International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 7, Number 2, pages 372–386, July 2017

Deep-pipelined FPGA Implementation of Real-time Object Tracking using a Particle Filter

Theint Theint Thu, Yoshiki Hayashida, Akane Tahara, Yuichiro Shibata, Kiyoshi Oguri
Graduate School of Engineering, Nagasaki University
1-14 Bunkyo-machi, Nagasaki, 852-8521, Japan

Received: February 15, 2017
Revised: May 3, 2017
Accepted: May 31, 2017
Communicated by Michihiro Koibuchi

Abstract

This paper presents a real-time FPGA implementation of the posterior system state estimation in dynamic models, which is developed using particle filter algorithm. Specially, our system is constructed by parallel resampling (FO-resampling) algorithm on a stream-based architecture. To be precise, the resampling is accomplished in a valid pixel area of an input image frame while prediction and update of particles are performed in a synchronization region, thus our approach achieves realtime performance of 60 fps for VGA images, synchronized with the camera pixel throughput without using any external memory devices. Through evaluation with an object tracking benchmark video, the tradeoff relationship between tracking quality and the number of particles is analyzed to find an appropriate hardware parameters. In addition, we address improvement of resource utilization for our particle filter architecture, especially by using a higher clock frequency to reuse hardware resources in a time sharing manner. The implementation experiments reveal that the proposed approach allows the original design to be fitted in a smaller FPGA chip. However, we also demonstrate this size reduction approach has an overhead of 2.7 to 3.0 times power consumption compared to original designs with a slow clock frequency.

Keywords: particle filter, FPGA, stream-based architecture, parallel resampling

1 Introduction

Recent technological developments regarding the object tracking have led to many computer vision applications such as robotics, video surveillance, biomedical engineering, etc. According to the filtering problems in high-dimensional space, particle filters can trace the object well and enable a fully nonlinear and non-Gaussian analysis step whereas Kalman filters are untraceable. Moreover, the particle filter has become popular among the nonparametric filters in marker-less tracking applications. Study of real-time processing for camera image using stream processing on FPGAs is also popular to fulfill the demands for fast and robust object tracking with low energy consumption. Hence, implementation of the particle filter in stream processing is a promising approach for effective object tracking systems. On the other hand, the resampling operation is the bottleneck in real-time

[0]The preliminary version of this paper has been presented at the Fourth International Symposium on Computing and Networking (CANDAR 2016) [1].

particle filter implementation [2, 3]. Difficulty of parallel resampling comes from it needs information of all particles and thus computational dependency arises. To cope with this problem, the FPGA optimized resampling (FO-resampling) [4] has been proposed, where virtual particles are randomly generated around the real particle to avoid the elimination of real particles with small weights. Although effectiveness of FO-resampling method has been shown with mathematical models [4], attempts to combine the FO-resampling with stream processing have not been addressed.

With the deep-pipelined stream-oriented image processing architecture, we can achieve a real-time throughput at a relatively low clock frequency without requiring any external memories. We have already shown such an approach is of benefit for a variety of image applications in terms of performance and power consumption [5, 6, 7, 8]. In this paper, we propose and implement a stream-based architecture of a particle filter based on FO-resampling method. In addition, we compare four kinds of FPGA implementation using the different resource usages and clock frequencies for the purpose of achieving compact hardware architecture. The major contributions of this paper include:

- A deep pipelined stream architecture of a particle filter with FO-resampling is proposed.
- Tracking quality of a real-time object tracking system based on the proposed architecture is evaluated and compared with a conventional multinomial resampling approach.
- The performance and resource utilization of FPGA implementation of the proposed architecture are evaluated.
- Comparative discussion on implementation alternatives with improved area efficiency is presented in terms of the hardware amount and power consumption.

The rest of the paper is organized as follows. Section 2 surveys related work on the efficient implementation of particle filters. Section 3 outlines the algorithm of particle filter. The implementation of the system in detail with three steps: prediction, likelihood calculation and parallel resampling are described in Section 4. Section 5 shows the evaluation results and discussion on our first implementation. Section 6 discusses the improvement of the proposed architecture to fit the design in a smaller FPGA. Finally, we draw some conclusions and describe the scope of the future work in Section 7.

2 Related Work

Since the object tracking plays a part in image processing and computer vision systems, fast and robust real-time image processing of non-rigid objects becomes a performance bottleneck. For this reason, image-based features for particle filters using color histogram were introduced by [9]. Particle filtering [10] approximates the density directly as a finite number of samples whereas the extended Kalman filter cannot approximate the probability density without a Gaussian. The color-based particle filter enables an embedded implementation in [11]. However, the parallel implementation does not offer in their tracking system. Many parallel resampling methods have been applied to FPGA-based particle filter implementation. For example, the particles are resampled using Independent Metropolis-Hastings (IMH) resampling method and the root mean square error is used to measure the accuracy [12]. Although they could apply the parallel particle filter implementation successfully with the high speed and accurate estimation performance, they did not fully parallelize the particles due to lack of hardware resources.

Major constraint of parallel resampling is to get the simultaneous information of all particles. Hence, FO-resampling method overcomes this constraint on the particle filter [4]. In their implementation, they compared FO-resampling with the multinomial resampling using root mean square error. The FO-resampling scheme is similar to Gibbs sampling but it can execute the resampling step without the complete particle set. In our work, we develop a deep-pipelined object tracking system with stream-based image processing on an FPGA. Image data obtained by the camera device is streamed in the system pixel by pixel. The streamed processing approach achieves real-time object detection for input video frames without any external memory modules.

