The Twenty-Seventh International Symposium on Artificial Life and Robotics 2022 (AROB 27th 2022),
The Seventh International Symposium on BioComplexity 2022 (ISBC 7th 2022),
The Fifth International Symposium on Swarm Behavior and Bio-Inspired Robotics 2022 (SWARM 5th 2022)
ONLINE, January 25-27, 2022

Efficiency and improvement of parallel calculation structure in Field Programmable Gate Array

Jieyuan He[1]†, Shiyu Wang[2], Takahiro Nitta[3], Yuichiro Toda[4], and Mamoru Minami[5]

[1]Okayama University, Japan
(Tel: 81-86-251-8233, E-mail: p0xd499l@s.okayama-u.ac.jp)
[2]Okayama University, Japan
(Tel: 81-86-251-8233; E-mail: paw6968v@s.okayama-u.ac.jp)

Abstract: Nowadays, robots that can explore and investigate in dangerous and unknown environments (such as the deep sea) have attracted extensive attention. When a robot works in a changing environment, it must respond to the changes of the environment in real-time. Therefore, the visual servoing system is widely used in various scenes. However, in the past, robots equipped with visual servoing systems often cannot achieve accurate recognition in complex environments such as turbid seawater. Therefore, we developed a visual servoing system using a luminous 3D marker to solve the above problems. However, in previous research, the system has always relied on CPU to realize computing, which not only brings a great burden to CPU but also the calculation speed of general CPU is not satisfactory. To realize high-speed visual servo, we transplant the evaluation function which consumes a lot of calculation resources to the FPGA (Field Programmable Gate Array) module. Different from CPU, parallel calculation using FPGA only needs to be realized through the circuit designed by the designer, to improve the efficiency. In the past, FPGA design was developed by hardware designers, but now the circuit can also be designed in C through High-Level Synthesis (HLS) tools. Now, we have successfully realized the fast calculation of the evaluation function and output the correct results. In this paper, I will explain how to quickly calculate the evaluation function using FPGA. The development system adopts the Zynq UltraScale + MPSoC ZCU104 evaluation board provided by Xilinx. The evaluation board is equipped with an ARM CPU and programmable logic units.

Keywords: FPGA Implementation, Visual servoing, Model-Based Method, Targets Recognition

1. INTRODUCTION

Japan has a long coastline and vast sea area. In addition to the human shortage caused by aging, the demand for intelligent robots that can replace people to perform work in complex and dangerous marine environments has become very prominent, such as automatic underwater vehicles (AUVs). Underwater vehicles are used in the fields of subsea exploration, pipeline maintenance, and oil field exploration. Due to the limited battery capacity, underwater vehicles usually have limited working time and travel distance. They require recharging, maintenance support, and data transfer. It is essential to provide an underwater battery charging system for the AUV return dock station. The real underwater environment is very complex. During the docking process between the underwater robot and the charging dock, it is difficult to observe the correct position and posture of the charging interface due to the influence of water flow disturbance, turbidity, and refraction effect. To solve this problem, we designed a 3D marker that can be mounted on a charging dock. The 3D marker can be illuminated. Therefore, even in turbid underwater environments, the AUV can recognize the 3D marker and accurately dock with the docking station.

In this recognition system, the recognition of 3D marker by AUVs is realized by two CCD cameras. These two cameras are similar to human eyes. By comparing the scene images obtained by the two cameras, the 3D spatial information of the target object is obtained. In our previous research, the CPU will get the hue value according to the image, and then calculate the fitness through the fitness function. This imposes a huge computational burden on the CPU and consumes a lot of power. Moreover, the computing system usually equipped with a CPU is not portable enough. Therefore, we try to transplant the part with a large computational load in the recognition system into FPGA (field programmable gate array). FPGA is a programmable integrated circuit that can be configured by designers. FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects allowing blocks to be wired together. Logic blocks can be configured to perform complex combinational functions or act as simple logic gates like AND and XOR. Now we have transplanted the fitness function with the largest computing load to FPGA and obtained the same accuracy as the CPU computing system.

