Feasibility Analysis of Ultra High Frame Rate Visual Servoing on FPGA and SIMD Processor

Yifan He, Zhenyu Ye, Dongrui She, Bart Mesman, and Henk Corporaal

Eindhoven University of Technology, the Netherlands
`{y.he, z.ye, d.she, b.mesman, h.corporaal}@tue.nl`

Abstract. Visual servoing has been proven to obtain better performance than mechanical encoders for position acquisition. However, the often computationally intensive vision algorithms and the ever growing demands for higher frame rate make its realization very challenging. This work performs a case study on a typical industrial application, *organic light emitting diode (OLED)* screen printing, and demonstrates the feasibility of achieving ultra high frame rate visual servoing applications on both *field programmable gate array (FPGA)* and *single instruction multiple data (SIMD)* processors. We optimize the existing vision processing algorithm and propose a scalable *FPGA* implementation, which processes a frame within 102 μs. Though a dedicated *FPGA* implementation is extremely efficient, lack of flexibility and considerable amount of implementation time are two of its clear drawbacks. As an alternative, we propose a reconfigurable wide *SIMD* processor, which balances among efficiency, flexibility, and implementation effort. For input frames of 120×45 resolution, our *SIMD* can process a frame within 232 μs, sufficient to provide a throughput of $1000 fps$ with less than $1 ms$ latency for the whole vision servoing system. Compared to the reference realization on *MicroBlaze*, the proposed *SIMD* processor achieves a $21\times$ performance improvement.

Keywords: Visual Servoing, *FPGA*, Reconfiguration, Wide *SIMD*

1 Introduction

Visual servoing applies image sensors instead of mechanical encoders for position acquisition. On one hand, it reduces the number and accuracy requirement of encoders, and has been proven to obtain better performance than encoders in several applications, e.g., inkjet printing [2, 9]. On the other hand, it also dramatically increases the computing workload due to the often computationally intensive vision algorithms. The ever growing demand for higher frame rate makes the realization of visual servoing systems even more challenging [3, 4].

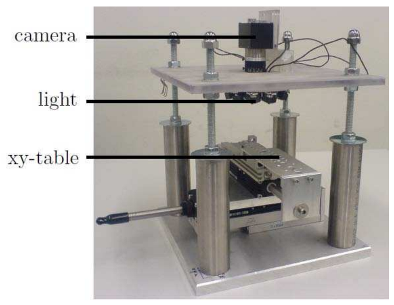
To address the issue of limited computing power, special purpose hardware is often used. Among them, the *field programmable gate array (FPGA)* is one of the most cost effective options. Though a dedicated *FPGA* implementation is usually of extreme efficiency, lack of flexibility and considerable amount of implementation effort are two of its clear drawbacks. On the other hand, wide *single instruction multiple data (SIMD)* processors are very popular among the stream processors for vision/image processing [1, 6, 7], because (*i*) the massive number of processing elements (*PEs*) inside an *SIMD* processor potentially renders very high throughput; (*ii*) massive parallelism in streaming applications typically shows up as data-level parallelism (*DLP*) which is naturally supported by *SIMD* architectures; and (*iii*) *SIMD* is a low-power architecture as it applies the same instructions to all *PEs*. The low-power feature is crucial to an embedded system. For a non-embedded system, this feature is sill very important as the design of the heat removal system can be greatly simplified. All these merits make *SIMD* architecture a very interesting candidate for visual servoing.

In order to demonstrate the feasibility of achieving ultra high frame rate visual servoing on both *FPGA* and *SIMD* processor, a typical industrial application, *organic light emitting diode (OLED)* screen printing, is analyzed in detail. Firstly, we improve the existing *OLED* center detection algorithm developed by Roel [9]. The proposed vision pipeline is not only more robust, but also more friendly to embedded processors and *FPGA/ASIC* realization. Only 32-bit fixed-point operations are used, while rendering sub-pixel accuracy. Moreover, the processing time is deterministic, which is crucial for latency oriented applications. After developing the visual pipeline, we propose an *FPGA* implementation with a processing time of only 102 μs on input image size of 120×45. However, realizing a dedicated visual servoing application implementation in *FPGA* requires considerable amount of effort, and a tiny change in the algorithm may cause re-design of the whole circuit. To balance among efficiency, flexibility, and implementation effort, we also propose a highly-efficient *SIMD* architecture for vision servoing applications, which is based on our previous design [6]. The number of *PEs* in this proposed *SIMD* processor can be dynamically reconfigured to match the resolution of the input frame and/or the performance requirement of the application. For input frames of size 120×45, our *SIMD* processor can process a frame within 232 μs, sufficient to meet the throughput requirement of *1000 fps* with a latency of less than 1 ms for the whole vision servoing system. Compared to the reference realization on *MicroBlaze* [11], the proposed *SIMD* processor achieved a 21× performance improvement.