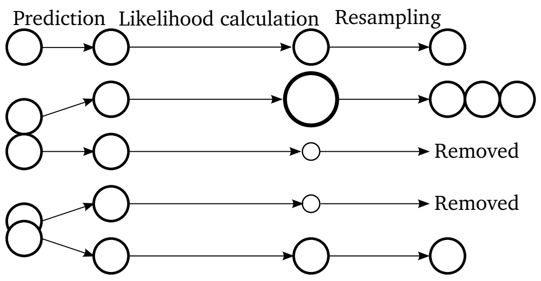

Figure 1: Overview of a particle filter

3 Particle Filter Algorithm

3.1 Particle Filter

Particle filter is a recursive filter that provides the estimation of non-linear and non-Gaussian processes. Each particle has application-specific states such as coordinates and velocities, and these states are randomly initialized at $t = 0$. Then, particle filter performs three steps for each time step as shown in Fig. 1. Prediction utilizes the previous observations to predict the state of a system in future time instants for each particle. Moreover, the noise is added to the states to adjust the irregular movement. For example, when each particle has a 2D coordinate (x, y) and velocities (v_x, v_y), and a uniform linear motion is assumed for a tracking target, this process will be expressed as follows:

$$x_t = x_{t-1} + v_{x_{t-1}} + n_x \tag{1}$$

$$y_t = y_{t-1} + v_{y_{t-1}} + n_y \tag{2}$$

$$v_{x_t} = v_{x_{t-1}} + n_{vx} \tag{3}$$

$$v_{y_t} = v_{y_{t-1}} + n_{vy} \tag{4}$$

where x_t and x_{t-1} represent the predicted state of object at time t and $t-1$. v denotes the velocity based on the position of a particle for each time state. According to the noises, n_x and n_y stand for the positioning noises and n_{vx} and n_{vy} are the velocity noises. The values of noises are randomly generated. In likelihood calculation step, weight for the sampled particle is computed as:

$$w_t^{[m]} = p(z_t | x_t^{[m]}) \tag{5}$$

where z_t is current sensor measurement and x_t shows the m-th particle ($1 \leq m \leq M$ and M is the total number of particles).

Resampling reduces the variance of particles to be loss of diversity and sample again from the particles until the total number of particles (M) are the same as the previous stage. During resampling step, some particles are replicated in proportion to their weights whereas some particles with small weights are eliminated to remedy for weight degeneration.

3.2 FPGA Optimized Resampling (FO-resampling)

The conventional multinomial resampling algorithm draws M particles according to the probability defined by the weights of each particle, so that small weight particles far from target are died out and big weight particles near the target are replicated. This process needs information of all particles, making parallel FPGA implementation difficult. However, FO-resampling, which is an approximated resampling method [4], can solve this problem using virtual particles. Pseudocode for FO-resampling process describes in Algorithm 1. B denotes the number of virtual particles. $\hat{x}_{i,n}$ represents a virtual particle and is randomly generated around a real particle x_i. A random number r is generated from a uniform distribution over $[-1, 1]$. σ_{x_i} measures the spread of $\hat{x}_{i,n}$ around x_i.

Algorithm 1 FO-resampling process

```
1: **for** $i = 1$ to $M$ **do**
2:     **for** $n = 1$ to $B$ **do**
3:         $\hat{x}_{i,n} = x_i + \sigma_{x_i} * r$
4:         $\hat{w}_{i,n} = p(z_t|\hat{x}_{i,n})$
5:         **if** $\hat{w}_{i,n} > w_i$ **then**
6:             $x_i = \hat{x}_{i,n}$
7:             $w_i = \hat{w}_{i,n}$
8:         **end if**
9:     **end for**
10: **end for**
```

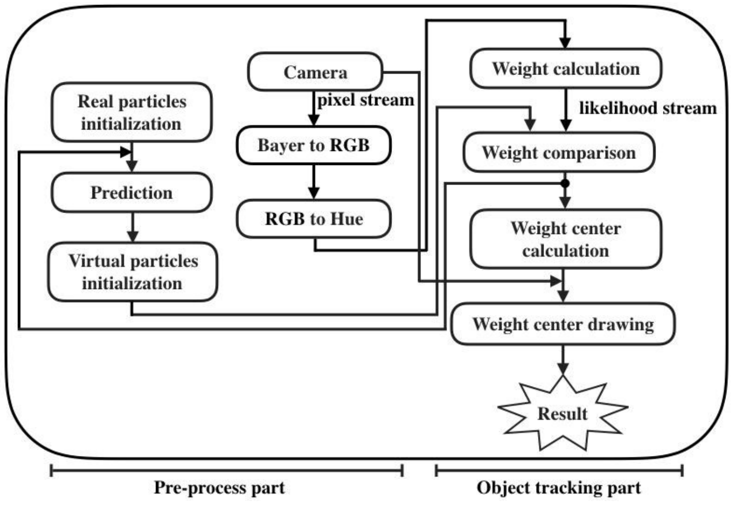

Figure 2: Overview of the proposed system

To the highest possible weight, the value of σ_{x_i} varies inversely with the initial weight. The weight of a virtual particle $\hat{w}_{i,n}$ is obtained from the likelihood calculation of virtual particle with the observation z_t at the location of object $\hat{x}_{i,n}$. If $\hat{w}_{i,n}$ is greater than w_i, x_i and w_i will be replaced by $\hat{x}_{i,n}$ and $\hat{w}_{i,n}$, respectively whereas x_i and w_i will keep the values in the opposite case. After being compared the weights of a real particle and the related B virtual particles, FO-resampling method replaces a particle with the largest weight. Since the FO-resampling is an approximated method, it does not produce exactly the same results with the conventional multinomial resampling. Especially, one of the potential weak points of using the FO-resampling would be that any particles even with very low weights are not eliminated through replacement. However, this drawback can be mitigated to an insignificant level by using an enough number of virtual particles [4]. On the other hand, the main benefit of the FO-resampling is that a process of each particle is fully independent and easily parallelized. Since no particle is eliminated or duplicated during the process unlike the multinomial resampling, no interaction among particles is needed. Therefore, parallel hardware architecture with multiple particle modules can be efficiently implemented on FPGAs, without devoting a chip are to a mechanism for inter-particle communication.