† Jieyuan He is the presenter of this paper.

2. BASIC PRINCIPLE OF 3D POSE ESTIMATION

2.1. Model-based Matching method

The system adopts dual-eye cameras to realize the perception of 3D space. The model-based matching method is used to estimate the matching degree between the projection model and the captured image. Fig. 1 shows the principle of the model-based matching method. Σ_h is the reference coordinate system of the right camera image and the left camera image. The 2D model of the real target object in space is projected naturally to the dual-eye cameras images and the dotted 3D marker model where the pose is given by the gene of Genetic Algorithms (GA) is projected from 3D-to2D [1]. The different relative pose is calculated by comparing the projected model and the captured images by the dual eye cameras. Finally, the best model of the target object that represents the true pose can be obtained based on its highest fitness value. The recognition is performed in 3D but evaluation is done in 2D.

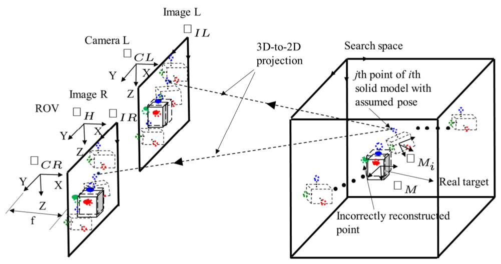


Fig. 1. Principle of Model-Based Method

2.2. 3D marker model

Fig. 2 shows the structure of the real target object. The target object has four small balls. The colors of each ball are blue, green, red and yellow. Fig. 3 shows the construction of the 3D marker model. Almost the same as the real target object, the 3D marker model is also composed of a blue, green, red, and yellow ball. Each ball is composed of points which divided into inner portion and outer portion. The points of inner portion and outer portion have two layers and are evenly distributed every 30 degrees around the center of the sphere. There are 192 points per model. The size of the inner portion is the same as that of the real target object. These points are used to calculate the fitness. When the fitness is the highest, the inner portion of the model can be considered to represent the real target object, and the outer portion represents the background of the real target object. The points in each portion mean points to calculate the correlation degree on how much the inner portion overlaps the target object and the outer portion does not overlap the target object [2].

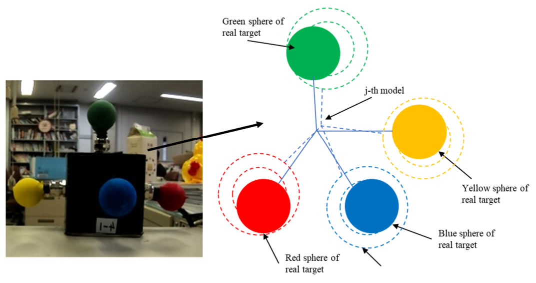

Green sphere of real target
j-th model
Yellow sphere of real target
Blue sphere of real target
Red sphere of real target
1-4

Fig. 2. 3D marker made of 4-spheres with red, green, blue and yellow colors

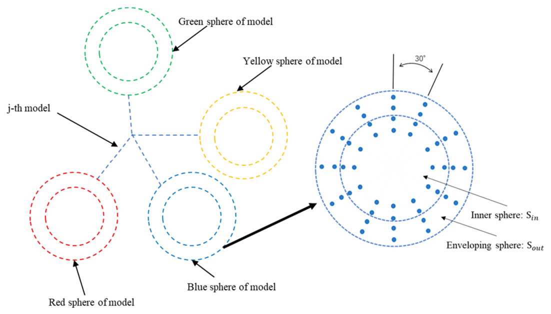


Fig. 3. Model of 3D marker

2.3. Fitness function

The fitness function is designed to evaluate the matching degree between model and object. The pose of the model is randomly generated by GA and then projected onto the 2D image taken by the camera. Finally, the fitness function is used to calculate the fitness of all models. The fitness will converge to a peak with the passage of time. At this time, the inner portion of the model will almost coincide with the true target object. The pose of this model can be regarded as the pose of the true target object.

