Image Processing, Electronics and Computers
L. Trajkovic et al. (Eds.)

doi:10.3233/ATDE240473

Optimization and FPGA Implementation of RANSAC Algorithm Based on HLS

Chenglong LIAN[a], Yanbing XUE[b], Xiaojie LI[a], Panshi HU[a] and Chong FENG[c,1]
[a]*School of computer and communication engineering, Dalian Jiaotong University, Dalian 116028, China*
[b]*College of automation and electrical engineering, Dalian Jiaotong University, Dalian 116028, China*
[c]*School of information and communication engineering, Dalian Minzu University, Dalian 116600, China*

Abstract. With the rapid development of artificial intelligence technology, research on autonomous driving technology is becoming increasingly popular. In the autonomous driving panoramic imaging system, it is necessary to use image stitching technology to perform real-time stitching of image information collected by multiple cameras around the vehicle body. Therefore, requirements for image data processing include large data volume, high speed, and low power consumption. The parallel computing characteristics of Field Programmable Gate Array (FPGA) can accelerate key image stitching algorithms. This article focuses on hardware acceleration of the RANSAC (Random Sample Consensus) algorithm in image stitching. The algorithm is designed in parallel using high-level synthesis (HLS) technology and optimized accordingly to find a suitable method for algorithm optimization. Comparing the FPGA algorithm circuit with the Open CV algorithm program, the hardware running time of the algorithm is only 2.816ms, and the processing speed is 38 times faster than the latter. The RANSAC algorithm implemented by FPGA meets the requirements of real-time image processing and can be applied in real-time image stitching systems.

Keywords. Image processing, hardware speedup, feature point matching, field programmable gate array; parallel optimization; embedded image processing

1. Introduction

In the process of automatic driving, when panoramic image system is collecting visual information, traditional single camera cannot meet the requirements of high quality and wide field of vision at the same time [1]. It is particularly important to use the high real-time image stitching technology to splice the images around the car body collected by multiple cameras in real time to obtain high-quality images with wide field of view. Therefore, real-time is the focus of image mosaic technology. Because of its unique parallel characteristics, FPGA can greatly improve the real-time performance of images, reaching the ability that is almost invisible to the human eye [2]. High level synthesis (HLS) technology is a method of converting high-level language code into hardware circuit, which can automatically convert C, C++functions into HDL code, thus greatly

[1] Corresponding Author: Chong FENG, fengchong@dlnu.edu.cn.

reducing the threshold of FPGA design, improving development efficiency and reducing costs [3].

In recent years, the hardware acceleration of FPGA image mosaic algorithms based on HLS has become increasingly prevalent. Zhang [4] and colleagues utilized the parallel characteristics of programmable gate arrays to hardware accelerate the FAST corner detection algorithm and Sobel edge detection algorithm, employing HLS high-level synthesis for design and optimization. Li [5] and colleagues developed the CLAHE image enhancement algorithm based on ZYNQ, capable of processing multiple sets of images and displaying them on HDMI monitors with minimal latency, thus demonstrating the real-time performance of image algorithm enhancement. Bouris [6] and colleagues leveraged FPAG to accelerate the implementation of the SURF feature detection algorithm, achieving a speed of 56 frames per second in standard video with 640 × 480 pixels. Furkan [7] and colleagues implemented the standard deviation module and main fusion module using HLS, achieving image fusion in hardware and combining two or more images through a color transformation process. When compared to other publications on image fusion, it is evident that utilizing HLS tools can reduce development time and enhance productivity.

During the image mosaic process, the matching results from feature point extraction often include a significant number of incorrect matches, which can be effectively reduced using the RANSAC algorithm [8]. However, when dealing with large data volumes, high outlier rates, or complex models, the traditional RANSAC algorithm experiences a significant slowdown in computational speed, making it impractical for real-time image processing [9]. Therefore, it is necessary to accelerate the RANSAC algorithm using an FPGA platform. This paper employs a hardware-software collaborative approach, leveraging the Vivado HLS development environment provided by Xilinx, to encapsulate the RANSAC algorithm IP (Intellectual Property) core within the image mosaic system. This approach facilitates FPGA hardware acceleration of the RANSAC algorithm, enhancing the real-time performance of image processing to meet the requirements of real-time image processing.