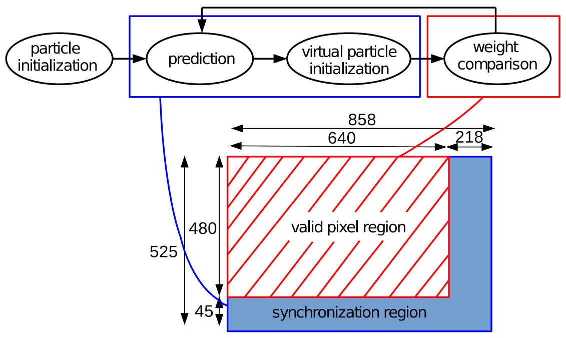

Figure 3: State transition diagram of weight comparison step

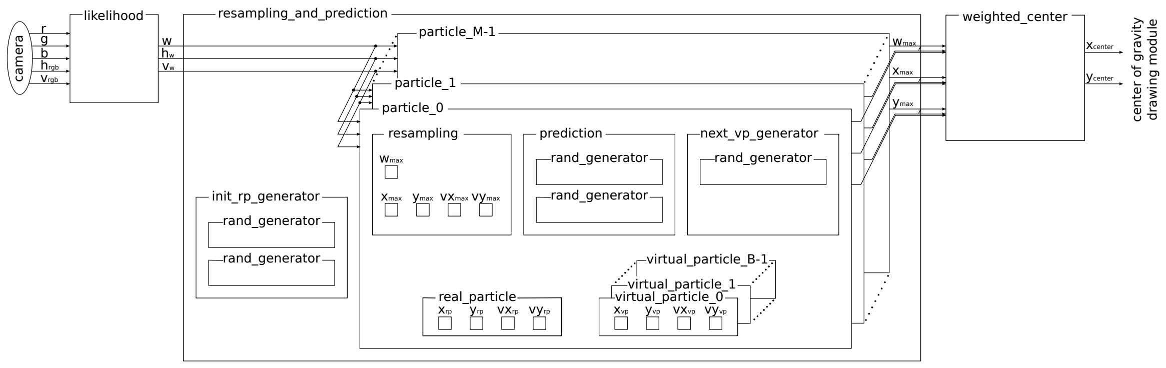

Figure 4: Overview of a particle filter module

4 Implementation

4.1 Overview

Our object tracking system with FO-resampling using deep-pipelined stream architecture is shown in Fig. 2. The whole system is synchronized with the pixel clock of the camera. Pixel data from the camera interface are streamed into the system for 1 pixel per 1 clock cycle and the weights of streamed pixel data are calculated in a pipelined manner. Compared with the weights of real and virtual particles, the maximum weight of particles are replaced as real particles. As shown in Fig. 3, the system is managed by a state machine with four states, which are initialization, prediction of real particles, setting of virtual particles, and weight comparison of both particles (resampling). Focusing on the fact that a video frame consists of a valid pixel region and a synchronization region, weight comparison is processed in the valid pixel region in a pipelined manner, while prediction and setting of virtual particles are processed in the synchronization region.

Fig. 4 presents an overview of our particle filter system. Firstly, RGB color values from camera are fed into a likelihood module, which calculates and outputs a stream of weights as well as the corresponding coordinate $(h_w,\ v_w)$. Then, the weights of the likelihoods are compared with M numbers of real particles, each with B numbers of virtual particles. After selecting the maximum weight of each particle, the coordinates and velocities of the selected particles are sent to a weighted center module, where the center of gravity of real particles calculated as an estimated position of the tracking target. In parallel with this weighted center calculation, prediction of the next states of the real particles and generation of new virtual particles are performed.

4.2 Likelihood Calculation

The likelihood module takes a stream of RGB pixels as an input, and outputs a stream of weights of likelihood in a pipelined manner. While the calculation process of likelihood may vary depending on tracking targets, we implemented a color-based object tracking function as an example in this experiment. Since RGB representation is not intuitive for color object detection, color space conversion to hue, saturation and intensity (HSI) model is performed [13]. In order to mitigate computational costs and the hardware amount, an integer-based RGB to hue conversion method [14] was adopted. In this method, hue value is calculated as:

$$H = \begin{cases} 42 + \left\lfloor \dfrac{42(G-B)}{\text{delta}} \right\rfloor & \text{if } R = \max(R,G,B), \text{delta} > \dfrac{\max(R,G,B)}{2} \\ 126 + \left\lfloor \dfrac{42(B-R)}{\text{delta}} \right\rfloor & \text{if } G = \max(R,G,B), \text{delta} > \dfrac{\max(R,G,B)}{2} \\ 210 + \left\lfloor \dfrac{42(R-G)}{\text{delta}} \right\rfloor & \text{if } B = \max(R,G,B), \text{delta} > \dfrac{\max(R,G,B)}{2} \\ -1 & \text{otherwise} \end{cases} \tag{6}$$

$$\text{delta} = \max(R,G,B) - \min(R,G,B) \tag{7}$$

where R, G, B are 8-bit color intensities for red, green and blue, while H is a hue value ranging from -1 to 252. Then, the difference value of hue (H_d) is calculated as follows:

$$H_d = \min(|H - H_t|, 253 - |H - H_t|) \tag{8}$$

where H_t shows the hue of the tracked object and was set to 40 for this experiment. In the final step, the weight of the likelihood of the input pixel is calculated as:

$$w = \begin{cases} \left\lfloor \alpha \exp\{-\frac{{H_d}^2}{2s^2}\} \right\rfloor & \text{if } H \neq -1, R \geq 64, \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where α denotes a parameter related to the scale of weights and s represents a parameter indicating the spread of weight distribution. In this implementation, α and s were set to 1023 and 20, respectively. Since H_d is an 8-bit integer value, the function in Eq. 9 can be simply implemented as a table in FPGAs.

4.3 Weight Comparison or Resampling

As shown in Fig. 5, a resampling module takes weights of likelihood (w) as an input stream, as well as the corresponding coordinates (x_w, y_w). Each resampling module includes one real particle and B virtual particles. The purpose of this module is to find the particle which has the coordinate of the maximum weight in the frame. This particle will become the next real particle in this resampling module.

Each particle compares its coordinate with the input coordinate every clock cycle in parallel. If one of the particles matches and the input weight is larger than the value of the w_{max} register, the register is updated by the input weight, so that the w_{max} register stores the maximum weight after processing all the weights in the frame. When the value of w_{max} is updated, x_{max}, y_{max}, vx_{max}, and vy_{max} are also updated by the states of the matched particle. Logically, the matched particle can send its states to the registers by a bus. However, since wired-OR buses cannot be implemented inside an FPGA chip, this mechanism was implemented as actual OR gates and multiplexers.