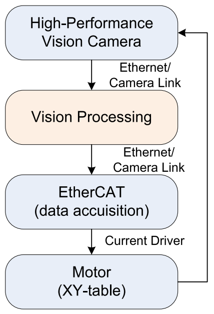
The remainder of this paper is organized as follows. In section 2, we show the visual servoing setup and our proposed vision pipeline. In Section 3, we elaborate the proposed *FPGA* implementation and performance analysis. The proposed reconfigurable wide *SIMD* processor architecture and mapping of the vision pipeline are presented in detail in Section 4. Finally, we draw the conclusions of this work in Section 5.

2 Visual Servoing System & Algorithm Development

The experimental visual servoing setup, which is described in [2], is shown in Fig. 1(a). The camera and lights are fixed at the top of the setup. The *OLED* structures are mounted on the moving *X*-*Y* table, which moves on a 2D plane.



(a) Visual Servoing Setup



(b) System Architecture

Fig. 1. The Experimental Setup and System Architecture

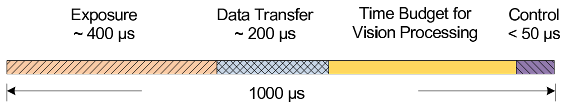


Fig. 2. System Delay Breakdown (*1000 fps*)

The system architecture is described in Fig. 1(b). At each frame interval, the camera takes an image of the moving *OLED* structures. The image is then transferred to the vision processing platform through *Ethernet* or *Camera Link* interface. In the vision processing step, which is the main focus of this paper, the vision processing platform processes the input image and localizes the centers of the *OLED* structures. The data-acquisition is realized by using *EtherCAT*, where *DAC*, *I/O*, and *ADC* modules are installed to drive the current amplifiers of the motors. Based on the relative positions of the detected *OLED* centers, X-Y table is then driven to a proper position.

In visual servoing systems, encoders are typically sampled at 1kHz [10]. To ensure the stability, we set the same sample rate (*1000 fps*) as the basic requirement for our visual servoing system. The timing breakdown of the complete visual servoing system is shown in Fig. 2, which consists of four components: (*i*) exposure of the image sensor; (*ii*) data readout from the image sensor; (*iii*) vision pipeline computing; and (*iv*) control algorithm. The required exposure time is measured on a real setup with the *OLED* structure. The exposure time depends on the lighting condition and the type of surface of the plate. It can vary from 10 μs for paper [2] to 400 μs for an *OLED* wafer [9]. The image read out time is measured on the *CameraLink* interface. The control algorithm takes a relatively small amount of time, which is common in industrial applications.

To reduce the delay of the system, only the *Region Of Interest (ROI)* of the image taken by the camera is read out and processed. A typical size of *ROI* for our *OLED* substrate localization application is 120×45 pixels or 160×55 pixels. The exposure time and image readout time is deterministic for a specific *ROI* size, lighting source, and camera interface. The timing of the control part is also deterministic given a specific mechanical setup. Therefore, the major source of reduction in the delay can only come from the vision processing component. In order to achieve $1000fps$ throughput as well as 1 ms latency, the timing budget remaining for vision processing is only about 350 μs.