2. RANSAC Algorithm Principle and IP Core Design

2.1. Principles of RANSAC Algorithm

The RANSAC algorithm [10], also known as the Random Sampling Consensus algorithm, employs an iterative approach to estimate the parameters of a mathematical model from a dataset that includes outliers. This algorithm can be used to filter coordinate data pairs corresponding to image feature matches, and by utilizing a randomly sampled set of matching coordinate pairs, it calculates a homography matrix that allows for the seamless merging of two images for the purpose of image transformation.

2.2. RANSAC Algorithm IP Design

Based on the principles of the RANSAC algorithm, it functions to precisely match image feature points and compute image transformation parameters in image mosaic. The hierarchical structure of RANSAC algorithm's functions is depicted in Figure 1.

In Figure 1, the RANSAC algorithm calls a total of six functional function modules, they are checkSubset function, getSubset function, Kernel function, findInliers function, computeReprojError function, and RANSACUpdateNumIters function.

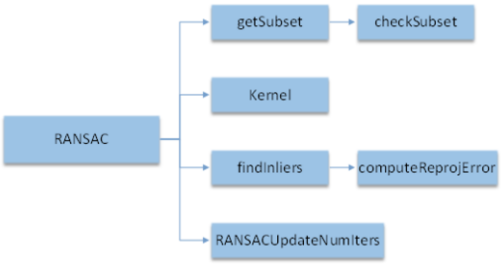


Figure 1. RANSAC function call hierarchy.

Figure 2 is the program flow chart of RANSAC algorithm IP design.

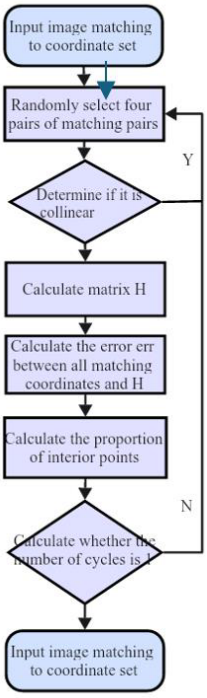


Figure 2. Procedure flow chart of RANSAC IP calculation design.

The specific steps are as follows:

Step 1: First input the image matching set R, randomly select four matching pairs to determine whether it is collinear.

Step 2: If collinear, return to randomly select four matching pairs; if it is not collinear, the homologous matrix H is calculated by selecting four pairs of matching coordinate points.

Step 3: Using the parameters of matrix H, calculate the fitting error err of all input image matching pairs and H.

Step 4: Use error err to judge the inner point proportion e, and then use the inner point proportion e and the last calculated cycle number Iters to calculate the next cycle number NIters.

Step 5: Determine whether NIters is equal to 1, if not, return to randomly select four matching pairs to continue the calculation; If it is 1, the loop is no longer used, and the final homography matrix H parameter is output as the optimal image transformation parameter model.

2.3. RANSAC Algorithm HLS Source Program Verification

As shown in Figure 3, the coordinates of the feature matching pair in the figure are the test data of the error calculation function, which comes from the partial coordinate data filtered by RANSAC algorithm when OpenCV calculates image stitching. The function uses the output result of Kernel function as the model parameter to conduct model fitting test and comparison with the input data. The image feature matching pairs are 30 in total.