4.4 Virtual Particle Arrangement

After selecting the real particles in the resampling modules, the next states of the real particle are predicted according to Eq. 1 to Eq. 4. Then, new B virtual particles are generated by adding a random number ranging from $-\sigma$ to σ to the states of the real particle, to avoid too big or too small

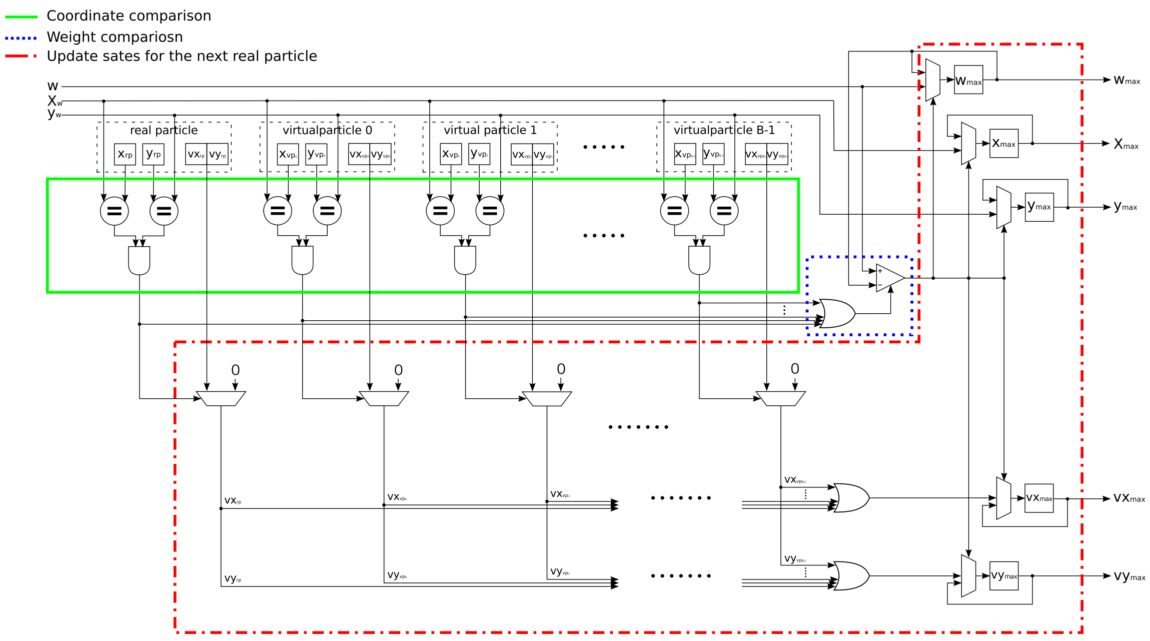

Figure 5: Resampling module

dispersion of virtual particles around one real particle. In this experiment, considering the ease of FPGA implementation σ was defined as:

$$\sigma = \left\lfloor \frac{1023 - w}{16} \right\rfloor \tag{10}$$

where the division by 16 can be simply implemented with a shift operation.

4.5 Center of Gravity Calculation

To the tracking of a moving model, the center of gravity calculation purposes for estimation of the object position across frames. The center of gravity calculation is given by:

$$g(x, y) = \left(\frac{\sum_{i=1}^{M} x_i w_i}{\sum_{i=1}^{M} w_i}, \frac{\sum_{i=1}^{M} y_i w_i}{\sum_{i=1}^{M} w_i} \right) \tag{11}$$

where x_i, y_i and w_i represent x and y coordinates and weight of the i-th particles, respectively. We also implemented a video output interface circuit on the same FPGA, where the center of gravity known as the intersection point of two lines is drawn on the object for tracing the trajectories of the object.

4.6 Random Number Generation

A linear feedback shift register (LFSR), which can generate two random numbers for each clock cycle, was implemented. By adding x^{32} in LFSR, the first random generator (rand1) generates the random bits in the range of 31 to 0 and the second one (rand2) selects the numbers from the range 32 through 1. Bitwise XOR operators provide the four feedback taps at 31, 21, 1, and 0-th bit. Thus, the characteristic polynomial is

$$x^{31} + x^{21} + x^{1} + 1. \tag{12}$$

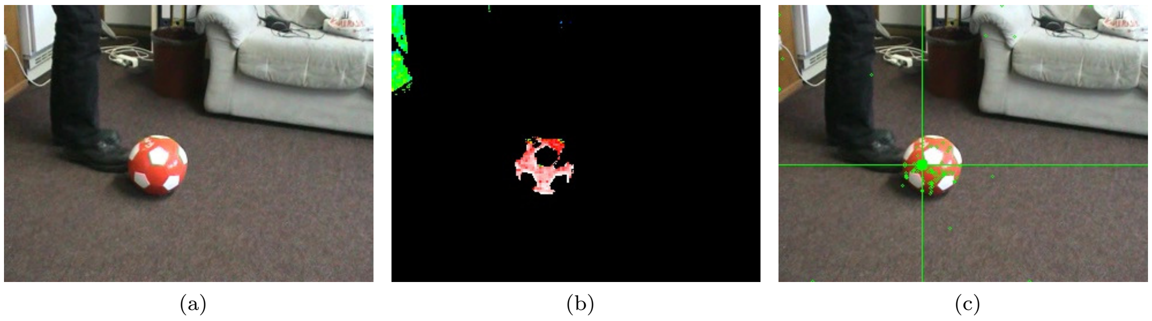

Figure 6: (a) A captured image from video and (b) Corresponding image of weight representation (c) Example frame of tracking results

A two-bit shift operation is performed every clock cycle. Random numbers generated by this module are used for initialization of particles, prediction of next states of particles, and arrangement of virtual particles.