In order to meet this tight budget, we firstly optimize the existing vision pipeline, which is based on a previous PC-based implementation [9]. This PC-based implementation is suboptimal for *FPGA/ASIC* and embedded processor realization. On one hand, the using of contour tracing algorithm in the PC-based vision pipeline has several drawbacks: (*i*) the processing time is not deterministic; (*ii*) less robust due to higher false detection rate, e.g., fail to detect the *OLEDs* with scratches on them in the bottom image of Fig. 3(a); and (*iii*) less efficient on parallel architectures. By utilizing the characteristics of the repetitive structures, this paper propose an erosion-projection method (Fig. 4) to replace contour tracing to solve the aforementioned issues. The input of the pipeline is a *region of interest (ROI)*, and the output of the pipeline are the coordinates of the centers of *OLED* structures. The input image is binarized with the optimum threshold calculated by the *OTSU* algorithm [8]. After binarization, noise and unrelated patterns are removed through the erosion step, remaining only the dominant structures (i.e., *OLEDs*). The number of erosion iterations is determined by the feature to be detected, the size of the unrelated patterns, and the quality of

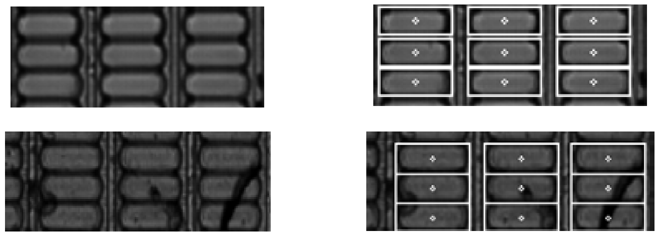

(a) *OLED* Structures (b) Detected Centers

Fig. 3. *OLED* Structures and Detected Centers

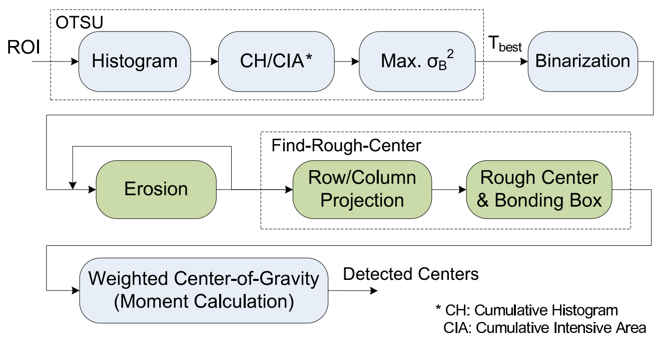

ROI
OTSU
Histogram
CH/CIA*
Max. σ_B^2
T_{best}
Binarization
Find-Rough-Center
Erosion
Row/Column Projection
Rough Center & Bonding Box
Weighted Center-of-Gravity (Moment Calculation)
Detected Centers
* CH: Cumulative Histogram
CIA: Cumulative Intensive Area

Fig. 4. Proposed Vision Pipeline for *OLED* Center Detection

the picture. In our case, two iterations are applied to a 120 $\times$ 45 input frame. Reduction of the segmented *OLED* structures into two vectors are performed by horizontal and vertical projection. The rough centers of the *OLED* structure are found by searching the two vectors reduced from projection. The accurate *OLED* centers are finally located by the weighted center-of-gravity inside each bonding box (Fig. 3(b)). Every stage of this new pipeline has a deterministic execution time, which leads to determinism of the complete vision pipeline. On the other hand, the floating point operations are usually too costly for embedded processor and dedicated hardware realization. Therefore, floating-point to fixed-point transformation is applied carefully. The new algorithm only uses 32-bit fixed-point operations, yet still renders sub-pixel accuracy. This new approach reduces the complexity of the algorithm, provides improved robustness, and is also more friendly to embedded processors and dedicated hardware realization.

3 Feasibility Analysis of FPGA Realization

Before implementing the vision pipeline on *FPGA*, we first analyze the sequential reference implementation on *MicroBlaze*. *MicroBlaze* is a simple soft-core with a *RISC* instruction set [11]. It is chosen because it represents the typical *general purpose processors (GPP)* in embedded systems. Also *MicroBlaze* is configurable and easy to verify on *FPGA*. Although its performance is relatively low, it can still be served as a good reference of a dedicated *FPGA* implementation. The *MicroBlaze* used here has the following configuration:

- 5-stage pipeline.
- 32-bit multiplier and 32-bit divider.
- All instructions and data are in local memory with one cycle latency.

