image. It is downsampled, upsampled, and subtracted from the original. The downsampling and upsampling in a single level can be approximated as an average filter without performing the costly operation of modifying the resolution of the image. In this design, the Gaussian and average filters were implemented as a set of scan chains to perform 3×3 kernel convolution.

3.2.3 Application of Non-maximum Suppression. Though not a kernel operator, non-maximum suppression depends on the data points around it. Non-maximum suppression is to be applied to the Hessian determinant values in order to reduce them to a smaller, more precise set of features. In a 3×3 neighborhood, if the central pixel is not the maximum, it is set to zero. In this way, only the peak features remain for processing. The 3×3 neighborhood search was implemented as a 3×3 scan chain with comparison operators.

3.3 Computation of Transform from Detected Features

As features are generated, they are passed into a feature buffer as a tuple of their magnitude, x, and y coordinates. The buffer is made up of bubble-sort cells as shown in Figure 5.

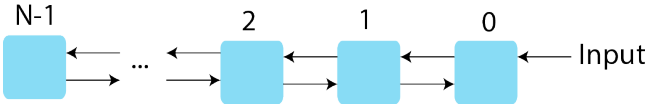


Figure 5: Bubble-sort architecture of the feature buffer

On the rising edge of the clock, the even cells swap values with their neighbor to the left if the cell contains a greater value than its neighbor. On the falling edge, the odd cells do the same. In this way, the highest valued elements propagate to the top of the buffer, and lower valued elements are dropped out of the bottom. The comparison criteria for this operation is the Hessian magnitude, and the sort is active for as long as features are being generated. At the end of the frame, the buffer contains only the points with the highest Hessian magnitudes.

Once the points have been collected, the transform set is compared to the reference set and correspondences are assigned based on euclidean distance to find the closest point. This is done via a brute-force search. Every point is compared to every other, and the closest one emerges.

For low numbers of feature points, the inefficiency of brute-force search is mitigated by its simplicity. In hardware, a brute force comparison like this does not suffer from additional time complexity since the comparisons all happen in parallel. However, the space complexity and fanout of the circuit goes up exponentially as more cells are added.

This architecture is only useful for small numbers of feature points. The high fanout of the brute force comparison quickly becomes unmanageable to route in most designs with more than a handful of features. In the case that this becomes difficult, it is possible to trade the space complexity of the design for time complexity by making the computation iterative.

In this approach, a match register holds the point that is the current closest match. A point from the reference set is loaded into a register, A. Then, each point in the transform set is loaded into a register, B, in turn. If the distance between the loaded value and the value in register A is less than the distance between the value stored in the match register and register A, then the loaded value from register B is placed into the match register. This is repeated for all elements in the reference set to find their closest points in the transform set.

3.4 Application of Transform to Real-Time Data

The reference image remains fixed and is outputted as normal. The address for the transform image is decomposed into x and y coordinates. This is multiplied through the computed transform, and the transformed coordinates are used to select pixels from the frame buffer. The reference pixel and transform pixel are then fused together to create the output stream.

The transform is recomputed at the end of the frame from the detected features and then applied to the next generation of incoming data. In this way, the transform is additively refined as in iterative closest point. The transform computed for a frame is concatenated with the transform from the last frame, eventually converging to a local error minimum for alignment in a least squares sense.

4 CONCLUSIONS

The proposed pipeline architecture for computing the Hessian determinants with minimal latency allows FPGA implementation of video alignment where the video sources fields of view varies with time. The technique has potential for future use in video stabilization where consecutive video frame fields of view vary due to camera jitters. The treatment of each frame as a generation of feature points to be transformed to a local error minimum as opposed to a more rigid architecture of performing multiple iterations on one feature set helps alleviate the overhead of the design, and make it realizable. The available chipset was not of sufficient size to perform the larger corrections of scale and shear due to the prohibitive nature of the instantiating large arrays of SVD solving cells. However, on a more robust chipset, scale and shear should also be correctable factors, as well as giving the ability to compute the transform from a larger set of feature points.

REFERENCES

[1] N. Battezzati, S. Colazzo, M. Maffione, and L. Senepa. SURF algorithm in FPGA: A novel architecture for high demanding industrial applications. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 161–162, March 2012.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[3] J. Blinn. Consider the lowly 2 x 2 matrix. *IEEE Computer Graphics and Applications*, 16(2):82–88, March 1996.

[4] P. Burt and E. Adelson. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications*, 31(4):532–540, April 1983.

[5] S. Chappell, A. Macarthur, D. Preston, D. Olmstead, B. Flint, and C. Sullivan. Exploiting Real-time FPGA Based Adaptive Systems Technology for Real-time Sensor Fusion in Next Generation Automotive Safety Systems. In *The IEE Seminar on Target Tracking: Algorithms and Applications 2006 (Ref. No. 2006/11359)*, pages 61–68, March 2006.

[6] W. Chen, S. Ding, Z. Chai, D. He, W. Zhang, G. Zhang, Q. Peng, and W. Luo. FPGA-Based Parallel Implementation of SURF Algorithm. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 308–315, December 2016.

[7] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, April 1992.

[8] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek. The Trimmed Iterative Closest Point algorithm. In *Object recognition supported by user interaction for service robots*, volume 3, pages 545–548 vol.3, 2002.

[9] L. M. Ledesma-Carrillo, E. Cabal-Yepez, R. d J. Romero-Troncoso, A. Garcia-Perez, R. A. Osornio-Rios, and T. D. Carozzi. Reconfigurable FPGA-Based Unit for Singular Value Decomposition of Large m x n Matrices. In *2011 International Conference on Reconfigurable Computing and FPGAs*, pages 345–350, November 2011.

[10] Hui Li, B. S. Manjunath, and S. K. Mitra. Multi-sensor image fusion using the wavelet transform. In *Proceedings of 1st International Conference on Image Processing*, volume 1, pages 51–55 vol.1, November 1994.

[11] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[12] Wencheng Wang and Faliang Chang. A Multi-focus Image Fusion Method Based on Laplacian Pyramid. *Journal of Computers*, 6(12), December 2011.

[13] Y. Wang, K. Cunningham, P. Nagvajara, and J. Johnson. Singular Value Decomposition Hardware for MIMO: State of the Art and Custom Design. In *2010 International Conference on Reconfigurable Computing and FPGAs*, pages 400–405, December 2010.