5 Evaluation and Discussion

5.1 Preliminary Evaluation

At first, we evaluated the required number of real and virtual particles for the FO-resampling method in software before implementing on an FPGA, using an object tracking benchmark video [15] whose tracking target is a red soccer ball. The video-captured image and a snapshot of weight representation are shown in Fig. 6(a) and (b). An example of our tracking results is also shown in Fig. 6(c),

The estimation accuracy was evaluated using four metrics: Tracker Detection Rate (TDR), Average Tracking Error (ATE), Maximum Tracking Error (MTE) and Root Mean Square Error (RMSE) given by Eq. 13, Eq. 14, Eq. 15 and Eq. 16.

$$\mathrm{TDR} \;=\; \frac{F_t}{F} \tag{13}$$

$$\mathrm{ATE} \;=\; \frac{1}{F}\sum_{n\,=\,1}^{F}\sqrt{(x_g-x_n)^2+(y_g-y_n)^2} \tag{14}$$

$$\mathrm{MTE} \;=\; \max\Big(\sqrt{(x_g-x_n)^2+(y_g-y_n)^2}\Big) \tag{15}$$

$$\mathrm{RMSE} \;=\; \sqrt{\frac{1}{F}\sum_{n\,=\,1}^{F}(x_g-x_n)^2+(y_g-y_n)^2} \tag{16}$$

where F and F_t denote the total number of frames and successfully tracked frames in a video. The frame size for benchmark video is 320×240 and F is 602. A tracker is initialized in the first frame of a sequence and tracks the object of interest up to the end. The produced trajectory is then compared to ground truth using a number of measures specified in the particular experiment [16].

The ground truth data indicating the correct coordinate of the target object, which is distributed with the benchmark video, is given as the rectangular area for each frame. If the target object is inside a rectangular area, F_t will increase. ATE measures average discrepancy between centroid of ground truth bounding box and that of tracking system. MTE specifies the maximum tracking error between the center coordinate of each ground truth rectangular (x_g, y_g) and tracking coordinate

Table 1: Tracker Detection Rate of Each Implementation

TDR		No. of real particles: M				
		50	100	500	1000	2000
No. of virtual particles: B	20	0.934	0.935	0.935	0.939	0.942
	40	0.965	1.000	1.000	1.000	1.000
	50	1.000	**1.000**	1.000	1.000	1.000
	60	1.000	1.000	1.000	1.000	1.000
	80	1.000	1.000	1.000	1.000	1.000

Table 2: Average Tracking Error of Each Implementation

ATE		No. of real particles: M				
		50	100	500	1000	2000
No. of virtual particles: B	20	0.033	0.029	0.027	0.026	0.025
	40	0.023	0.018	0.017	0.016	0.016
	50	0.019	**0.017**	0.015	0.016	0.015
	60	0.020	0.018	0.016	0.016	0.016
	80	0.020	0.019	0.018	0.018	0.018

Table 3: Maximum Tracking Error of Each Implementation

MTE		No. of real particles: M				
		50	100	500	1000	2000
No. of virtual particles: B	20	0.452	0.315	0.387	0.365	0.368
	40	0.155	0.082	0.100	0.083	0.076
	50	0.106	**0.054**	0.048	0.048	0.045
	60	0.061	0.056	0.048	0.047	0.048
	80	0.085	0.063	0.055	0.053	0.052

Table 4: Accuracy Comparison of FO-resampling and Multinomial Resampling

M	Sampling method	F_t	$F - F_t$	TDR	ATE	MTE	RMSE
50	Multinomial	158	444	0.262	0.476	0.864	0.037
	FO ($B = 50$)	**602**	**0**	**1.000**	**0.019**	**0.106**	**0.001**
100	Multinomial	256	346	0.425	0.388	0.910	0.022
	FO ($B = 50$)	**602**	**0**	**1.000**	**0.017**	**0.054**	**0.001**
500	Multinomial	600	2	0.997	0.052	0.383	0.005
	FO ($B = 50$)	**602**	**0**	**1.000**	**0.015**	**0.048**	**0.001**
1000	Multinomial	600	2	0.997	0.051	0.171	0.004
	FO ($B = 50$)	**602**	**0**	**1.000**	**0.016**	**0.048**	**0.001**
2000	Multinomial	602	0	1.000	0.050	0.094	0.004
	FO ($B = 50$)	**602**	**0**	**1.000**	**0.015**	**0.045**	**0.001**

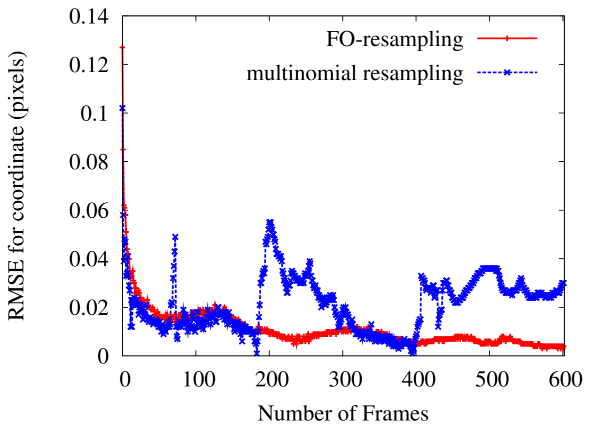

Figure 7: RMSE comparison of FO-resampling and multinomial resampling

Table 5: FPGA Mapping Result for the First Design (X1_NORMAL)

LUT	160,940(78.97%)
FF	177,229(43.48%)
BRAM	0(0.00%)
DSP48E	408(48.57%)
Max clock frequency (MHz)	28.00

$(x_n,\ y_n)$. When M is above 100, the tracker detecting rate and the accuracy errors are not much different as shown in Table. 1, Table. 2 and Table. 3. The number of real particles ($M = 100$) on the number of virtual particles ($B = 50$) is chosen as the appropriate parameters in terms of the reasonable tracking accuracy and less amount of resource utilization. Table. 4 shows the accuracy comparisons using FO-resampling ($B = 50$) and multinomial resampling on 602 frames. Obviously, FO-resampling can track successfully in all frames with smaller errors. Fig. 7 shows how tracking error changes along the frames with the two resampling methods. While the error with the FO-resampling was decreased along frame goes, the error was sometimes increased with the multinomial resampling, showing the effectiveness of the FO-resampling.