For a 120 $\times$ 45 resolution input frame, the execution time of the *MicroBlaze* implementation is almost 5 *ms* at 125 MHz, which is over 14 times of the 350 μs

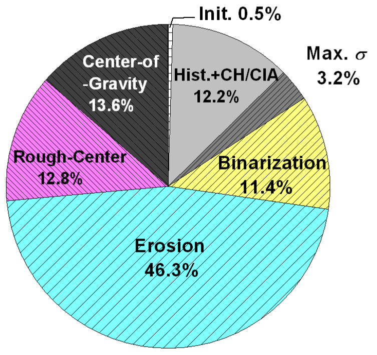


Fig. 5. Cycle Breakdown on *MicroBlaze* (4.92 *ms*/frame for image size of 120 $\times$ 45)

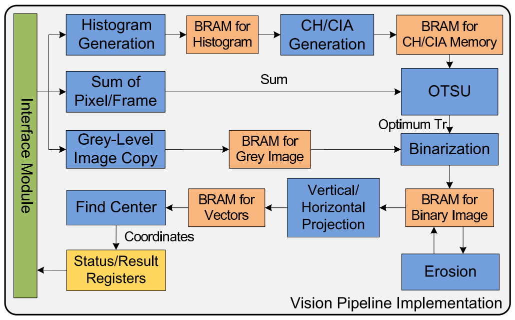


Fig. 6. Block Diagram of the vision pipeline implemented on *FPGA*

budget available for vision processing. Fig. 5 shows the cycle count breakdown of the vision pipeline.

To address the issue of limited computing power of the soft-core on *FPGA*, we propose a dedicated implementation (Fig. 6). Each stage of the vision pipeline is realized with dedicated modules, which runs as a synchronous systolic array. Frames is streamed through the pipeline, and intermediate values between each stage are buffered in *Block Random Access Memory (BRAM)*. Table 1 shows the detailed timing breakdown for image of size 120×45. This implementation can run at 160 MHz on *Virtex II-Pro FPGA (XC2VP30)*. The resource utilization is less than 26%. We can see that the proposed *FPGA* implementation achieves a speed-up of $48\times$ (comparing to the reference *MicroBlaze* implementation), resulting in an execution time of 102 μs, which is far below the 350 μs budget. Since our *FPGA* implementation is a parameterized design, it is easy to adapt to input images of different sizes too. Table 2 presents the performance at different resolutions, where w is the image width and h is the image height. It shows that the proposed design has very good scalability. We conclude that *FPGA* is a feasible choice to achieve ultra high frame rate visual servoing.

The vision pipeline can be further accelerated by utilizing more *FPGA* hardware resources. However, since the vision pipeline is no longer a bottleneck, further acceleration has a diminishing return in reducing the delay.

4 Feasibility Analysis of SIMD Realization

We have shown in Section 3 that a dedicated *FPGA* implementation is very suitable for ultra high frame rate visual servoing. However, lack of flexibility and considerable amount of implementation effort are two of its clear drawbacks. As an alternative, we exploit the feasibility of realizing ultra high frame rate visual servoing on *single instruction multiple data (SIMD)* processor.

Wide *SIMD* processors are very popular among the stream processors for vision/image processing [1, 6, 7]. The massive number of processing elements (*PEs*) in it potentially renders very high throughput. And massive parallelism in streaming applications typically shows up as data-level parallelism (*DLP*)

Table 1. Cycle Breakdown of *FPGA* Implementation (image size of 120 × 45)

Kernel	MicroBlaze (125 MHz)	Proposed FPGA (160 MHz)	Speed-up
Initialize	2819	256	11.01×
OTSU: Hist. & CH/CIA	74797	5659	13.22×
OTSU: Max. σ_B^2	19936	312	63.90×
Binarization	70201	5401	13.00×
Erosion	284819	97	2936×
Find-Rough-Center	78832	170	463.72×
Weighted Center of Gravity	83790	4501	18.62×
Total Cycles	615194	16396	37.5×
Time	4.92 ms	102 μs	48.2×

Table 2. Performance Scalability on the Proposed *FPGA* Implementation

Kernel	120 × 45	160 × 55	
Initialize	256	256	*O(1)*
OTSU: Hist. & CH/CIA	5659	9059	*O(wh)*
OTSU: Max. σ_B^2	312	312	*O(1)*
Binarization	5401	8801	*O(wh)*
Erosion	97	117	*O(h)*
Find-Rough-Center	170	220	*O(w+h)*
Weighted Center of Gravity	4501	7335	*O(wh)*
Total Cycles	16396	26100	
Time	102 μs	163 μs	

which is naturally supported by *SIMD* architectures. Moreover, *SIMD* is a low-power architecture which is crucial to embedded systems. All these merits make *SIMD* architecture a very interesting candidate for visual servoing.

