IJCTEE Innovation Research Destination

ISSN 2249-6343

International Journal of Computer Technology and Electronics Engineering (IJCTEE)

Volume 1, Issue 3

# OpenCV Based Real-Time Video Processing Using Android Smartphone

**Ammar Anuar, Khairul Muzzammil Saipullah, Nurul Atiqah Ismail, Yewguan Soo**

***Abstract*— as the smarphone industry grows rapidly, the smartphone application needs to be faster and consumes lower power because the smartphone is only powered by a battery. In this paper, two Android applications based on video processing method are introduced; one by using OpenCV library, the other one is using Android library with self-implemented algorithm called CamTest. Eight image processing methods are applied to each frame of the video captured from the Android smartphone. The smartphone used in this study is the Samsung Galaxy S, with Android 2.3 Gingerbread Operating System. The efficiencies and power consumptions of the two applications are compared by observing their frame processing rate and power consumption. The experimental results show that out of the eight image processing methods, six methods that executed using OpenCV library are faster than that of CamTest with a total average ratio of 0.41. For the power consumption per frame test, six methods that executed using OpenCV library consume less power than that of CamTest with a total average ratio of 0.39.**

***Index Terms*— Android, computer vision, OpenCV, power consumption.**

## I. Introduction

Smartphone – the combination between the personal digital assistant (PDA) and mobile phone has totally changed the myth about mobile phone which is only mobile phone company can develop its application. Since the launch of the Android operating system (OS) [3] in 2007, mobile development has been high in demand [4]. Android is developed by Google and is based upon the Linux kernel and GNU software.

Recently, Android has reached great success in mobile operating system especially in smartphones and tablets. New versions of Android are being updated continuously to satisfy android users. Due to these circumstances, Android developers introduce new application to satisfy the needs of the Smartphone users. Libraries such as OpenGL (Open Graphics Library) and OpenCV (Open Computer Vision) [1] are used for the development of the application. Android application developers tend to interface hardware into their application such as camera, sensors, compass, Bluetooth, Wi-Fi and etc. Application that uses camera usually involves an image processing method such as Gaussian, Median, Mean Laplacian, Sobel filter and others. Developers who have basic knowledge about image processing can write their own codes to apply those image processing methods in their application but for the one who does not have any basic about image processing will face a lot of difficulties creating their applications. Developers usually prefer to import libraries in their work. In the image processing field, an open source image processing library known as OpenCV had made developers can apply image processing methods easily in their work. Nowadays OpenCV library has widely implemented in several of image processing projects such as in building a robot that can distinguish some objects [2].

The increasing need for low power systems had reflected Android developers to consider power consumption in their applications. Power dissipated in any embedded device can be reduced with hardware optimization techniques, which only applied in earlier design steps [5]. Another way to reduce power consumption is software transformation. In software optimization techniques, power dissipation can be reduced with compiler, instruction-level, and source code-level optimization methods [6]. Source code optimization has benefits in terms of readability, portability, and maintenance [7], [8]. Some research done in embedded software optimization have shown that source code optimization techniques tend to reduce power consumption [10].

In this work, we made the comparison between our own video processing implementation and OpenCV video processing implementation in term of performance and power consumption. To evaluate the efficiency performance, the frame processing rate is measured. We also use PowerTutor [9] application to measure the power consumption of each application.

This document is organized as follows: in section 2, related work in video processing method is discussed. Section 3 illustrates the methodology implemented and section 4 shows the results obtained and the analysis performed. Finally, conclusions are presented.

***Ammar Anuar***, *Faculty of Electronic Engineering and Computer Engineering, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Melaka, Malaysia. (e-mail: m021110015@student.utem.edu.my)*

***Khairul Muzzammil Saipullah***, *Faculty of Electronic Engineering and Computer Engineering, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Melaka, Malaysia. (e-mail: muzzammil@utem.edu.my).*

***Nurul Atiqah Ismail***, *Faculty of Electronic Engineering and Computer Engineering, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Melaka, Malaysia. (e-mail: m021110036@student.utem.edu.my).*

***Yewguan Soo***, *Faculty of Electronic Engineering and Computer Engineering, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Melaka, Malaysia. (e-mail: soo@utem.edu.my).*

## II. OpenCV in Android Platform

The OpenCV library was officially introduced in 1999 by Intel Research initiative to advance CPU-intensive application [1]. The OpenCV library in the earlier version written in C, However since version 2.0, OpenCV includes both C interface and C++ interface. Starting version 2.2, OpenCV can be built for Android OS. The latest OpenCV version, OpenCV 2.3.1 (beta2) was launched August 2011.

