IJCCS (Indonesian Journal of Computing and Cybernetics Systems)
Vol.14, No.3, July 2020, pp. 309~318
ISSN (print): 1978-1520, ISSN (online): 2460-7258
DOI: https://doi.org/10.22146/ijccs.57594

Multithreading Application for Counting Vehicle by Using Background Subtraction Method

Yohanssen Pratama*[1], Puspoko Ponco Ratno[2]
[1]Faculty of Informatics and Electrical Engineering, Institut Teknologi Del, Sitoluama, Indonesia
[2]Faculty of Tarbiyah, Institut Agama Islam Negeri Kediri, Jawa Timur, Indonesia
e-mail: ***[1]yohanssen.pratama@del.ac.id,** [2]puspoko.ponco@iainkediri.ac.id

Abstrak

Pemrosesan gambar dan video telah menjadi bagian penting dalam aplikasi sistem transportasi cerdas (ITS), terutama untuk mengumpulkan data lalu lintas jalan. Gambar-gambar yang dikumpulkan oleh kamera CCD (coupled device) biasanya diproses oleh beberapa algoritma pemrosesan gambar. Kode dalam aplikasi akan dieksekusi dalam sejumlah besar iterasi karena banyak algoritma yang terlibat dalam pemrosesan frame ketika gambar sudah ditangkap oleh kamera. Aplikasi pada umumnya akan memproses frame pertama hingga selesai dan melanjutkan ke frame berikutnya, sehingga aplikasi harus menunggu sampai frame yang masuk pertama kali selesai diproses. Jika algoritma yang dieksekusi cukup kompleks dan memiliki kompleksitas waktu yang tinggi maka akan ada frame yang didrop dari memory. Dalam penelitian ini kami mengusulkan implementasi multithreading untuk meningkatkan kinerja aplikasi sehingga data dapat diperoleh dalam waktu yang tepat dan setiap frame baru akan dapat diproses dalam waktu yang singkat. Kinerja aplikasi sebelum dan sesudah menggunakan multithreading didapatkan dengan membandingkan waktu perolehan data yang disimpan dalam database. Efektivitas aplikasi juga dapat ditentukan dengan menjalankan beberapa streaming video dengan resolusi yang sama.

Kata kunci*—background subtraction, image, intelligent transport system, multithreaded, parallel processing*

Abstract

Image and video processing has become important part in intelligent transportation system (ITS) application, especially for collecting road traffic data. Pictures that already collected by a charged coupled device (CCD) camera usually being processed by several image processing algorithms and the application's code will be executed in a large number of iteration because many algorithms are getting involved in processing the frame which captured by the camera. Typical application will process the first frame until finish and then continue to the next frame, so the application must wait until the first frame being processed. If the algorithms that executed quite complex and have a significant running time there will be a dropped frame and the time difference between data acquisition and real time video is divided by large margin. We proposed an implementation of multithreading to boost the application performance so the data can be acquire in real time and every new frame could be processed in short time. The application performance before and after using a multithreading is known by comparing the data acquisition time that stored in the database. The application effectiveness could define by running a multiple video streaming in same resolution.

Keywords—*background subtraction, image, intelligent transport system, multithreaded, parallel processing*

Received July 5th,2020; Revised July 21th, 2020; Accepted July 30th, 2020

1. INTRODUCTION

In recent years, image processing has been used for collecting the road traffic data by using camera as a sensors. As proposed by Anandhanarayanan and Govindaraj (2013) Image processing encompasses processes whose inputs and outputs are images and extract that images attributes [1]. Based on the definition, the image data that stored on the storage will be available to be processed as an input by some image processing algorithms. The real time processing is needed here in order to maintain the algorithm's speed in processing every frame. The multiple frames which came from different resources couldn’t be processed at the same time without fast time execution. We prefer a multi-threading approach to increase an algorithm speed so it could provide the data extraction in a real time.