If we look at the cycle breakdown on *MicroBlaze* (Fig. 5) in more detail, we will find that over 95% of the total execution time is spent on five kernels: histogram + *Cumulative Histogram* and *Cumulative Intensive Area* (*CH/CIA*), binarization, erosion, find-rough-center, and weighted center-of-gravity. The computation part of these kernels are mostly pixel-wise operations with few dependency (thus *DLP*), which makes them very suitable for *SIMD* processing.

4.1 Proposed Reconfigurable Wide SIMD Architecture

The proposed reconfigurable wide *SIMD* processor for visual servoing is based on *Xetal-Pro*, our ultra low-energy and high-throughput *SIMD* processor [6]. Fig. 7 presents its block diagram. The *control processor (CP)* is a 32-bit *MIPS-like* processor, equipped with a 32-bit 1-cycle multiplier and a 32-bit 16-cycle pipelined divider. The main task of the *CP* is to control the program flow, to handle interrupts, to configure other blocks, and to communicate with the outside world. The *processing elements (PEs)* and their corresponding *scratchpad memory (SM)* and *frame memory (FM)* banks are partitioned into tiles. Each tile consists of 8 *PEs*. This is based on the reconfiguration granularity requirement as well as the layout constraints. In the current implementation, there are

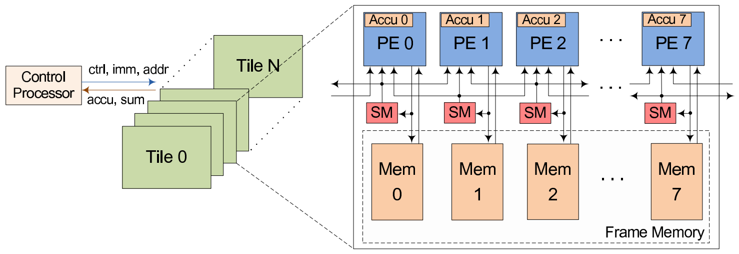


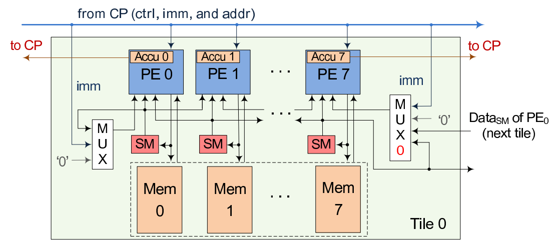
Fig. 7. Block Diagram of the Reconfigurable Wide *SIMD* Processor

320 *PEs* in total (40 tiles). The 128bit×1024 pseudo-dual port *SRAM* per *PE* constitutes the *frame memory (FM)*. The relatively large capacity of the *FM* allows on-chip storage of multiple *VGA* frames or images with higher resolution. The *SM* is a 32-entry scratchpad memory to exploit the often available data locality and to reduce the energy consumption of accessing the large *FM*. The communication network between *PEs* and *SMs* enables each *PE* to directly access data from its left and right neighbors. The whole system runs at 125MHz with 1.2*V* supply voltage under *TSMC* 65 *nm* technology, and offers a peak throughput of 80 *GOPS* (counting multiply and add operations only).