5.2 FPGA Mapping

We implemented the proposed system on a Kintex-7 XC7K325T FPGA using Xilinx Vivado 2016.3. According to the results of the preliminary evaluation, M and B were set to 100 and 50, respectively. The constraint for the clock frequency was set to 27 MHz, which is the pixel clock of a Video Graphic Array (VGA: 640×480 pixels) camera with the frame rate of 60 fps. Table. 5 illustrates FPGA resources for the first design, namely X1_NORMAL. As shown in the table, the clock constraint was met with the maximum clock frequency, achieving the throughput of 60 fps for VGA frames.

When it comes to the latency in clock cycles, latencies for the likelihood calculation ($L_{\text{likelihood}}$), resampling ($L_{\text{resampling}}$), prediction ($L_{\text{prediction}}$), virtual particle generation (L_{virtual}), and weighted center calculation (L_{center}) have to be considered. As aforementioned, the prediction and virtual particle generation are performed in parallel with the weighted center calculation. Thus, the total latency is given as:

$$L_{\text{total}} = L_{\text{likelihood}} + L_{\text{resampling}} + \max(L_{\text{prediction}} + L_{\text{virtual}}, L_{\text{center}}) \quad (17)$$

and each latency in this implementation is given as:

$$L_{\text{likelihood}} = 3 \tag{18}$$

$$L_{\text{resampling}} = (V_h + S_h) \times (V_v - 1) + V_h = (640 + 218) \times (480 - 1) + 640 = 411,622 \tag{19}$$

where V_h, V_v, and S_h are the horizontal length of the valid pixel region, the vertical length of the valid pixel region, and the length of the horizontal synchronization region, respectively. As shown in Fig. 3, $V_h = 640$, $V_v = 480$, and $S_h = 218$, in this implementation. Note that the resampling is performed for each clock cycle of pixel input and is finished when the last (the lower right) valid pixel is processed. Other latencies are as follows.

$$L_{\text{prediction}} = 1 \tag{20}$$

$$L_{\text{virtual}} = B + 1 \tag{21}$$

$$L_{\text{center}} = M + 36. \tag{22}$$

By assigning Eq. 18 $\sim$ Eq. 22, $M = 100$, and $B = 50$ to Eq. 17, we get:

$$L_{\text{total}} = 411,625 + \max(B + 2, M + 36) = 411,761. \tag{23}$$

On the other hand, as Fig. 3 shows, available clock cycles for one frame including the synchronization region is

$$L_{\text{frame}} = 858 \times 525 = 450,450. \tag{24}$$

Since $L_{\text{total}} < L_{\text{frame}}$, it has been shown that the implemented particle filter system achieves an in-frame operation.

Our design fits in XC7K325T which is a mid-range Kintex-7 device and achieves real-time performance in terms of both throughput and latency. At the same time, the mapping results in Table 5 suggest there is a possibility of design improvement for three reasons. Firstly, the resource utilization was not well-balanced especially due to no BRAM utilization. If we can change the balance of resource utilization, the design may fin in a smaller chip. Secondly, we can take up the slack of clock cycles in synchronization region. If so, then we can put more tasks into synchronization region to change resource balance. Finally, the required clock frequency of 27 MHz is too slow for typical FPGA design. The FPGA actually may operate much faster, by inserting registers in combinations logic paths. By increasing clock frequency, we can reduce the hardware amount by reusing hardware resources in a time-sharing manner. In Section 6, we implement and discuss the new designs for the sake of more compact design.

6 Improvement of Resource and Time Management

6.1 Design Alternatives

In this section, we focus on the impact of the resource and time management which is available to improve area and speed on FPGA. We implemented the following three additional design alternatives, aiming at reducing the hardware amount.

X1_SYNC_V The first attempt of improvement focuses on the resampling module shown in Fig. 5. The inefficiency comes from the $(B + 1)$-input OR gates, which are utilized to sent the states of the matched particle to registers. Considering we have not used any BRAM resources and there is a slack of the latency, we modified the task mapping of the resampling. Instead of completing the whole resampling task in the valid pixel region, we divided the resampling task into two sub-tasks: weight comparison and state reading. While the weight comparison is performed in the same manner with the first design, the velocity of the next real particle is sequentially searched and read out in the synchronization region. In this way, large amounts of OR gates can be eliminated as shown in Fig. 8. On the other hand, as Fig. 9 illustrates, BRAM with a 29-bit width and 50 depth is used