The fitness of the 3D marker model is calculated by hue. The fitness function of the left camera used in the proposed method is defined as the Eq. (1) $\sim$ Eq. (3). Moreover, the fitness function of the right camera is consistent with that of the left camera. In Eq. (1) , N represents the total number of points of the inner and outer portion. ${}^{I}r_j\left(\phi_i\right) \in S_{in}$ represents the point in the inner portion and ${}^{I}r_j\left(\phi_i\right) \in S_{out}$ represents the point in the outer portion. ϕ_i defines the position and posture of the model as $\phi_i = [x, y, z, \varepsilon_1, \varepsilon_2, \varepsilon_3]$. j means the j-th model given by GA, and I represents the I-th point of the model. Therefore, ${}^{I}r_{j,in}\left(\phi_i\right)$ represents the I-th point of the inner portion on the j-th model generated by GA, and ${}^{I}r_{j,out}\left(\phi_i\right)$ represents the I-th point of the outer portion on the j-th model generated by GA. As shown in Eq. (2) and Eq. (3), when the ball of the 3D marker situated in the corresponding inner portions (red, green, blue, and yellow) of 2D model, $P_{H,1}\left({}^{I}r_{j,in}\left(\phi_i\right)\right) = +1$, otherwise $P_{H,1}\left({}^{I}r_{j,in}\left(\phi_i\right)\right) = -1$. When the 4 balls situated in corresponding inner portions at the same time, $P_{H,2}\left({}^{I}r_{j,out}\left(\phi_i\right)\right) = -1$, otherwise $P_{H,2}\left({}^{I}r_{j,out}\left(\phi_i\right)\right) = +1$. We use hue value to judge whether the 4 balls of 3D marker overlapped in the 2D model, and the evaluation range of balls of each color is shown in the Table. 1. Finally, the fitness for evaluation is the average of the fitness of the model located on the left and right cameras, as shown in Eq. (4).

$$
{}^{L}F_H = \frac{1}{N}\left(\sum_{{}^{I}r_j(\phi_i)\in S_{in}} P_{H,1}\left({}^{I}r_{j,in}(\phi_i)\right) + \sum_{{}^{I}r_j(\phi_i)\in S_{out}} P_{H,2}\left({}^{I}r_{j,out}(\phi_i)\right)\right) \tag{1}
$$

$$
P_{H,1}\left({}^{I}r_{j,in}(\phi_i)\right) = \begin{cases} +1, & if\ H_r\left({}^{I}r_{j,in}(\phi_i)\right) \in [H_{r,min}, H_{r,max}] \\ -1, & otherwise \end{cases} \tag{2}
$$

$$
P_{H,2}\left({}^{I}r_{j,out}(\phi_i)\right) = \begin{cases} -1, & if\ H_r\left({}^{I}r_{j,out}(\phi_i)\right) \in [H_{r,min}, H_{r,max}] \\ +1, & otherwise \end{cases} \tag{3}
$$

Table 1. Hue ranges of left and right cameras that are detected and calculated as red, green, blue and yellow in fitness function

		Red	Green	Blue	Yellow
Right Camera	$H_{r,min}$	150	320	0	245
	$H_{r,max}$	195	350	70	300
Left Camera	$H_{r,min}$	150	320	0	245
	$H_{r,max}$	195	350	70	300

$$
F_H = \frac{{}^{L}F_H + {}^{R}F_H}{2} \tag{4}
$$

3. DEVELOPMENT ENVIRONMENT

3.1. Development hardware

In order to realize high-speed visual servoing, we are transplanting the part which consumes a lot of calculation resources into FPGA. By using the integrated circuit designed by the designer to perform parallel processing, the efficiency can be improved. The current development hardware is the zynq ultrascale + MPSoC zcu104 Evaluation Kit developed by Xilinx. It is a combination of FPGA chip that is called PL(Programmable Logic), and CPU unit of arm architecture that is called PS(Processing System). Data is transferred between them through DMA. In addition, both CPU units and FPGA can perform calculation processing, but in order to maximize efficiency, it is best to let the FPGA side perform as many calculations as possible. The dual-cameras are connected to the board through USB to obtain images. Development computer is used to design circuits and communicate with the board through USB.