Shanty and Anthony (2009) stated that multi-threaded process has multiple points of concurrent execution within the process [2]. The situation here where the single thread waiting for a resources pictures / frames made we need run another thread to process another frame from different resources. We use concurrency to run multiple threads in a single process. Figure 1. Proposed by Alda and Silvana (2013) show two threads within a process communicating each other through shared memory [3]:

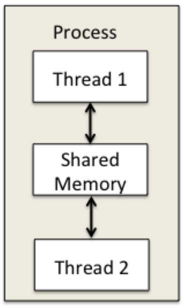


Figure 1 Communication between a pair of threads in a single process

The thread in Figure 1 shared the same address space. Thread 1 capture frames and thread 2 consumes the frames for image processing. So multiple frames grabbed and processed in different threads by using a circular buffer

The producer consumer problem which proposed by Dijkstra (1972) has purpose to make save way of passing tasks from producer to consumer threads. One or more producers are generating and placing data in a buffer, then one or more consumers are taking items out of the buffer [4]. In this work we only use 1 producer and 1 consumer thread, and try to utilize the idle time while waiting for the next frame to do the processing. If the capture thread grab the frame at 20fps (50ms) and the time that we need to capture the frame is only 2ms, in this case we have 48ms idle time. We could use this idle time to do some processing and it must be shorter than 48ms. If the processing is too long and exceed the 48ms, the frame/image will put as queue in the buffer. If buffer already full then we need to drop/discard some picture in the buffer.

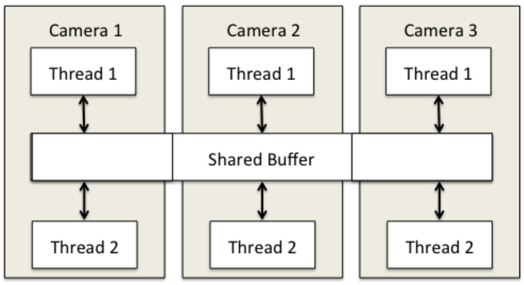


Figure 2 Communication between a pair of threads in a single process

Because we use multiple cameras and each camera have own capture and processing thread, so we will try to make the same buffer can be accessed by two or more threads at the same time on different parts. It means the different processing threads from each cameras could access the buffer at different parts (this model is depicted in Figure 2).

The producer consumer model presented above makes it possible to write highly concurrent multithreaded applications and it could exceed the mutex-based program if we use a multiprocessor machine, since more than one threads could access the different parts of buffer at the same time.

Outcome that expected by using multithread applications is to get the time precision data from extracting the real time video. The idle time that available between each frame capturing is being used to speed up the image processing algorithm, so the data could derived from the image in no matter of time. With this outcome we could process the traffic data in real time with no delay and give precision output in collecting the road traffic data (no information loss that caused by dropped frame).

2. METHODS

In this section will be explained about the research design that used to measure the effectiveness of multithreading implementation in speed up the image processing process. We tried to compare between the method that didn't use the multithreading process and the another one that implementing the multithreading.