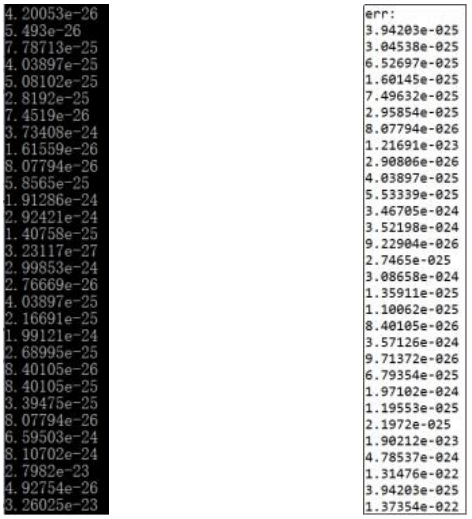
```
const Point_hls M[30] = {
        {320, 189}, {342, 213},{385, 214},{404, 253},{328, 205},{453, 208},
        {441, 255},{440, 223},{385, 229},{458, 227},{408, 250},{427, 237},
        {462, 227},{418, 235},{458.4, 225.6},{328.8, 212.4},{327.6, 226.8},{454.8, 198},
        {453.6, 207.6},{385.2, 228},{335.52, 216},{384.48, 227.52},{341.28, 227.52},{440.64, 221.76},
        {341.28, 213.12},{427.68, 205.92},{385.344, 228.096},{433.728, 255.744},{440.64, 222.912},{393.984, 259.2}};

const Point_hls m[30] = {
        {31, 161},{53, 185},{96, 187},{115, 226},{39, 178},{163, 182},
        {151, 228},{151, 196},{96, 202},{168,201},{119, 223},{137, 211},
        {172, 201},{129, 208},{168, 200.4},{39.6, 184.8},{38.4, 199.2},{164.4, 172.8},
        {163.2, 181.2},{96,201.6},{46.08, 188.64},{96.48, 201.6},{51.84, 201.6},{151.2, 195.84},
        {51.84, 185.76},{138.24, 180},{96.768, 200.448},{145.152, 228.096},{150.336, 196.992},{103.68, 232.243}};
```

Figure 3. Error calculation function test data.

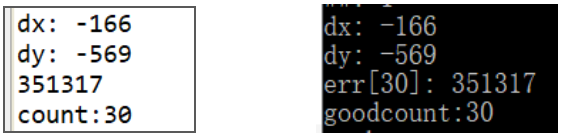
The output results are shown in Figure 4. Figure 4 (a) shows the output results of OpenCV with 30 correct data input for this function, and Figure 4 (b) shows the output results of HLS with 30 correct data input for this function. As can be seen from the figure, the errors are very close to zero because they are internal point data after screening. However, there are also errors in the output results of OpenCV and HLS, so an obviously wrong data is added as shown in Figure 5, and it is observed whether the results calculated by the error calculation function of this data are consistent.

As can be seen from Figure 5, when the error obtained when the error data is fitted with the homologous matrix model is greater than the threshold value, the data is the outer point and the inner point set is not included, and the error results are completely consistent, which proves that the HLS source program of the error calculation function is successfully designed.



(a) OpenCV output results (b) HLS output results

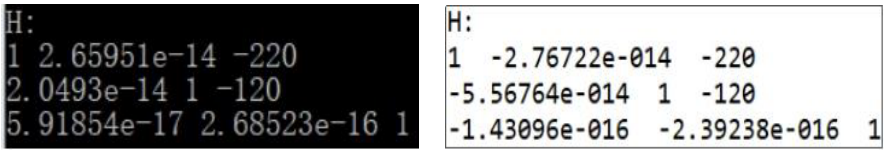
Figure 4. Comparison of output results of error calculation functions



(a) OpenCV output results (b) HLS output results

Figure 5. Comparison of results after adding error data.

Figure 6 shows the comparison between the output results of RANSAC function in OpenCV and HLS. It can be seen from the figure that the homography matrix output by the HLS source program of RANSAC function is completely consistent with that of OpenCV, except that the HLS source program design of RANSAC is correct when some parameter in the matrix is close to zero.



(a)OpenCV output results (b) HLS output results

Figure 6. RANSAC function output comparison.

3. Method of algorithm optimization under Vivado HLS development environment

The HLS tool contains various instructions that guide hardware synthesis to produce more efficient designs [11]. These instructions include loop pipelining and loop expansion, and array partitioning. Since most algorithms contain many loops and arrays of data, finding an optimal set of instruction Settings can be a difficult task [12].

3.1. RANSAC algorithm optimization

3.1.1. Loop optimization

RANSAC algorithm is the processing of image data. In the IP design of the algorithm, the for loop is used in the code to complete the traversal operation of each coordinate pair, and the cycle calculation is completed in serial mode, which will greatly reduce the computing efficiency. Therefore, the algorithm designed in C/C++ style does not have high parallelism when directly transplanted to FPGA environment [13]. In an FPGA environment, cycles can be accelerated in a variety of ways. Vivado HLS provides a variety of parallel optimization methods, including PIPELINE.