3.2. Software environment preparation

In order to carry out development, the preparation of several software environments is indispensable. Ubuntu Linux 16.04 LTS(64 bit) is installed. The Vitis unified software platform enables the development of FPGAs. The Vivado HLS Design Suite supply design teams with the tools and methodology needed to leverage C-based design. The PetaLinux Tools offers everything necessary to customize, build and deploy Embedded Linux solutions on Xilinx processing systems. The circuit design and configuration files can be compiled and generated by the development computer and written into the evaluation board through SD card[3].

4. SCHEME OF FITNESS CALCULATION WITH FPGA IMPLEMENTATION

In the past, all calculation processing of the model-based matching method recognition system was completed by the CPU. This places a huge load on the CPU. Therefore, in an ideal state, programmable logic will undertake most of the computing processing, and the CPU, that is, the processing system, only needs to undertake the task of reading images or some initial tasks. In addition, the performance of the processing system on the Evaluation Kit is not enough for a large number of calculations. Therefore, it is necessary to optimize the algorithm of programmable logic in both the GA algorithm and fitness evaluation. At this stage, we have completed the transplantation of the fitness function which consumes a lot of calculation resources.

4.1. OpenCL

Fig. 4 shows the collaboration principle of processing system and programmable logic at this stage. The communication between them utilizes OpenCL tools. OpenCl (Open Computing Language) is a framework for writing programs that execute across heterogeneous platforms consisting of CPUs, FPGAs and other processors or hardware accelerators. Programmable logic undertakes the calculation part of fitness evaluation. The fitness evaluation calculation can be said to be the part that consumes the most computing resources in the whole recognition system.

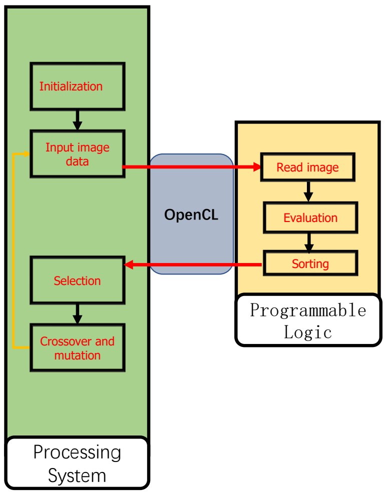


Fig. 4. Present recognition system

4.2. HLS Stream Library

Streaming data is a type of data transfer in which data samples are sent in sequential order starting from the first sample. Streaming requires no address management. Therefore, data can be read and written at high speed through streaming. Vivado HLS provides a C++ template class $hls :: stream <>$ for modeling streaming data structures. The streams implemented with the $hls :: stream <>$ class have the following attributes. In the C code, an $hls :: stream <>$ behaves like a FIFO of infinite depth. There is no requirement to define the size of an $hls :: stream <>$. They are read from and written to sequentially. That is, after data is read from an $hls :: stream <>$, it cannot be read again. Also, an array such as the pixels of a image, can use streaming. When an image is transferred from the processing system to programmable logic in the form of an array, the image can be specified as streaming. Then, the image pixels will be read by the $hls :: stream <>$ data type in the method of Fig. 5. The first pixel of the first row is read first, followed by the second pixel of the first row, and read down in order. When all pixels in the first row are read, go to the second row and start reading from the beginning. Follow the above rule until the last pixel is read. The last pixel to be read is the 640×480 pixel. By keeping the data streaming, the data can be read without an address[4]. Fig. 6 shows how to specify the image array and data as streaming. The mat variable is defined as $xf :: mat$ array format, which stores the pixel information of the whole image. Fig. 7 shows the time spent with and without streaming when calculating the fitness of 40 models. It can be seen that it only takes 4.34 ms with streaming, which is nearly 700 times faster than without streaming.