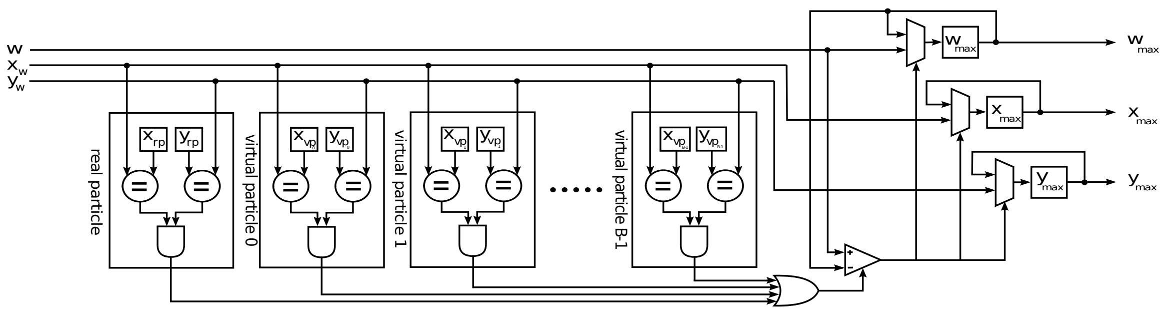

Figure 8: Weight comparison module of X1_SYNC_V

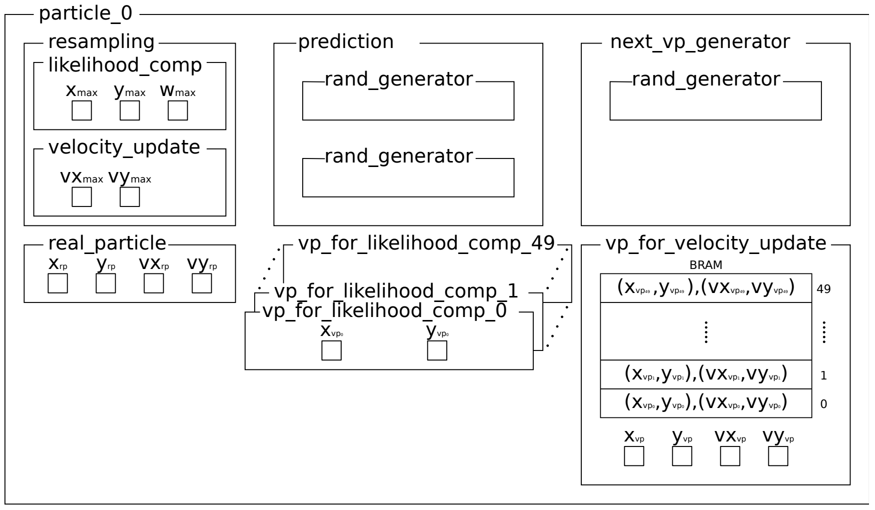

Figure 9: Overview of a particle filter module for X1_SYNC_V

to store velocity of the particles. In the weight comparison sub-task, the coordinate of the next real particle is found. After that, using this coordinate as a key, the corresponding velocity is searched on this BRAM. In addition, in order to reduce the logic amount for arithmetic, bit widths for each particle data are optimized.

X5_LUT_RAM This design is based on X1_SYNC_V, but operates at a 5 times higher clock frequency, that is 135 MHz. Thus, a new pixel is given in this design every 5 clock cycles. Comparisons with up to 5 different particles can be performed using the same logic in a time sharing manner. As shown in Fig. 10, the states of particles are stored in on-chip memory with a 190-bit width and 5 depth, since only one fifth of the particles are accessed at the same time. This memory is implemented as distributed RAM using LUTs.

X5_BRAM This design is based on X5_LUT_RAM. The difference is that the 5-depth table for particle states is implemented as BRAM, not as distributed RAM unlike the previous design.

6.2 Mapping Results

We implemented the above three designs on the Kintex-7 XC7K325T FPGA with 100 real particles and 50 virtual particles. Table. 6 summarizes the FPGA mapping results. Compared to the results of the original X1_NORMAL design (Table. 5) each new design reduces the utilization of LUTs and FFs while increasing BRAM usage. Especially, as shown in Table. 6, the X5_LUT_RAM design was successfully implemented on a smaller FPGA chip (XC7K160T), demonstrating that resource

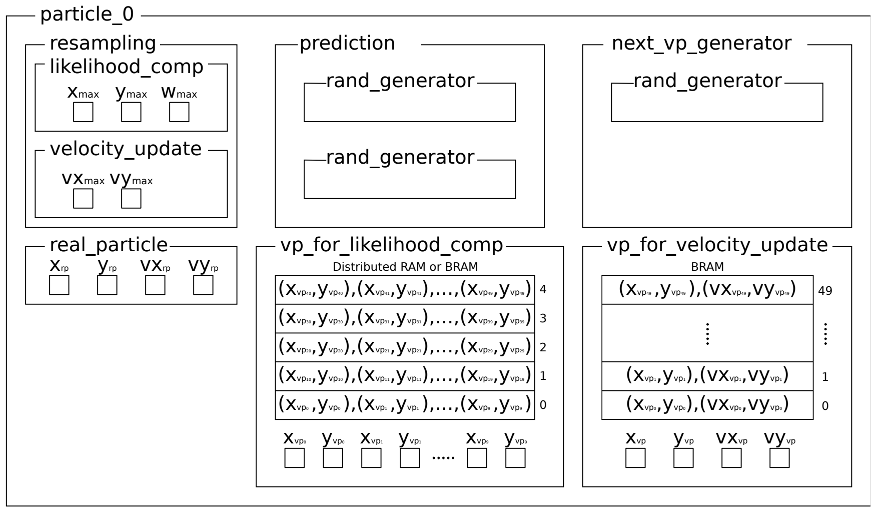

Figure 10: Overview of a particle filter module for X5_LUT_RAM and X5_BRAM

sharing with a high clock frequency leads to the compact architecture and the cost down. Although X5_BRAM achieved the highest resource reduction rates in terms of LUTs and FFs, it consumed approximately 80% of the BRAM offered by XC7K325T. That is why this design cannot be fitted in the smaller chip.

The clock constraints were met for each new design, achieving the throughput of 60 fps for VGA frames. The difference in the latency between X1_SYNC_V and X1_NORMAL only exits in the resampling:

$$L_{\text{resampling}} = 411,622 + B \tag{25}$$

since the state reading sub-task was added. Thus, we get:

$$L_{\text{total}} = 411,811 \tag{26}$$

which is also smaller than L_{frame} given by Eq. 24.

For X5_LUT_RAM and X5_BRAM, the latencies were changed as follows.

$$L_{\text{likelihood}} = 7 \tag{27}$$

$$L_{\text{resampling}} = (858 \times 479 + 640) \times 5 + B + 2 = 2,058,112 + B \tag{28}$$

$$L_{\text{prediction}} = 4 \tag{29}$$

$$L_{\text{virtual}} = B + 5 \tag{30}$$

$$L_{\text{center}} = M + 37. \tag{31}$$

For $M = 100$ and $B = 50$, the total latency becomes:

$$L_{\text{total}} = 2,058,119 + B + \max(B + 9, M + 37) = 2,058,306 \tag{32}$$

which is smaller than $L_{\text{frame}} \times 5 = 2,252,250$. Thus, it has been revealed that for each design alternative, realtime performance was achieved in terms of not only throughput but also latencies.