PIPELINE is the most commonly used method for loop instructions [14]. The specific principle is shown in Figure 7. Pipelining allows the for loop to execute operations in parallel: the second read operation does not need to wait for the compute and write operations to complete before execution begins. It can be compared that the cycle after the PIPELINE is reduced to 4 clock cycles.

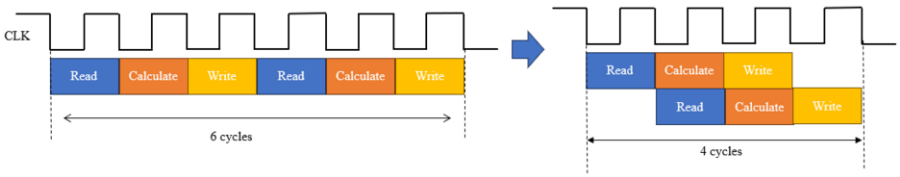


Figure 7. Pipeline instruction optimization diagram.

The core loop of RANSAC algorithm is optimized and compared in this design. Now we optimize the findliers_labe1 loop for finding the number of inside points. As shown in Figure 8 (b), the cycle does not reach the ideal state, and the cycle interval of 8 is larger.

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
+ findInliers_label1	1568	1568	14	-	-	112	no
+ RANSAC_hls_label32	149	149	4	-	-	37	no
+ RANSAC_hls_label33	18	18	2	-	-	9	no
- RANSAC_hls_label4	224	224	2	-	-	112	no

(a) Before loop optimization

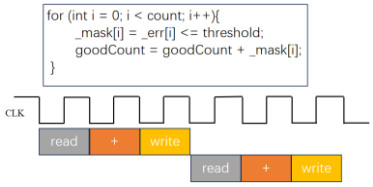
	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
+ findInliers_label1	899	899	12	8	1	112	yes
+ RANSAC_hls_label32	149	149	4	-	-	37	no
+ RANSAC_hls_label33	18	18	2	-	-	9	no
- RANSAC_hls_label4	224	224	2	-	-	112	no

(b) After cycle optimization

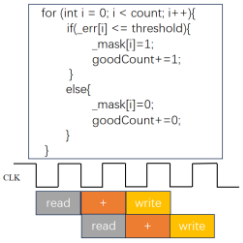
Figure 8. Comprehensive report before and after pipeline optimization.

As shown in Figure 9 (a), there is a dependency within the findliers_labe1 loop that prevents PIPELINE optimization. The loop dependency is due to the presence of the good Count variable. Since each iteration operation needs to write the result of the array _mask sum into this variable, and each iteration also reads this register value. Therefore, a read and write dependency is formed in the iterative operation. For the dependencies of the loop, conditional statements are used to separate the relationship

between good Count variable and _mask array, and the variable good Count does not depend on _mask array. Continue to optimize the cycle PIPELINE. The optimization results are shown in Figure 9(c), and the cycle interval reaches the ideal state of 1. After optimization, the total cycle delay is 114 clock cycles, compared with 1568 clock cycles before optimization, PIPELINE instruction optimization greatly reduces the delay.



(a) Pre-optimization code



(b) Optimized code

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
+ findInliers_label1	114	114	3	1	1	112	yes
+ RANSAC_hls_label32	112	112	2	1	1	112	yes
+ RANSAC_hls_label33	9	9	2	1	1	9	yes
- RANSAC_hls_label4	112	112	2	1	1	112	yes

(c) Optimization results reporting

Figure 9. Loop dependent pipeline optimization.

3.1.2. Array optimization

After the array is synthesized in HLS, it is rendered as memory (RAM, ROM, or FIFO). In order to improve the parallelism of array data access and the parallel execution ability of the program, the continuous data can be transformed into block or ring storage by changing the storage mode of array data.

As shown in Figure 10, for the RANSAC function structure array m1, m2 also uses the ARRAY_PARTITION instruction complete partition mode, and uses dual-port ram to improve the read speed.

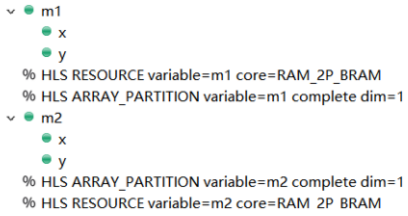


Figure 10. Complete split array instruction code in RANSAC.