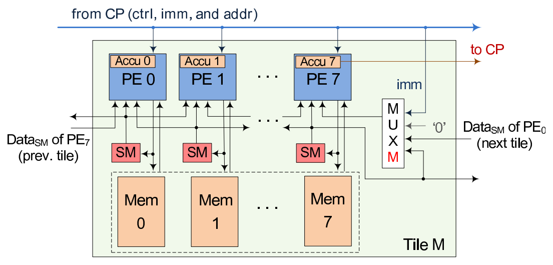
With 320 *PEs*, the proposed *SIMD* processor is able to provide a tremendous amount of processing power. However, in practice, applications may not require or cannot fully utilize its entire capability. For example, in our *OLED* substrate localization application, two typical image resolutions are 120×45 pixels and 160×55 pixels, which lead to natural vector sizes of 120 and 160 respectively. Thus, only 120 *PEs* (15 tiles) or 160 *PEs* (20 tiles) is required for each case. The processor must be configured in such a way that the number of active *PEs* and the corresponding communication network meet this requirement. Another important motivation for a reconfigurable *SIMD* processor is the power consumption. When not all *PEs* are required, the unused *PEs* can be fully shut down to save power consumption.

The number of active tiles of the proposed *SIMD* processor is dynamically configurable to meet various vector lengths or performance requirements. In order to enable this feature, two types of tiles are designed, which are only of slight difference. *Basic Tile* (Fig. 8(a)) composes the minimal system (*8-PE SIMD*) when MUX_0 is configured to choose only among immediate number (*imm*), constant '0', and data read from PE_7's own scratchpad memory. *Augmented Tiles* (Fig. 8(b)) can be enabled/disabled according to the application. To configure an *SIMD* processor with *M+1* tiles, $MUX_0 \sim MUX_{M-1}$ are fixed to their right neighbor (i.e., data from next PE_0*'s SM*) and MUX_M has the freedom to choose among the other three inputs except its right neighbor. The configuration is done through setting the control registers ($CTRL_0 \sim CTRL_M$) by *CP* at run time.

Each *PE* has a two-stage pipeline and shares the instruction fetch and decode stage of the *CP*. 16-bit *ADD/SUB*, *MUL*, *MAC*, and logical operations are supported. All instructions are executed in a single cycle. The global sum of the *ACCU* registers (one per *PE*) is calculated by an adder tree. The latency of the



(a)


from CP (ctrl, imm, and addr)
to CP
Accu 0
PE 0
Accu 1
PE 1
Accu 7
PE 7
imm
MUX M
'0'
Data$_{SM}$ of PE$_7$ (prev. tile)
Data$_{SM}$ of PE$_0$ (next tile)
SM
SM
SM
Mem 0
Mem 1
Mem 7
Tile M

(b)

Fig. 8. (a) Basic Tile; (b) Augmented Tile

adder tree is three cycles. Another main differences between this proposed *SIMD* and our previous *Xetal-Pro* is that local indirect addressing is supported with the local address generator. It has been shown that local indirect addressing can significantly improve the performance of some applications (e.g., Histogram, Hough Transform) in a massively-parallel *SIMD* processor [5].

4.2 Visual Servoing on the Proposed Wide SIMD Processor

Based on the analysis of the kernels in the vision pipeline, the parts that can benefit from vectorization are identified. With this information, the kernels of the vision pipeline are partitioned into (i) the vector part, which is executed on the *PE* array of the proposed *SIMD* processor; and (ii) the scalar part, which is processed on the *CP* of the proposed *SIMD* processor.

There are two main sources of inefficiency in the sequential implementation. Firstly, the fundamental limitation in operation throughput makes it impossible to achieve high performance. Secondly, the overheads such as loop control and address calculation greatly reduce the effective computational throughput. On the proposed architecture, the *PE* array provides a peak throughput of $2 \times num\_of\_PEs$ operations per cycle if *MAC* instruction is utilized. In addition, the concurrent execution of the *CP* and *PE* array exploits the *instruction-level parallelism (ILP)* and the overhead is reduced by overlapping the execution of control and computation operations.

The distribution of kernels between *PE* array and *CP* is shown in Fig. 9(a). With this mapping, the data communication between *CP* and *PE* array is minimized. To process an image of size $w \times h$, where w is the image width and h

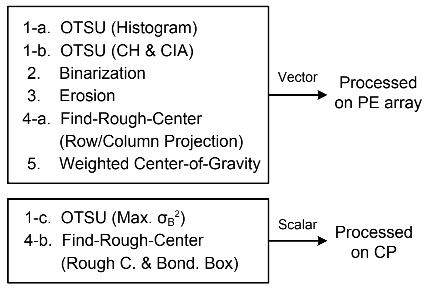
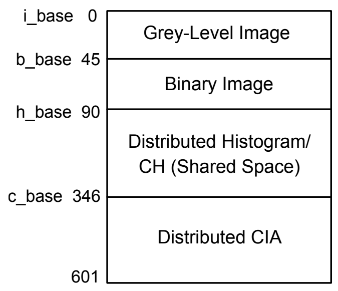