In OpenCV 2.3.1 for Android library, they also included samples image processing code using camera such as face detection, FAST feature finder that use combination of Java and C++. In order to make an Android application to be able to write in C++, the C++ parts have to be built before executing the whole project. The most popular way to build C++ parts is by using Android native development kit (NDK) together with Cygwin: Linux-like environment for Windows. The project folder will be accessed by Cygwin, and then it will be built by a file from Android NDK, which is the ndk-build file.

Some improvements made in OpenCV 2.3.1 are currently about 700 unique OpenCV methods/functions are available in Java, added OpenCV native camera support for armv5te devices and added two detailed tutorials for quick start of development with OpenCV for Android.

## III. Real-Time Video Processing In Android With OpenCV

As explained earlier, the real-time video processing conducted in this paper is divided into two groups which are the OpenCV library group and build in Android library group that we called CamTest. Firstly, the OpenCV library needs to be linked with an integrated design environment (IDE). In our case, the OpenCV library is linked with Eclipse IDE and Android software development kit (SDK) and NDK.

We exploit the OpenCV's Imgproc.java class to perform the image processing methods for the OpenCV library group. For the Android library group, we only utilize the raw data from android.hardware.Camera and android.hardware. Camera.PreviewCallback as the input frame image of self-made image processing algorithms. The algorithm we applied is the basic algorithm of the image processing method without any source-code level optimization. For the CamTest, a standard loop is conducted to each frame using YUV to RGB conversion, YUV to gray image conversion, image thresholding, image blurring with mean and Gaussian filter, noise removal with median filter, edge detection with Laplacian and Sobel operator image processing methods. On the other hand, for the OpenCV library, the functions cvtColor( ), threshold( ), blur( ), GaussianBlur( ), medianBlur( ), Laplacian( ) and Sobel( ) are applied.

In OpenCV library the frame data are saved in the Mat structure. This Mat structure is then passed to the OpenCV's image processing functions in order to process each pixel in the frame. Meanwhile, for CamTest, the data are saved in one dimensional byte array that is obtained from the Android library.

## IV. Experiments And Discussions

In order to evaluate the efficiency and power consumption of the video processing, eight basic image processing methods are applied to each frame captured from the 5 mega pixels camera of Samsung Galaxy S's smartphone. The Samsung Galaxy S is powered by 1 GHz ARM Cortex-A8 processor running with Android 2.3 Gingerbread OS. The eight image processing methods are conducted using both OpenCV library and CamTest in order to compare the efficiency and power consumption between the OpenCV and the build-in Android library on an embedded device namely the smartphone. Each image processing method is iterated 30 times and the average value is recorded for each experiment.

**Table I**: **Frame processing methods and its description.**

| **Frame Processing** | **Description** |
|---|---|
| RGB | Convert The Original YUV Color Space To RGB Color Space |
| Grayscale | Convert the Y color space to 0~255 grayscale |
| Threshold | Threshold the grayscale pixel with 70 |
| Mean | Filtering the grayscale frame with average of all the pixel values in a 3x3 window |
| Gaussian | 2D convolution with Gaussian 3x3kernel |
| Median | Filtering the grayscale frame with median of all the pixel values in a 3x3 window |
| Laplacian | 2D convolution with Laplacian 3x3 kernel |
| Sobel | Filtering of the grayscale frame in horizontal and vertical direction using 3x3 Sobel operator |

The description the eight image processing methods are shown in Table I. The input format from the Samsung Galaxy S camera is in YUV color space. So it needs to be converted to RGB color space for video processing in standard color space. For video processing in grayscale, the luma (Y) is mapped to 0~255 scale. For RGB processing, all the channels in YUV color space are used to convert from the YUV space into the RGB space. And lastly, for the threshold, mean, Gaussian, median, Laplacian and Sobel image processing, the resulting grayscale frame from the grayscale processing method is utilized.

The YUV to RGB conversion formula is calculated using

$$\begin{aligned} R &= 1.164(Y-16) + 1.596(V-128) \\ G &= 1.164(Y-16) - 0.813(V-128) - 0.391(U-128) \\ B &= 1.164(Y-16) + 2.018(U-128) \end{aligned} \tag{1}$$

For image thresholding each pixel is thresholded against a constant number *T*. If the pixel value larger than *T*, the pixel value will set to 1, otherwise the pixel value will be set to 0. The image thresholding can be calculated using the following formula:

$$g(x,y)=\begin{cases}1, & \text{if } f(x,y)>T\\ 0, & \text{otherwise}\end{cases} \qquad (2)$$

where, $f(x,y)$ is original frame and $g(x,y)$ is thresholded frame.
To remove the noise from the frame using median filter, each 3x3 window of the original frame is processed by calculating the median value of the whole pixels in 3x3 window. This median value is then will be the new pixel value on the median filtered frame.