3.1.3. Interface optimization

The main function RANSAC interface consists of four data flow interfaces. The interface can be configured as the s_axi interface protocol to obtain the maximum transfer speed. Figure 11 shows the port of the RTL configured as the s_axi interface protocol.

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_AXILiteS_AWVALID	in	1	s_axi	AXILiteS	array
s_axi_AXILiteS_AWREADY	out	1	s_axi	AXILiteS	array
s_axi_AXILiteS_AWADDR	in	13	s_axi	AXILiteS	array
s_axi_AXILiteS_WVALID	in	1	s_axi	AXILiteS	array
s_axi_AXILiteS_WREADY	out	1	s_axi	AXILiteS	array
s_axi_AXILiteS_WDATA	in	32	s_axi	AXILiteS	array
s_axi_AXILiteS_WSTRB	in	4	s_axi	AXILiteS	array
s_axi_AXILiteS_ARVALID	in	1	s_axi	AXILiteS	array
s_axi_AXILiteS_ARREADY	out	1	s_axi	AXILiteS	array
s_axi_AXILiteS_ARADDR	in	13	s_axi	AXILiteS	array
s_axi_AXILiteS_RVALID	out	1	s_axi	AXILiteS	array
s_axi_AXILiteS_RREADY	in	1	s_axi	AXILiteS	array
s_axi_AXILiteS_RDATA	out	32	s_axi	AXILiteS	array
s_axi_AXILiteS_RRESP	out	2	s_axi	AXILiteS	array
s_axi_AXILiteS_BVALID	out	1	s_axi	AXILiteS	array
s_axi_AXILiteS_BREADY	in	1	s_axi	AXILiteS	array
s_axi_AXILiteS_BRESP	out	2	s_axi	AXILiteS	array
ap_clk	in	1	ap_ctrl_hs	RANSAC_hls	return value
ap_rst_n	in	1	ap_ctrl_hs	RANSAC_hls	return value
interrupt	out	1	ap_ctrl_hs	RANSAC_hls	return value

Figure 11. RTL port configured as s_axi interface protocol.

3.2. RANSAC Algorithm Optimization Results Analysis

This paper balances resource utilization and speed, and pays more attention to acceleration algorithm. The final optimization results of RANSAC algorithm are shown in Table 1. In the optimized RANSAC1 solution, BRAM_18K occupies 184 resources, and the resources occupied by LUT lookup table and FF are 86012 and 76849, respectively. Compared with the pre-optimized RANSAC0 solution, the hardware resources are slightly increased. Table 1 also shows the delay comparison of IP cores. Compared with the non-optimized scheme, the minimum clock cycle is reduced by 6.19% and the maximum clock cycle is reduced by 13%. The optimization method can improve the efficiency of the image Mosaic system without increasing the hardware resources.

Table 1. Comparison of resources and clock cycles before and after optimization

	Required resources LUT FF RAM(bit)	**Latency (min)**	**Latency (max)**
RANSAC0	77299 71399 174	8051	2113477
RANSAC1	86012 76849 184	7552	1828347

4. System Implementation and Analysis

4.1. System Construction

After Vivado HLS generates the RANSAC algorithm IP core, it is added to Vivado's Block Design and connected to other modules. The system structure block diagram of RANSAC algorithm is shown in Figure 12.

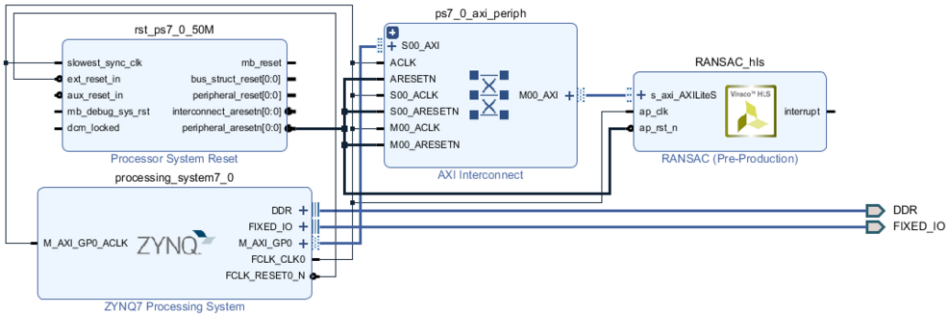


Figure 12. RANSAC algorithm verification system structure.