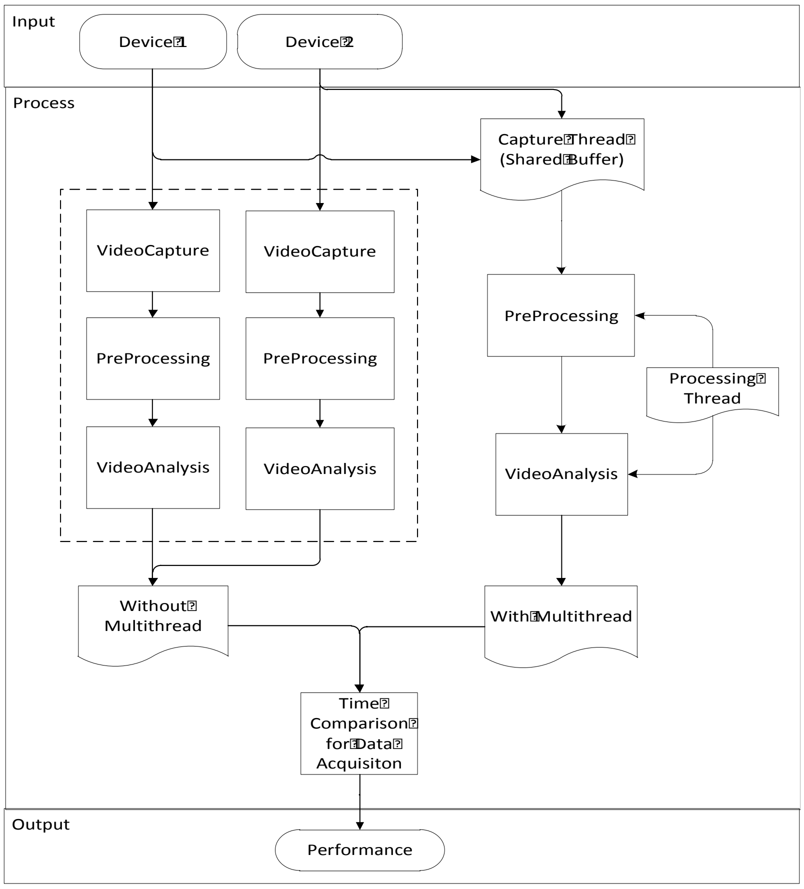


Figure 3 Research Design

In Figure 3 we could see that if we didn’t use a multithreading, the frame from each devices will be processed by separate algorithms. Which means that the frame from the ‘device 1’ will have it’s own capture and preprocessing unit, as well as ‘device 2’. So the idle time couldn’t be used to process another frame. Before the first frame has been processed by the algorithms, another frame must wait for 48ms (times required until the first frame finished being processed). Figure 3 above presented the research step and design of this paper:

2.1 Input

The data inputs are comes from every camera devices that used to capture an image. In this case the video was captured to be analyzed by image processing.

2. 2 Process

Video that has been captured will undergo a processing stage. In this stage performed the video processing using a background subtraction algorithm [5]. After the vehicle could be detect as an object, we use a two line counting method to capture the time when the vehicle passing the counting line. All image processing step shown by Figure. 4.

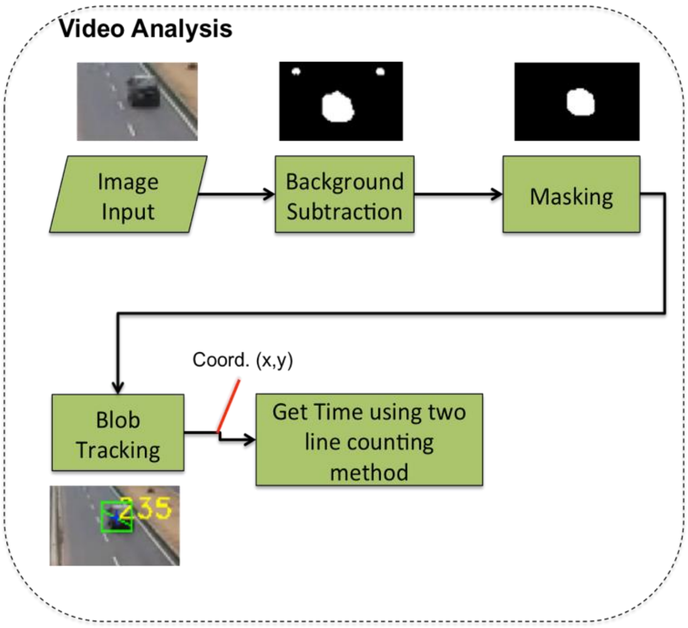


Figure 4 Diagram Block for Object Detection

For background subtraction, we used the Pixel Based Adaptive Segmenter (PBAS)[6] which is base on ViBe algorithm [7] and Sigma-Delta algorithm [8]. We used PBAS because this algorithm has a good background modeling. PBAS use a non-parametric background modeling, so the background is modeled by a history of recently observed pixel values [9]. On the other hand sigma delta (Σ-Δ) was lightweight algorithm and has a good processing speed, but we don’t use it alone here because we want to see the effect of the multithreading in speed-up the video analysis process. (using Σ-Δ alone is too lightweight)

For masking after the video recordings are processed using a background reduction algorithm, the current image only consists of binary images, each pixel can only have 2 added values. Foreground is usually generated by pixels. However, in order for the foreground to be more easily distinguished from the background, we need to refine this foreground and reduce existing noise by using the masking method or also called filtering. To determine the binary foreground mask, [10] suggests the use of a low-pass filter (LPF) by:

$$F_t(x,y) = \begin{cases} 1, & if\ |LPF(S_t(x,y)| > T \\ 0, & otherwise \end{cases} \quad (1)$$

The advantage of the Eq. above is the noise reduction in the foreground mask. where T is the threshold. However, Eq. (1) results in a foreground mask with real values [0, 1]. In the lower right image can be seen by masking some of the noise that appears in the left image becomes no longer visible [11].