For video blurring each frame is convolved using a 3x3 mask. For Gaussian blurring, the frame will be convolved with the 3x3 mask as shown in Fg. 1(a). For mean filter, the frame will be convolved with 3x3 mask as shown in Fig. 1(b)

| | | |
|---|---|---|
| 1/16 | 2/16 | 1/16 |
| 2/16 | 4/16 | 2/16 |
| 1/16 | 2/16 | 1/16 |

(a)

| | | |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

(b)

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | -8 | 1 |
| 1 | 1 | 1 |

(c)

**Fig. 1. (a) Gaussian mask, (b) Mean filter mask, (c) Laplacian mask**

For edge detection,each frame is convolved using a 3x3 mask. For Laplacian, the frame will be convolved with the 3x3 mask as shown in Fig. 1(c). The Sobel edge detection uses two 3×3 masks which are convolved in the *x* and *y* direction with the original frame. The two 3x3 masks are as shown as in Fig. 2.

| | | |
|---|---|---|
| -1 | 0 | -1 |
| -2 | 0 | -2 |
| -1 | 0 | -1 |

(a)

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

(b)

**Fig. 2. (a) The *x*-direction Sobel 3x3 mask. (b) The *y*-direction Sobel 3x3 mask.**

Fig. 3 and Fig. 4 show the image processing methods using OpenCV library and the CamTest. Those images were captured in real-time using Samsung Galaxy S camera at the same position for the eight different methods as explained above. The output images were quite same for the grayscale, mean, Gaussian, median processing. One of the significant differences between the OpenCV library and CamTest can be seen in the resulting image of Laplace edge detection. In OpenCV library, the background region is dark whereas in the case of the CamTest, the edge in the background region is clearer. The reason is in the background regions, the Laplacian( ) function from the OpenCV only returns raw output data of convolution. To produce a clear and high contrast edge, the output data of the convolution need to be scaled into an appropriate range.

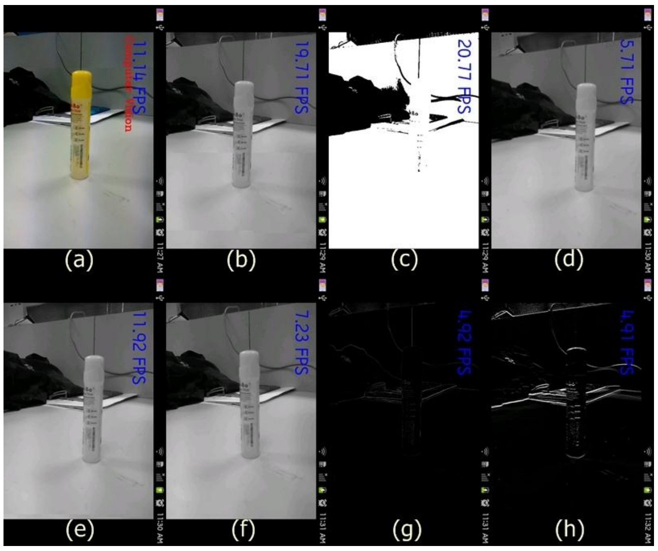


**Fig. 3. OpenCV library implementation. Those images were captured when it processes images in real-time video processing. (a) is RGB image, (b) is Greyscale image, (c) is Threshold image, (d) is Mean filter image, (e) is Gaussian image, (f) is Median filter image, (g) is Laplacian filter image, (h) is Sobel filter image.**

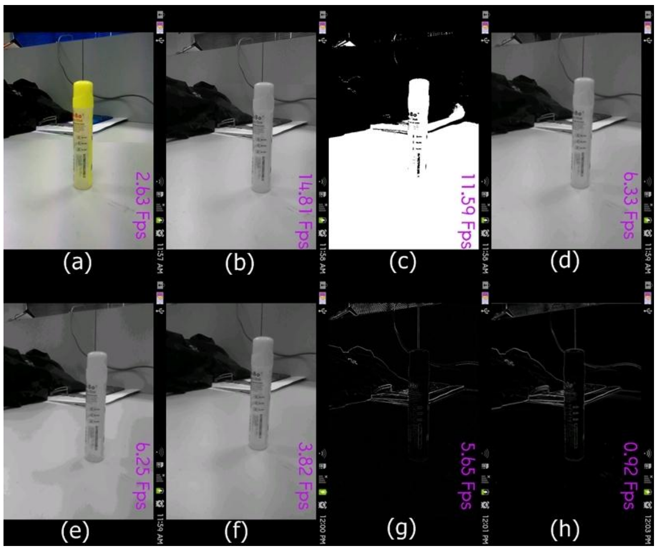


**Fig. 4. CamTest algorithm implementation. Those images were captured when it processes images in real-time video processing. (a) is RGB image, (b) is Greyscale image, (c) is Threshold image, (d) is Mean filter image, (e) is Gaussian image, (f) is Median filter image, (g) is Laplacian filter image, (h) is Sobel filter image**