1-a. OTSU (Histogram)
1-b. OTSU (CH & CIA)
2. Binarization
3. Erosion
4-a. Find-Rough-Center (Row/Column Projection)
5. Weighted Center-of-Gravity
Vector
Processed on PE array
1-c. OTSU (Max. σ_B^2)
4-b. Find-Rough-Center (Rough C. & Bond. Box)
Scalar
Processed on CP
(a)


Fig. 9. (a) Kernel Assignment; (b) Frame Memory Mapping

Algorithm 1: Erosion

```
   Input  : $w \times h$ binary image at $i\_base$ in FM
   Output : $w \times h$ binary image at $i\_base$ in FM
1  $temp\_base \leftarrow$ address for a two-entry buffer in FM
2  on PE $0$ to $w-1$ do
3      for $i \leftarrow 1$ to $h-1$ do
4          CP: $temp \leftarrow temp\_base + (i \mod 2)$
5          $mem[i\_base + i - 2] \leftarrow mem[temp]$
6          $accu \leftarrow mem[i\_base + i]$
7          $accu \leftarrow accu + mem[i\_base + i - 1]$
8          $accu \leftarrow accu + mem[i\_base + i + 1]$
9          $accu \leftarrow accu + l\_mem[i\_base + i]$
10         $accu \leftarrow accu + r\_mem[i\_base + i]$
11         $flag \leftarrow accu < 5$
12         $mem[temp] \leftarrow flag ? 0 : 1$
13     end
14     $mem[i\_base + h - 2] \leftarrow mem[temp]$
15     CP: $temp \leftarrow 1 - temp$
16     $mem[i\_base + h - 3] \leftarrow mem[temp]$
17 end
```

is the image height, w *PEs* are required (thus $w/8$ tiles are enabled in our proposed *SIMD* processor), which indicates that the *PE* array can provide a peak throughput of $2w$ operations per cycle. When a frame is captured and ready for processing, the processor uses the input shift register to get the frame to the *frame memory (FM)* line by line. Each *PE* processes one column of the frame. The unused tiles are shut down to save power. Each *PE* needs $2h + 512$ *FM* entries to process a column (h entries for the space of the grey-level input image, h entries for the binary image, 256 entries for the shared space of the distributed histogram and *CH*, and 256 entries for the space of distributed *CIA*). The typical frame size in our case is 120×45 or 160×55, so the capacity of the *FM* is sufficient to process the whole frame. The memory mapping of the frame memory for a 120×45 resolution input is shown in Fig. 9(b).

Algorithm 1 (pseudo code) gives an example of how to program the proposed *SIMD* processor. The erosion step in Fig. 4 uses a cross kernel. To calculate one output pixel, a *PE* needs to get the four neighboring pixels, two of which are located in the neighborhood *PE*s' *FM*. These two pixels can be accessed using

Table 3. Cycle Breakdown of *SIMD* Implementation (image size of 120×45)

Kernel	MicroBlaze (125 MHz)	Proposed SIMD (125 MHz)	Speed-up
Initialize	2819	1280	$2.20\times$
OTSU: Hist. & CH/CIA	74797	4047	$18.5\times$
OTSU: Max. σ_B^2	19936	15840	$1.26\times$
Binarization	70201	225	$312\times$
Erosion	284819	1038	$274\times$
Find-Rough-Center	78832	4601	$17.1\times$
Weighted Center of Gravity	83790	1971	$42.5\times$
Total Cycles	615194	29002	$21.2\times$
Time	4.92 *ms*	232 μs	$21.2\times$

Table 4. Performance Scalability on the Proposed *SIMD* Implementation

Kernel	120×45	160×55	
Initialize	1280	1280	*O(1)*
OTSU: Hist. & CH/CIA	4047	4097	*O(h)*
OTSU: Max. σ_B^2	15840	15840	*O(1)*
Binarization	225	275	*O(h)*
Erosion	1038	1278	*O(h)*
Find-Rough-Center	4601	5770	*O(w+h)*
Weighted Center of Gravity	1971	2295	*O(h)*
Total Cycles	29002	30835	
Time	232 μs	247 μs	

the neighborhood communication in the proposed processor. For *OLED* center detection, erosion is called twice.