For tracking a blob (blob tracking), we used a tracking algorithm based on “Appearance Models for Occlusion Handling” [12] because this algorithm has a good performance in complex environment. In the last step to count volume and vehicle velocity, we proposed a two-line counting method. Until the third step we use existing algorithm but for last step (vehicle count) we tried to implement the method that we proposed.

The first parameters can be obtained by activated the GetTickCountFunction (the function to count the running time) between the begin and end process in application. For the second parameter could be gain by using a two counting line method. Here is the brief explanation about two counting line method that adapted from Pratama et al. (2016).

The vehicle in the video that already detected as an object will have a centroid that represent the vehicle itself. This centroid located in the middle of rectangular mark that surrounded the vehicle. To obtain the time that needed by centroid to travel from the first counting line (counting line 1) to an additional counting line (counting line 2) we use a GetTickCountFunction in every line. We could see in Figure 5 line 2 serves as ‘counting line 2’ and line 1 as ‘counting line 1 [13]’.

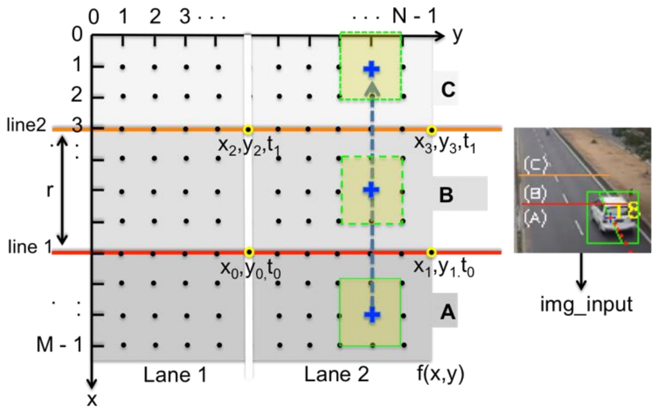


Figure 5 Additional Line 2 to Get the Parameters

The pseudocode to count a time that needed by the vehicle that move in x coordinate to reach line 2 from line 1 (Figure 5) could be seen below. If we want automatically detect whether the vehicle move in x or y coordinate, we need to added the new ‘conditional if ‘into the pseudocode line.

```
Function Count(centroid, line, state, width)
BEGIN
    if centroid.x < line1.x && centroid.x < line2.x
     state ← A;
      else if centroid.x > line1.x   && centroid.x < line2.x
       state ← B;
         else centroid.x > line1.x && centroid.x > line2.x
           state ←C;
  if oldstate == A && currentstate == B;
     $t_o$ = GetTickCount();
```

```
            else if oldstate == B && currentstate == C &&
    oldstate !=A && currentstate !=B
                $t_1$ = GetTickCount();
    t=$t_1$-$t_0$ ;
    END
```

In pseudocode above when the state of centroid were change from one state to another state then the GetTickCount function will capture the current time at instant. When the state change from A → B the t_0 time will be captured and this also applies for t_1 time that will be taken when the state change from B → C. After we get t we could use it as second parameter by averaging all t that stored in the system database. For the first parameter we could use a system date and time to capture the time when the first and the last object being detected in the video, after that we can know the running time to process the 10 minutes video.

2. 3 Output

The output of the application is the difference value between the application that didn't use a multithreading and those who used multithreading. We include the performance comparison testing between the application that use multithreading with different number videos running in background (1 until 5 video) and the application without multithreading feature. We will see the number of queue in the buffer and how long the application could running until it drop some frame if the buffer already full. Table 1 is the data output that will be stored in the database and used to analyzed the application performance. 'System time' will become the first parameter and the 'time to travel' become a second parameter to be analyzed. The number of centroid that passing counting line 1 and 2 could be seen in column line 1 and 2 respectively. The number of vehicle which represented by centroid in line 1 is higher than line 2, this happened because there were some centroids that hasn't been passed yet the counting line 2. System time and time to travel will be updated when there was a centroid that passing the counting line 2. This applied because for the 'system time' will be ended when the last centroid passed the counting line 2 also for 'time to travel' is completed when the centroid passed the counting line 2.