Create the IP core of the ZYNQ system, configure the DDR, UART0, and lead out the 50MHz PL clock and a reset signal. The whole system is connected through AXI Interconnect IP to realize data communication between PS and PL. Finally, the driver is written using SDK platform. The PS part is configured in the SDK environment, which includes configuring the IP of the algorithm, as well as configuring the input and output of the data. After the program is burned, the data can be output in the SDK platform to verify the correctness of the algorithm.

4.2. System Test and Analysis

Figure 13 shows the output result of hardware operation. 112 pairs of input coordinate data are double type data, which is converted into hexadecimal data and written into the sorted register address first, as shown in Figure 13 (a). The data results calculated by RANSAC algorithm are shown in Figure 13 (b), which is converted into double data output as shown in Figure 13 (c). It can be seen that the hardware operation results are consistent with the HLS simulation results, which proves that the hardware functions are correct.

0x0400 <Hex Integer> | 0x0400 : 0x400 <Hex Integer>

Address	0 - 3	4 - 7	8 - B	C - F
00000400	00000000	00E07240	00707240	00C07140
00000410	00207340	00D07240	00707240	00207240
00000420	00707140	00C07040	00107140	00E07140
00000430	00107140	00806D40	00607340	00406C40
00000440	00D07240	00107040	00307140	00007040
00000450	00B07040	00406C40	00C06B40	00406B40
00000460	00606C40	00E06F40	00406B40	00107140
00000470	00407240	66267240	66067140	33537140

(a) Write register data

0x1400 : 0x1400 <Hex Integer> | New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
00001400	21EA3E00	08C7813F	39D067C0	75FF563F
00001410	094BF03F	9EE8E3BF	3342023F	DAC0133F
00001420	0000F03F	00000000	79CA6ABA	00000000
00001430	E07E8B11	00000000	EF4FC0AE	00000000

(b) Homography Matrix Results

```
H[0]:1.018960
H[1]:0.008680
H[2]:-190.507000
H[3]:0.001404
H[4]:1.018320
H[5]:-0.622146
H[6]:0.000035
H[7]:0.000075
H[8]:1.000000
```

(c) Convert to double data output result

Figure 13. Hardware operation output.

The unified input parameters of RANSAC algorithm in OpenCV and HLS are 112 pairs of image coordinates obtained after rough matching with ORB algorithm. The data in Table 2 are the IP check ratio results of OpenCV and RANSAC algorithm. It can be seen that the obtained 8 parameter data are not exactly the same. Since the parameters of the homologous transformation matrix calculated by the RANSAC algorithm IP kernel are not exactly the same as OpenCV, the two-dimensional correlation coefficient corr2 function of MATLAB is used to calculate the similarity of the matrix.

The formula for calculating the two-dimensional correlation coefficient between matrices is as follows:

$$r = \frac{\sum_M \sum_N (A_{MN} - \overline{A})(B_{MN} - \overline{B})}{\sqrt{\left(\sum_M \sum_N (A_{MN} - \overline{A})^2\right)\left(\sum_M \sum_N (B_{MN} - \overline{B})^2\right)}} \tag{4.1}$$

Where M and N are the number of rows and columns of the matrix, are the average value of all elements of matrix A, are the average value of all elements of matrix B, and r is the correlation coefficient of matrix A and B.

When A and B are not correlated, that is, A and B are independent; The closer r is to 1, the more correlated A and B are. Until then, A and B are linearly correlated, that is, A and B are linearly proportional.