## A. *Efficiency Test*

In order to evaluate the frame processing efficiency between OpenCV library and CamTest, the frame processing rate (FPR) is calculated and observed. The formula to calculate the frame processing rate is as follows:

$$FPR=\frac{No.\ of\ processed\ frame}{1s} \qquad (3)$$

The unit for the FPR is frame per second (*fps*). It is the number of frames the image processing algorithm can be processed in one second. The higher the value of the FPR, the more efficient the method is. Fig. 5 shows the frame processing rate of eight image processing methods using OpenCV library and CamTest. As it can be seen, the chart shows a significant FPR difference between the OpenCV library and CamTest for the RGB, grayscale, threshold and Gaussian processing. A bit unexpected result showed for mean and Laplacian methods because for these two methods, CamTest achieves FPR higher than that of OpenCV. This may be cause by the similar algorithm applied for both mean and Laplacian methods in OpenCV and CamTest.

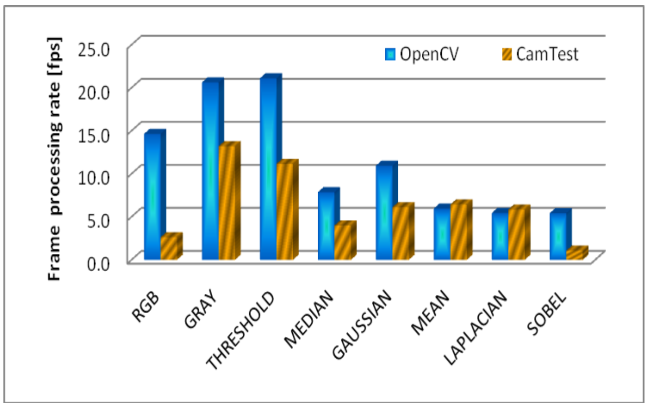


**Fig. 5. Frame Processing Rate in OpenCV library and CamTest for the eight image processing methods.**

The Gaussian, mean and Laplacian methods should result in the similar FPR because they apply the same convolution algorithm by moving the 3x3 mask on the image. In the case of CamTest, it can be seen that the FPR of those methods are similar. This is because in the CamTest we use the same algorithm to compute the convolution for those three methods. However, in the case of OpenCV library, the Gaussian blurring executes almost two times faster than mean and Laplacian methods. This shows that the GaussianBlur( ) function utilizes difference convolution algorithm compared to that of mean and Laplacian convolution algorithm.

To evaluate how much better the FPR of OpenCV compared to that of CamTest, the FPR ratio is calculated. The FPR ratio is calculated using the following formula:

$$FPR\ ratio = \frac{OpenCV\ FPR - CamTest\ FPR}{\max(OpenCV\ FPR, CamTest\ FPR)} \qquad (4)$$

If the FPR ratio is a positive number *N*, it means that the FPR of OpenCV is 1/*N* times better than CamTest. If the FPR ratio is a negative number –*M*, it means that the FPR of CamTest is 1/*M* times better than OpenCV. The overall FPR ratios of the eight image processing methods are shown in Table II. The total average FPR ratio is 0.41. This means that overall, OpenCV is 1/0.41 or 2.4 times faster than the CamTest.

**Table II**: **The FPR ratio of the eight image processing methods**

| Frame Processing | FPR ratio |
|---|---|
| RGB | 0.82 |
| Grayscale | 0.36 |
| Threshold | 0.47 |
| Mean | -0.07 |
| Gaussian | 0.44 |
| Median | 0.49 |
| Laplacian | -0.07 |
| Sobel | 0.80 |
| Total Average | 0.41 |