Table 1 Data Output

system time	**line 1**	**line 2**	**time to travel**
11:12:33	2	1	0.396
11:12:38	4	2	0.728
11:12:47	5	3	0.478
11:12:49	6	4	0.516
11:12:59	7	5	0.409
11:13:02	8	6	0.633
11:13:06	9	7	0.506

3. RESULTS AND DISCUSSION

In this experiment we use the hardware with current specification: Processor: Intel® Core ™ i5 CPU @ 2.5GHz, Memory(RAM): 4 GB 1600MHz, Harddisk: 500 Gb and the video that used for the test case have a 140x180 resolution (Figure 6). We use a low resolution video here to speed up the processing progress. Since the hardware that we used has a limited resources.

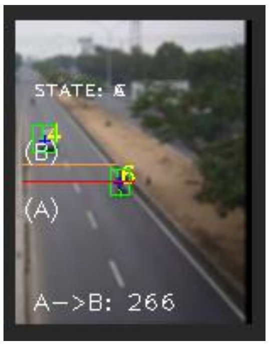


Figure 6 Test Case Video

Table 2 below is the comparison of first parameter (time that needed to process whole video) that already made between the application that use a multithreading (varies between 1 until 5 videos) and those which doesn't:

Table 2 First Parameter Time Comparison

application	first parameter (minutes)
w/o multithreading (1 video)	11.42
w/multithread (1 videos)	8.28
w/multithread (2 videos)	9.28
w/multithread (3 videos)	10.32
w/multithread (4 videos)	10.54
w/multithread (5 videos)	12.37

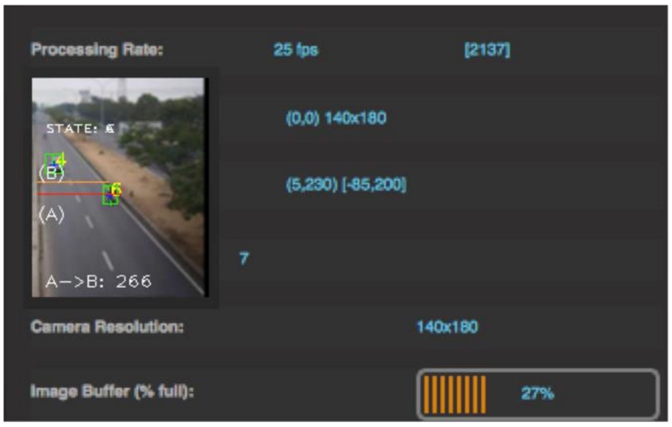
We could see that running time of the application with multithreading is outrun the application without multithreading, except if the multithreading application open and processing 5 videos simultaneously.

Table 3 Second Parameter Time Comparison

application	second parameter (seconds)
w/o multithreading (1 video)	1.0568
w/multithread (1 videos)	0.512
w/multithread (2 videos)	0.718
w/multithread (3 videos)	0.6677
w/multithread (4 videos)	0.7093
w/multithread (5 videos)	0.7686

In table 3, the second parameters that shown an average time that needed by the vehicle to travel between the line 1 and line 2, it seems that the multithreading application still outrun the application that didn't implemented the multithreads. Even though 5 videos play simultaneously, the multithreading application still have advantage in terms of performance.

To see the frame queue behavior in this application, we set buffer size into 999 frames, so it can hold 999 frames in the queue. The greater number of videos that play simultaneously in the application, the faster the frame queues will grow and the buffer being filled. We will see the frame queue growth rate in the buffer if we run more than 1 video simultaneously. Based on observation the maximum frame that piles up in the buffer when running 1 video is up to 16 frames. This happen when there were many objects that must be detected in the video. After that, the frame will reduce again to zero because the processing rate could overcome the capture rate. Below (Figure 7) is the image of the application and its buffer when running.



Processing Rate: 25 fps [2137]

(0,0) 140x180

(5,230) [-85,200]