Table 2. Comparison of results of RANSAC algorithm

Group Name	The H parameter of OpenCV ($h_{33}=1$)	Hardware output H parameter($h_{33}=1$)	matrix Similarity
C	1.025155704749668 0.001839434873948079 -408.5725443207193 -0.02256456101563815 1.00035943978395 7.94349623216679 -2.124402347316837e-05 1.228399338248041e-05	0.982399 -0.00370534 -387.159 -0.0262459 0.970867 10.4707 -6.09826e-005 2.18133e-006	1.0000
D	0.9635532197542308 0.007343239790621947 -178.7862994757098 -0.007145752763137457 0.9835752885501619 1.588615779543054 -9.285904610154091e-05 5.125834220430886e-05	1.01896 0.00868041 -190.507 0.00140368 1.01832 -0.622146 3.48255e-005 7.5353e-005	0.9999

The correlation coefficient between the homography matrix parameters and OpenCV obtained by RANSAC algorithm HLS source program is calculated. The closer to 1, the more correct the designed algorithm is proved. If the error is not large, it can be applied to the real test graph group to view the output homography matrix results and picture stitching effects, and judge the practical application results of the RANSAC algorithm.

In Table 2, it is calculated that the similarity of the two matrices in group C is 1, and the similarity of the two matrices in group D is 0.9999. Therefore, the IP core of the RANSAC algorithm designed in this paper can be considered effective, and the algorithm can be applied in the image stitching system instead of the original algorithm to test its practical application effect.

By setting the timing function in the program, you can calculate the data transfer and RANSAC algorithm calculation processing time. OpenCV running time is 110.698ms, RANSAC algorithm in hardware running time is 2.816ms, hardware time is about 1/39 of the software time, it can be seen that the use of FPGA hardware to accelerate RANSAC algorithm is more efficient.

Table 3 compares this article to hardware implementations of other RANSAC algorithms. In terms of resources, the RANSAC algorithm IP designed in this paper consumes far less resources in LUT and RAM than other research results in the table, while the FF resource consumption increases. In terms of speed, the algorithm time of literature [15] and literature [16] is 26.8ms and 2.063ms, which is much lower than this design. Literature [17] and literature [18] have the same purpose as the RANSAC algorithm IP designed in this paper, which is used to optimize image feature point matching. The processing speed designed in this paper is much higher than that in the literature [17]. Although reference [18] is slightly higher than the design in this paper, it only supports a fixed number of 128 matching pairs. The feature point screening of arbitrary matching pairs is designed in this paper.

Table 3. Hardware circuit performance comparison of RANSAC algorithm

	Clock rate (MHz)	**Required resources**			**Processing quantity**	**Processing time**(ms)
		LUT	**FF**	**RAM**		
Mao [15]	100	—	—	—	—	26.8
Jiang [16]	100	91870	66750	—	1400	2.063
Vourvoulakis [17]	15	98980	11438	5632	128	22.87
Vourvoulakis [18]	15	90326	11430	2448	128	0.5458
Proposed	100	88789	79298	184	112	2.816

5. Conclusions

This paper employs Vivado to accomplish the IP encapsulation design for the RANSAC refinement algorithm, and identifies an optimization method apt for the RANSAC algorithm. Upon completion of the design, a RANSAC algorithm system based on FPGA was constructed, underwent hardware verification, design performance evaluation, and a comparison of software and hardware. The results indicate that the FPGA achieves low power consumption and consumes fewer resources. By utilizing a hardware-accelerated RANSAC algorithm IP core, the hardware execution time for image processing is shortened to 2.816ms, a significant reduction to 2.54% of the software execution time, ensuring the algorithm's real-time performance.

The research work of this paper is relatively limited, and it is only a beginning work, which needs to be further improved and deepened. The work that deserves further study mainly includes:

(1) The part of image acquisition by camera is not designed in this paper, and the part of image acquisition by camera can be added to the image stitching algorithm in the future.

(2) The input and output part of the RANSAC algorithm source program is designed to use a coordinate array, not an image array, and the input and output of the algorithm can be improved later, so that the algorithm becomes a complete middle part of the image stitching processing.

(3) This paper only completes the hardware acceleration of some algorithms in image stitching, and the hardware acceleration of the whole image stitching algorithm can be realized later.

Acknowledgement

First of all, I would like to thank my mentor Xue Yanbing for his careful training of me. She gave me unselfish guidance and took pains to help me revise and improve my thesis. I would also like to thank teacher Feng Chong for solving the difficulties encountered in my research.

I would also like to thank the students of the research group, without your support and assistance, I could not solve these difficulties and doubts.