As indicated by Fig. 5, the erosion kernel is the most time consuming part in the *MicroBlaze* implementation. Table 3 shows that after vectorization, speed-up of $274\times$ is achieved for a 120×45 resolution input. And we can also see that it is no longer a bottleneck on the proposed *SIMD* processor implementation.

The performance of the complete implementation for a 120×45 resolution input is shown in Table 3. The wide *SIMD* implementation is able to achieve a speed-up of $21\times$ (comparing to the reference *MicroBlaze* implementation), resulting in an execution time of 232 μs, which is well below the 350 μs budget for vision processing. In contrast to Fig. 5, finding max σ_B^2 is now most time consuming, because it is sequential and can only be done on the *CP*. Table 4 shows the results of input images with different sizes. We can see that the wide *SIMD* implementation has even better scalability than the dedicated *FPGA* implementation. The result also shows that it is feasible to achieve > 1000 fps and < 1 *ms* latency visual servoing on the proposed wide *SIMD* processor.

5 Conclusions & Future Work

This work performed a detailed analysis of achieving ultra high frame rate visual servoing on both *FPGA* and *SIMD* processor. A typical industrial application, *organic light emitting diode (OLED)* screen printing, was chosen in our analysis. We optimized the existing vision pipeline for this application so that it is more robust and more friendly for hardware implementation. Through a proposed *FPGA* implementation, we shown that it is very efficient and feasible to achieve ultra high frame rate visual servoing on *FPGA*. However, a dedicated *FPGA* implementation is usually lack of flexibility, and requires considerable amount of implementation effort. As an alternative, we also explored the feasibility analysis on the popular *SIMD* processor. The result shows that our proposed *SIMD* processor is very suitable for ultra high frame rate visual servoing. It achieved a proper balance among efficiency, flexibility, and implementation effort. Compared to the reference realization on *MicroBlaze*, a 21× reduction on the processing time is gained, which greatly enables the performance improvement for visual servoing applications.

For the future work, we would like to measure and compare the energy consumption in detail. We would also like to enable the fault-tolerance features of our *SIMD* processor to deal with the increasingly severe manufacturing variability issue.

References

1. A. Abbo and et al. Xetal-II: a 107 GOPS, 600 mW massively parallel processor for video scene analysis. *IEEE Journal of Solid-State Circuits*, 43(1):192–201, 2008.
2. J. de Best and et al. Direct dynamic visual servoing at 1 khz by using the product as 1.5d encoder. In *ICCA 2009*, pages 361–366, dec. 2009.
3. N. Furukawa and et al. Dynamic regrasping using a high-speed multifingered hand and a high-speed vision system . In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 181–187, 2006.
4. R. Ginhoux and et al. Beating heart tracking in robotic surgery using 500 Hz visual servoing, model predictive control and an adaptive observer. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 274–279, 2004.
5. Y. He and et al. Real-Time Hough Transform on 1-D SIMD Processors: Implementation and Architecture Exploration. In *ACIVS*, pages 254–265, 2008.
6. Y. He and et al. Xetal-Pro: An Ultra-Low Energy and High Throughput SIMD Processor. In *Proceedings of the 47th Annual Design Automation Conference*, 2010.
7. S. Kyo and et al. IMAPCAR: A 100 GOPS In-Vehicle Vision Processor Based on 128 Ring Connected Four-Way VLIW Processing Elements. *Journal of Signal Processing Systems*, pages 1–12, 2008.
8. N. Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11:285–296, 1975.
9. R. Pieters and et al. Real-Time Center Detection of an OLED Structure. In *ACIVS*, pages 400–409. Springer, 2009.
10. R. Pieters and et al. High performance visual servoing for controlled m-positioning. In *WCICA*, pages 379–384, 2010.
11. Xilinx, Inc. `http://www.xilinx.com/tools/microblaze.htm`.