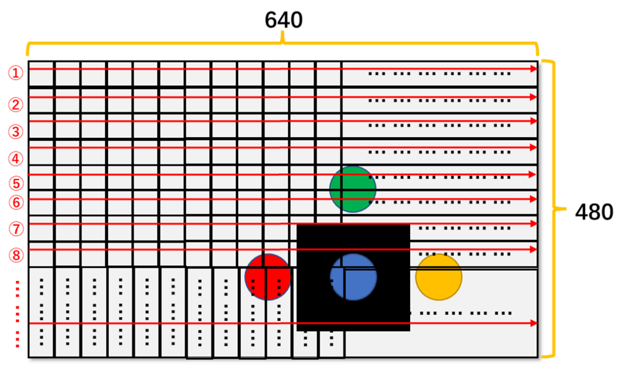
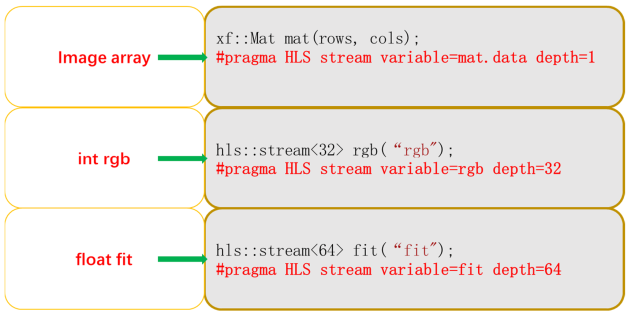


Fig. 5. The reading method when the image is specified as streaming

4.3. Pipeline processing

In the process of reading pixels and calculating fitness, a large number of loop calculations will be carried out. In a general CPU, if one 640×480 pixel image is read, it also takes at least 640×480 cycles of latency. With other processing, the latency will increase exponentially. In this case,



```
xf::Mat mat(rows, cols);
#pragma HLS stream variable=mat.data depth=1
```

```
hls::stream<32> rgb("rgb");
#pragma HLS stream variable=rgb depth=32
```

```
hls::stream<64> fit("fit");
#pragma HLS stream variable=fit depth=64
```

Fig. 6. The method of specifying the image array and data as streaming

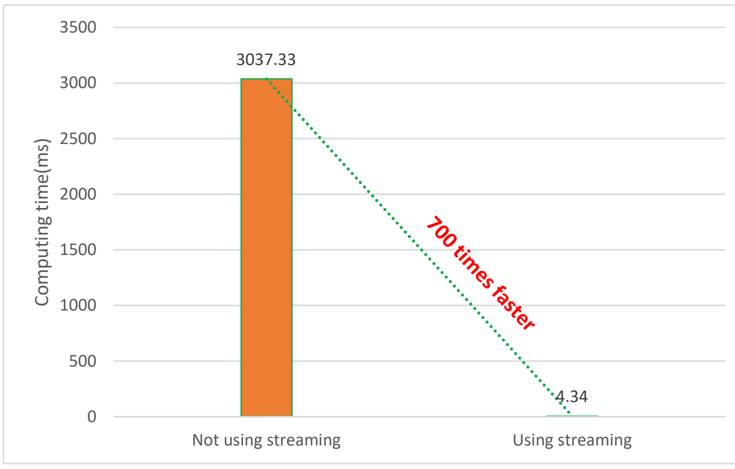
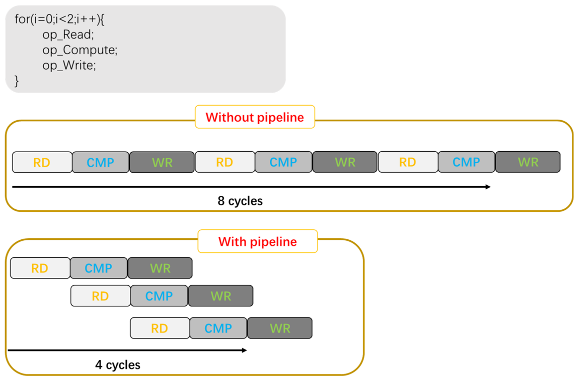