7

Camera Resolution: 140x180

Image Buffer (% full): 27%

Figure 7 Test Case Video

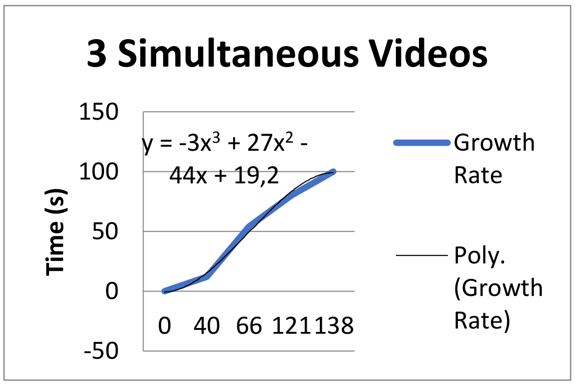
If we run 2 simultaneously video, the percentage of buffer that being occupied by the frames is around 22%. After some periods, it will reduce again and fluctuated between 0 until 22 percent but never overcome the 22%. But for the other case, when we run 3 or more videos simultaneously, the buffer will be full for a certain amount time and never been reduce again. This happened because the capture rate exceeded the processing rate. The frame rate of growth is faster here because it being supplied by 3 video resources. The relationship between time and the growth of frames can depict on Figure 8. The polynomial functions (Figure 8) for every parallel running videos are shown an increasing frame growth in the queue during the application running. The greater number of videos that running parallel, the more steep the gradient function. We get the polynomial function for the application that running 3 videos simultaneously as below:

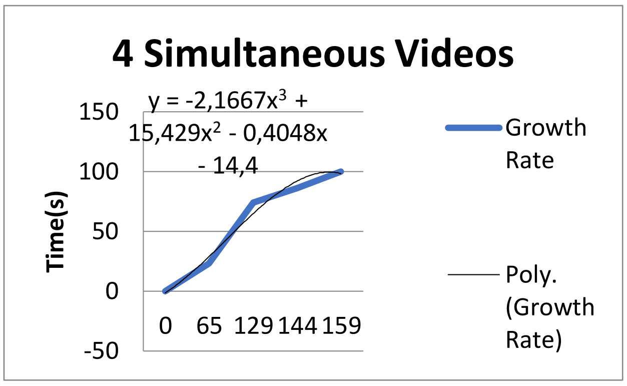
$$y = -3x^3 + 27x^2 - 44x + 19.2 \qquad (2)$$

and for 4 videos that running simultaneously, we get the different function:

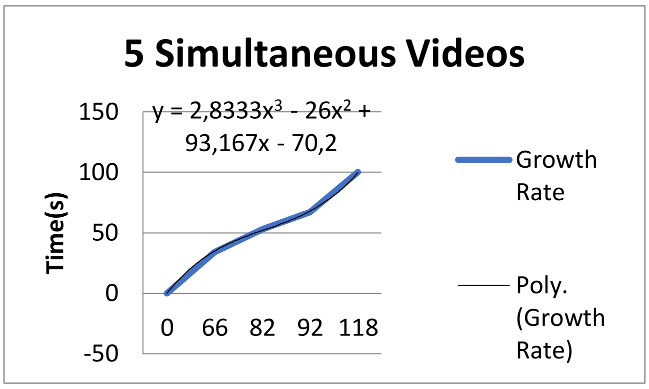
$$y = -2.1667x^3 + 15.429x^2 - 0.4048x - 14.4 \qquad (3)$$

If we see both functions, we could conclude that the more running videos that run simultaneously, the more steep the gradient function would be.





(a) (b)



(c)

Figure 8 Growth of rate 3 simultaneous videos (a), Growth of rate 4 simultaneous videos (b), Growth of rate 5 simultaneous videos (c)

We also can see from the chart that the greater number of videos that run simultaneously the more faster the rate of grow frames in the buffer. We have been seen that the polynomial functions that obtained from the regression process show us the same thing.

If we see the number of object that being detected during some period of time, we can conclude that the number of object that could be process by multithreading application is larger than the non multithreading application. It show on the Figure 9 that in shorter time the more number of object could be processed by multithreading application.