Finally, I would like to thank the fund project: Applied Basic Research Project of Liaoning Provincial Department of Science and Technology (2022JH2/101300267) for its support.

Reference

[1] Yan Xiwen. Research and implementation of embedded vehicle panoramic vision system. Guangdong University of Technology, 2022.

[2] Zhang Liguo, Lei Xuanrui, Jin Mei, et al. Design of circuit board defect detection System based on image processing. High-tech Communications, 2019, 34(02): 209-217.

[3] Zhang Canyu, Feng Ansong, Zhang Hualiang, et al. Design of an image processing hardware acceleration system based on FPGA. Computer Engineering and Design, 2024, 45(03): 723-731.

[4] Li Xiaoqi, Wang Yunfeng, Wu Qiannan, et al. Real-time acceleration design of CLAHE image enhancement Algorithm based on ZYNQ. SCM and Embedded System Applications, 2019, 23(11): 49-53.

[5] Wang Yunfeng, Fan Zhengji, He Xin, et al. Design of real-time endoscopic dark area enhancement algorithm based on Vivado HLS. Electronic measurement technology, 2022, (23): 31-37.

[6] Bouris Dimitris, Antonis Nikitakis, and Ioannis Papaefstathiou. Fast and Efficient FPGA-based Feature Detection Employing the SURF Algorithm. IEEE International Symposium on Field-programmable Custom Computing those. IEEE, 2010: 3-10.

[7] Aydin Furkan, H. Fatih Ugurdag, Vecdi Emre Levent, Aydin Emre Guzel, N. Fajar R. Annafianto, M. Akif Ozkan, Toygar Akgun, and Cengiz Erbas. Rapid Design of Real-Time Image Fusion on FPGA using HLS and Other Techniques. 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA). IEEE, 2018: 1-6.

[8] Cao Haojie, Zhang Xu. A real-time microscopic image Mosaic Method based on Improved ORB Algorithm. Automation & Instrumentation, 2024, (03): 18-25.

[9] Zhao Z, Wang F, Ni Q. An FPGA-based Hardware Accelerator of RANSAC Algorithm for Matching of Images Feature Points. 2019 IEEE 13th International Conference on ASIC (ASICON). IEEE, 2019.

[10] Wang Jiachen, Ye Zhurun, Ou Xin, et al. Research on GPU-based parallel ICP point cloud registration algorithm. Journal of Hefei University of Technology (Natural Science Edition), 2023, 46(11): 1501-1505.

[11] Zhao Sijie, Gao Shengshang, Wang Rugang, et al. Image zooming based on HLS and PYNQ hardware accelerator design. Journal of yancheng institute of technology (Natural Science Edition), 2023, 4 (02): 55-60.

[12] Wei Sulun, Tao Qingchuan. Implementation of Mobile-Net accelerator based on HLS. Modern Computer, 2023, 29(08): 91-97.

[13] Xu Cheng, Guo Jinyang, Li Chao, et al. Developing heterogeneous FPGA acceleration systems using HLS: Problems, optimization methods, and opportunities. Computer Science and Exploration, 2019, 17(08): 1729-1748.

[14] Zhang Ruihao, Li Xiaoqi, Dang Lizhi, et al. Noise removal algorithm of image sensor FPN based on Vivado HLS. Application of single Chip Microcomputer and Embedded system, 2019, 23(12): 55-58+62.

[15] Mao Xin, Yan libing, Song Jianbo. Image registration algorithm based on RANSAC and Mutual information and its hardware implementation. Electro-optics & Control, 2022, 29(06): 72-76.
[16] Jiang Jie, Ling Sirui. A voting parallel RANSAC algorithm and its FPGA implementation. Journal of Electronics and Information Technology, 2014, 36(05): 1145-1150.
[17] Vourvoulakis John, John Kalomiros and John Lygouras. FPGA accelerator for real-time SIFT matching with RANSAC support. Microprocessors and Microsystems, 2016, 49:105-116.
[18] Vourvoulakis John, John Kalomiros and John Lygouras. FPGA-based architecture of a real-time SIFT matcher and RANSAC algorithm for robotic vision applications. Multimedia Tools and Applications, 2018, 77: 9393-9415.