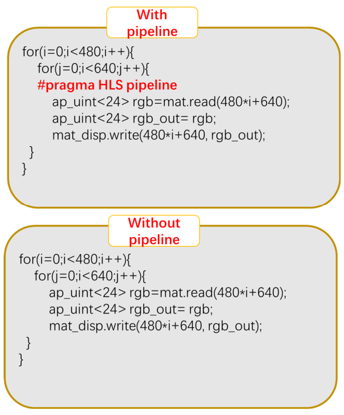
Fig. 7. Comparison of calculation speed with and without streaming

the loop calculation can be processed by the pipeline. The PIPELINE processing reduces the initiation interval for a function or loop by allowing the concurrent execution of operations. A pipelined loop can process new inputs every N clock cycles, where N is the initiation interval of the loop. The default initiation interval for the PIPELINE is 1, which processes a new input every clock cycle. Pipelining a loop allows the operations of the loop to be implemented in a concurrent manner as shown in the Fig. 8. The loop needs 3 clock cycles between each input read. The whole loop takes only 4 cycle clocks with Pipeline. It also shows the default sequential operation requires 8 clock cycles before the last output write is performed[5]. In order to verify the effect of the pipeline, as shown in the Fig. 9, enable pipeline processing and disable pipeline processing respectively under the cycle of reading and writing pictures. Then compare the time spent in the two cases. As shown in the Fig. 10, when reading a 640 × 480-pixel image, it takes a total of 2.02 ms in the case of enabling pipeline. In the case of disabling pipeline, it takes 4.34 ms. When reading a 1280 × 960- pixel image, it takes a total of 8.42 ms in the case of enabling pipeline. In the case of disabling pipeline, it takes 16.06 ms. When reading a 1920 × 1080-pixel image, it takes a total of 13.27 ms in the case of enabling pipeline. In the case of disabling the pipeline, it takes 27.75 ms.



```
for(i=0;i<2;i++){
      op_Read;
      op_Compute;
      op_Write;
}
```

Fig. 8. Pipeline processing



```
for(i=0;i<480;i++){
   for(j=0;j<640;j++){
   #pragma HLS pipeline
      ap_uint<24> rgb=mat.read(480*i+640);
      ap_uint<24> rgb_out= rgb;
      mat_disp.write(480*i+640, rgb_out);
   }
}
```

```
for(i=0;i<480;i++){
   for(j=0;j<640;j++){
      ap_uint<24> rgb=mat.read(480*i+640);
      ap_uint<24> rgb_out= rgb;
      mat_disp.write(480*i+640, rgb_out);
   }
}
```

Fig. 9. Setting position of pipeline

4.4. Optimization of pixel reading for FPGA characteristics

Although streaming data and pipeline can greatly improve the throughput efficiency of data and realize parallel computing of loop, at least 640 × 480 cycles of latency are still required because all pixels of the whole image needs to be read when processing a 640 × 480-pixel picture. However, it makes no sense to read pixels other than the model generated by GA. If only the pixels of the points on the model are read and the fitness is calculated, the cycle of latency can be greatly reduced. Fig. 11 shows the optimization principle of transferring RGB values from the processing system to programmable logic. The processing system first reads the RGB values of 192 evaluation points on the first model generated by GA, and then successively reads the RGB values of other models until the 40th model. Then write the RGB values of all the models into a new image array (using OpenCV's cv::mat class). Finally, transfer the new image to programmable logic. In this way, the cycle of latency can be reduced to 1 / 40 of that of reading the whole 640 × 480-pixel image. Table. 2 shows the time spent calculating different numbers of models before and after optimization. In the case of calculating 40 models, it takes only 0.11 microseconds, nearly 40 times faster than before optimization. And although the number of models increases, this advantage will be expanded.