6.3 Power Consumption

We have demonstrated our particle filter design can be fitted in a smaller FPGA chip, by introducing resource time sharing with a higher clock frequency. Use of a smaller chip will also contribute in reduction of power consumption, while increase in the clock frequency has a negative effect for power

Table 6: FPGA mapping results for improved designs

Resources	X1_SYNC_V	X5_LUT_RAM (XC7K325T)	X5_LUT_RAM (XC7K160T)	X5_BRAM
LUT	102,750(50.42%)	81,895(40.18%)	81,868(80.74%)	69,203(33.96%)
FF	156,119(38.30%)	106,646(26.16%)	106,646(52.59%)	68,646(16.84%)
BRAM	51(11.46%)	51(11.46%)	51(15.69%)	351(78.88%)
DSP48E	203(24.17%)	203(24.17%)	203(33.83%)	203(24.17%)
Max frequency (MHz)	33.70	139.38	141.18	141.25

Table 7: Power Consumption Comparison

	X1_NORMAL	X1_SYNC_V	X5_LUT_RAM (XC7K325T)	X5_LUT_RAM (XC7K160T)	X5_BRAM
Dynamic Power [W]	0.842	0.772	2.289	2.186	3.530
Static Power [W]	0.165	0.169	0.179	0.127	0.209
Total on-chip power [W]	1.008	0.941	2.469	2.314	3.740

reduction. In order to analyze and discuss the tradeoff, we estimated power consumption of each design with Xilinx Vivado tool. Table. 7 shows the results.

By comparing X5_LUT_RAM on XC7K325T and the same design on XC7K160T, approximately 6% of total on-chip power reduction was shown. However, the power consumption for X5_LUT_RAM on XC7K160T was 2.7 to 3.0 times higher than the designs synchronized with the camera clock. That is, the increase in power consumption caused by the increase in the clock frequency cannot be compensated by downsizing the chip. In summary, large designs with a slow clock frequency were more efficient than small designs with a fast clock frequency in terms of power consumption.

7 Conclusion

This paper proposed efficient FPGA implementation of particle filters with an FO-resampling method. The software-based preliminary evaluation demonstrated that the FO-resampling method can achieve better object tracking quality compared to multinomial resampling by setting an appropriate number of real particles and virtual particles. The implementation experiment on a KC7K325T FPGA revealed that the proposed architecture achieved realtime performance of 60 fps for VGA images without using any external memory devices, by making best use of a stream processing approach in a valid pixel region while performing some sequential process in a synchronization region. In addition, by introducing a higher clock frequency than the pixel clock and by improving a balance of resource utilization, we demonstrated our design can be fitted in a smaller XC7K160T FPGA. However, it was also shown that the power consumption for a smaller design with a 5 times clock frequency was increased by 2.7 to 3.0 times, compared to a design implemented on a larger chip synchronizing with a slow camera pixel clock. Our challenging future work includes the evaluation of particle filters with a large number of particles for non-rigid objects and to find the best tradeoff point between chip size and clock frequency.

References

[1] A. Tahara, Y. Hayashida, Theint Theint Thu, Y. Shibata, and K. Oguri. FPGA-based real-time object tracking using a particle filter with stream architecture. In *Proc. 2016 Fourth International Symposium on Computing and Networking (CANDAR)*, pages 422–428, 2016.

[2] A. Athalye, M. Bolicand S. Hong, and P. M. Djuric. Architectures and memory schemes for sampling and resampling in particle filters. In *Proc. IEEE Digital Signal Processing Workshop*, volume 1, pages 92–96, 2004.

[3] A. Athalye, M. Bolic, S. Hong, and P. M. Djuric. Generic hardware architectures for sampling and resampling in particle filters. *EURASIP Journal of Applied Signal Processing*, 17:2888–2902, 2005.

[4] F. Schwiegelshohn, E. Ossovski, and M. Hübner. A fully parallel particle filter architecture for FPGAs. In *Applied Reconfigurable Computing*, pages 91–102. Springer, 2015.

[5] H. Matsubayashi, S. Nino, T. Aramaki, Y. Shibata, and K. Oguri. Retrieving 3-d information with FPGA-based stream processing. In *Proc. 16th ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, pages 261–261, 2008.

[6] K. Dohi, Y. Yorita, Y. Shibata, and K. Oguri. Pattern compression of FAST corner detection for efficient hardware implementation. In *Proc. IEEE 21st Int. Conf. Field Programmable Logic and Applications (FPL)*, pages 478–481, 2011.

[7] K. Negi, K. Dohi, Y. Shibata, and K. Oguri. Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm. In *Proc. Int. Conf. Field-Programmable Technology (FPT)*, pages 1–8, 2011.

[8] K. Dohi, Y. Hatanaka, K. Negi, Y. Shibata, and K. Oguri. Deep-pipelined FPGA implementation of ellipse estimation for eye tracking. In *Proc. IEEE 22nd Int'l Conf. Field Programmable Logic and Applications (FPL)*, pages 458–463, 2012.

[9] K. Nummiaro, E. Koller-Meier, and L. V. Gool. An adaptive color-based particle filter. *Image and vision computing*, 21:99–110, 2003.

[10] M. Sanjeev Arulampalam, Simon Maskell, and Neil Gordon. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[11] Sven F. and Wolfgang S. Adaptive probabilistic tracking embedded in a smart camera. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 134–134, 2005.

[12] L. Miao, J. J. Zhang, C. Chakrabarti, and A. Papandreou-Suppappola. A new parallel implementation for particle filters and its application to adaptive waveform design. In *Proc. IEEE Workshop on Signal Processing Systems (SIPS)*, pages 19–24, 2010.

[13] M. C. Hanumantharaju, G. R. Vishalakshi, S. Halvi, and S. B. Satish. A novel FPGA based reconfigurable architecture for image color space conversion. In *Global Trends in Information Systems and Software Applications*, pages 292–301. Springer Berlin Heidelberg, 2012.

[14] Thomas Bräunl. *Embedded robotics: mobile robot design and applications with embedded systems.* Springer Science & Business Media, 2008.

[15] BoBoT – Bonn benchmark on tracking. http://www.iai.uni-bonn.de/ kleind/tracking/.

[16] K. Mikolajczyk Z. Kalal and J. Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34:1409–1422, 2012.