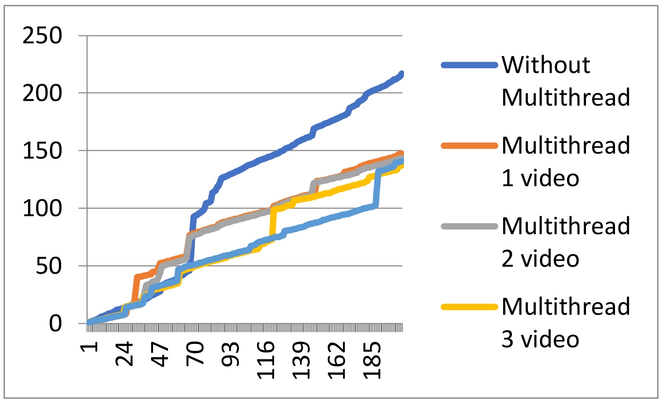


Figure 9 Number of object that being processed versus time

4. CONCLUSIONS

This method could be used to build a vehicle counting application for the real time condition. Because if we don't use the multithreading there will be delay in processing frame that will lead to application crash due to buffer overflow. The delay is about 3.14 minutes if the application only running 1 videos with 10 minutes duration. If we know the relation between time and the rate of growth frame in the buffer we could avoid such condition that leading to non-optimal performance. From the experiment that already done it can conclude that the multithreading could speed up the image processing in the application and improve the application running time. To prevent the buffer overflow if we run plenty videos, we could use a drop buffer scheme if the buffer already full.

ACKNOWLEDGEMENTS

This work was supported in part by Institut Teknologi Del and BlackBerry Innovation Center ITB.

REFERENCES

[1] K. Anandhanarayanan and R. Govindaraj, "High Performance Color Image Processing in Multicore CPU using MFC Multithreading," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 12, 2013 [Online]. Available: https://dx.doi.org/10.14569/IJACSA.2013.041207 [Accessed: 01-Jul-2020]

[2] M. Shanti and A. Anthony, "Multithreading – An Efficient Technique for Enhancing Application Performance," *International Journal of Recent Trends in Engineering*, vol.2, no.4, 2009.

[3] K. Alda and G. Silvana, "Multithreading Image Processing in Single-core and Multi-core CPU using Java," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 9, 2013 [Online]. Available: https://dx.doi.org/10.14569/IJACSA.2013.040926 [Accessed: 01-Jul-2020]

[4] E.W. Dijkstra, "Information Streams Sharing a Finite Buffer," *Information Processing Letters*, vol. 1.5, no. 1072, p. 179-180, 1972.

[5] A.Sobral, , C. Baker, T. Bouwmans, and E. Zahzah, "Incremental and multi-feature tensor subspace learning applied for background modeling and subtraction," *International Conference on Image Analysis and Recognition, ICIAR 2014*, 2014 [Online]. Available: https://doi.org/10.13140/2.1.4886.4322 [Accessed: 01-Jul-2020]

[6] M. Hofmann, P. Tiefenbacher, and G. Rigoll, "Background segmentation with feedback: The pixel-based adaptive segmenter," *IEEE Workshop on Change Detection*, p. 38-43, 2012 [Online]. Available: https://doi.org/10.1109/CVPRW.2012.6238925 [Accessed: 01-Jul-2020]

[7] O. Barnich and M. Van Droogenbroeck, "ViBe: A Universal Background Subtraction Algorithm for Video Sequences," IEEE Transactions on Image Processing, p. 1709-1724, 2011 [Online]. Available: https://doi.org/10.1109/TIP.2010.2101613 [Accessed: 01-Jul-2020]

[8] A. Manzanera and J. Richefeu, "A new motion detection algorithm based on sigma-delta background estimation," *Pattern Recognition Letters*, p. 320–328, 2007[Online]. Available: https://doi.org/10.1016/j.patrec.2006.04.007 [Accessed: 01-Jul-2020]

[9] R. Laganier, 2011, "*OpenCV2 Computer Vision Application Programming Cookbook*," Packt Publishing.

[10] M. Sigari, N. Mazayani, H. Pourreza, "Fuzzy running average and fuzzy background subtraction: concepts and application", *Int. J. Comput. Sci. Network Security*, vol.8, no. 2, p. 138-143, 2008.

[11] Y.-T. Chen, C.-S Chen, C.-R. Huang, Y.-P. Hung, "Efficient hierarchical method for background subtraction," *Pattern Recognition*, vol. 40, issue 10, p. 2706-2715, 2007 [Online]. Available: https://doi.org/10.1016/j.patcog.2006.11.023 [Accessed: 25-Jul-2020]

[12] A. Senior, A. Hampapur, Y. Tian, L. Brown, S. Pankanti, and R. Bolle, "Appearance models for occlusion handling," *Image and Vision Computing*, vol.24, p. 1233-1243, 2006 [Online]. Available: https://doi.org/10.1016/j.imavis.2005.06.007 [Accessed: 01-Jul-2020]

[13] Y. Pratama and B. Nugraha, "Vehicle Counting and Classification for Traffic Data Acquisition," *Jurnal Teknologi*, vol. 78, p.77-82, 2016.