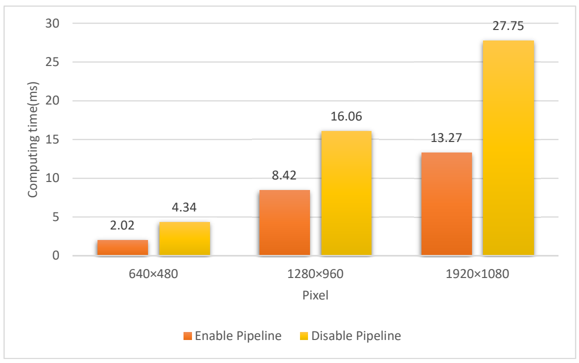


Fig. 10. Comparison of calculation time spent on enabling pipeline and disabling pipeline

Table 2. Comparison of computing time before and after optimizing the way of reading images

Number of genes	Before optimizing(ms)	After optimizing(ms)
40	4.25	0.11
80	8.08	0.16
120	13.15	0.22

5. RESULTS AND EFFICIENCY VERIFICATION

5.1. Correctness verification of calculation results

Because the calculation logic of FPGA is different from that of CPU, it is very important to verify the calculation results output by FPGA in the process of FPGA development. However, because the transplantation of the GA computing part has not been completed, it is difficult for FPGA to carry out real-time recognition directly. Therefore, we di-

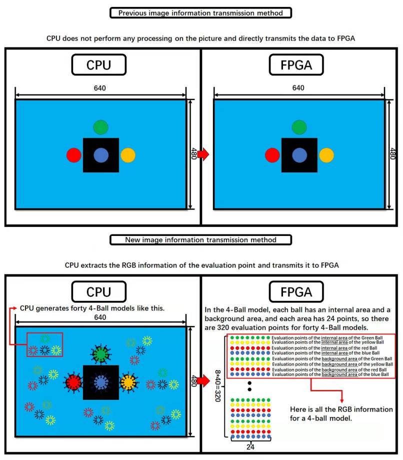

Previous image information transmission method
CPU does not perform any processing on the picture and directly transmits the data to FPGA
CPU
FPGA
640
480
New image information transmission method
CPU extracts the RGB information of the evaluation point and transmits it to FPGA
CPU generates forty 4-Ball models like this.
In the 4-Ball model, each ball has an internal area and a background area, and each area has 24 points, so there are 320 evaluation points for forty 4-Ball models.
Evaluation points of the internal area of the Green Ball
Evaluation points of the internal area of the yellow Ball
Evaluation points of the internal area of the red Ball
Evaluation points of the internal area of the blue Ball
Evaluation points of the background area of the Green Ball
Evaluation points of the background area of the yellow Ball
Evaluation points of the background area of the red Ball
Evaluation points of the background area of the blue Ball
8*40=320
Here is all the RGB information for a 4-ball model.
24

Fig. 11. Optimization of pixel reading

rectly give the positions and postures of 10 models, then let the CPU and FPGA calculate the fitness of all 10 models in Fig. 12 respectively, and finally compare them. The whole recognition system is developed based on the X86 CPU, so the fitness result calculated by the CPU can be regarded as true value. By comparing the values in Table. 3, it can be seen that the fitness of 10 models calculated by FPGA, that is, genes, is completely consistent with the results obtained by CPU. Therefore, the fitness calculation results developed by FPGA are verified.

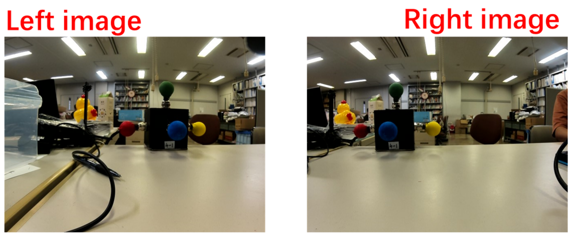


Fig. 12. Image of target

5.2. Efficiency of Fitness Calculation with FPGA Implementation

On the basis of ensuring that FPGA can calculate the fitness function correctly, it is very important to evaluate the computational efficiency of FPGA. As mentioned in the previous subsection, although it is difficult to directly recognize

Table 3. Correctness verification of results calculated by FPGA and CPU

Positions and postures of gene	Result of CPU	Result of FPGA
(-23.44,-19.92,710.60,-0.0031,-0.0627,0.0536)	0.29	0.29
(-16.80,-34.28,695.46,0.0536,-0.0124,-0.0153)	0.32	0.32
(-105.27,-78.03,782.86,0.0218,-0.0076,0.0776)	0.19	0.19
(-18.16,-30.57,702.54,0.0485,-0.0559,0.0025)	0.36	0.36
(-25.20,-9.08,712.30,-0.0925,-0.0848,0.0964)	0.27	0.27
(-23.44,-26.46,717.43,0.0107,-0.0753,0.0231)	0.16	0.16
(299.80,-0.39,857.81,-0.0372,0.1989,0.0605)	0.14	0.14
(-100.78,-63.38,814.60,-0.0439,0.1597,0.0234)	0.12	0.12
(10.55,-5.08,619.78,-0.0787,-0.0159,0.0982)	0.24	0.24
(-94.92,-75.29,863.92,-0.0049,0.1395,0.0484)	0.16	0.16

in real-time at this stage, the time spent in FPGA calculation can be timed by using the timer provided by OpenCL. Table. 4 shows the time taken by FPGA when calculating different numbers of genes. It should be noted that when calculating 160 genes, the utilization rate of LUT exceeds 100% at this time. It can be seen that using FPGA to calculate the fitness is very efficient.

Table 4. Efficiency of Fitness Calculation with FPGA Implementation

Number of genes	Latency(cycles)	Computing time(ms)	Utilization rate of LUTs(%)
40	7768	0.11	32%
80	15488	0.16	57%
120	23208	0.22	82%
140	27068	0.24	95%
160	30928	0.27	107%

6. CONCLUSION

In this paper, the fitness function calculation based on model-based matching method is successfully transplanted to FPGA calculation module. The calculation results and efficiency are verified. In the future, we will focus on transplanting the GA algorithm to FPGA, so as to build and complete the whole high-speed recognition system based on FPGA.

REFERENCES

[1] Lwin K N, Myint M, Mukada N, Yamada D, Matsuno T, Saitou K, Godou W, Sakamoto T, Minami M, “ Sea Docking by Dual-eye Pose Estimation with Optimized Genetic Algorithm Parameters, ” *Journal of Intelligent Robotic Systems*, Vol.92, Issue 1, pp.159-186, DOI (identifier) 10.1007/s10846-018-0970-x, 2018.
[2] Siyu Pan, Renya Takahashi, Jincheng Li, Yuichiro Toda, Mamoru Minami, ”Expanding the recognition distance using the Model-based Matching method and the 2D model by zoom cameras”, *26th International Symposium on Artificial Life and Robotics*, Online , pp.711-716, January 21-23, 2021.
[3] Shiqian Luo, Yuichiro Toda, Mamoru Minami, ”Fitness Calculation with a FPGA Implementation”, *26th International Symposium on Artificial Life and Robotics*, Online , pp.705-710, January 21-23, 2021.
[4] Vivado High-Level Synthesis tutorial (UG871).
[5] Vivado Design Suite User Guide: High-Level Synthesis (UG902).