### *B. Power Consumption Test*

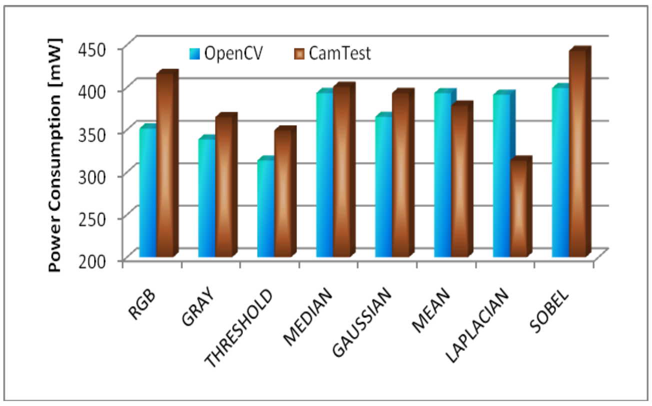


**Fig. 6. Power consumption average in 30 sec between OpenCV library and CamTest for the eight image processing methods.**

The power consumption test was conducted by using PowerTutor application. Each method whether in OpenCV library or in CamTest will be running in 30 seconds and the power consumption for each second will be recorded. The average of the power consumption in the 30s will be taken to be evaluated. Fig. 6 shows the average power consumption of the eight image processing method in the OpenCV library and the CamTest that is obtained from the PowerTutor application. The lower the power consumption, the better the library is. Overall, the power consumptions of OpenCV and CamTest are quite similar. OpenCV consumes less power compared to CamTest for almost of the image processing methods except for the mean and Laplacian methods. The Laplacian method that applied in Camtest consumes very less power compared to the one that is applied in CamTest.

However, the chart in Fig. 6 does not consider the number frame each method processed during the 30s of the power consumption test. So, in order to evaluate the real power consumption, the power consumption per frame (PCPF) is calculated. The formula to get power consumption is as follows:

$$PCPF = \frac{Average\ power\ consumption}{No.\ of\ frame} \qquad (5)$$

The PCPF of the eight image processing methods is shown in Fig. 7. This graph shows a very significant difference between the PCPF of OpenCV and CamTest for the RGB and Sobel filter. The conversion from YUV to RGB consumes heavy processing and without any code-level optimization of good memory management, this conversion will consume a lot power. This is what happened in the CamTest. For heavy processing with large amount of data, a proper memory management and optimization is very important to extend the lifetime of the embedded hardware. The same reason is applied to the Sobel methods. Sobel edge detection needs to be convoluted twice before the output image shown. This process for sure will use a lot of power to execute since convolute method will use many looping. Without any optimization conducted on the code-level or on the algorithm itself, the convolution will consume a lot of time and power. One can reduce the power consumption of convolution by computing the convolution in the Fourier domain.

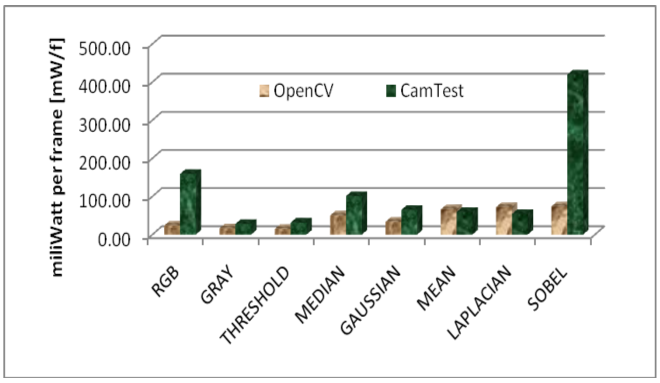


**Fig. 7. Power consumption per frame results between OpenCV library and CamTest for the eight image processing methods.**

Out of eight of the methods, OpenCV performs badly for Laplacian edge detection methods and mean blurring. It seems like there is no optimization is done with those methods. The PCPF ratio can also be computed by a little adjustment on formula (4). Instead of using the FPR, the PCPF is used. The result of PCPF ratio is shown in the Table III. From the table, we can see that OpenCV really consumes less power compared to that of CamTest. However, the two methods namely the mean and Laplacian consumes much power than the CamTest.

## V. Conclusions And Future Work

The majority of the image processing methods that using OpenCV library is higher performance than the self-made algorithm build in Android library. Based on the experimental results, we can conclude that OpenCV gives more attention to the efficiency rather than power consumption. For example, the Laplacian method in OpenCV consumes more energy than build in library. In the future, we would like to develop the techniques that can optimize power consumption in the video frame processing. This technique will be based on source code level optimization that would be able to solve the power consumption problem in OpenCV.

**Table III. The PCPF ratio of the eight image processing methods**

| Frame Processing | PCPF ratio |
|---|---|
| RGB | 0.85 |
| Grayscale | 0.41 |
| Threshold | 0.52 |
| Mean | -0.12 |
| Gaussian | 0.48 |
| Median | 0.50 |
| Laplacian | -0.34 |
| Sobel | 0.82 |
| Total Average | 0.39 |

## Acknowledgment

This paper is supported by Universiti Teknikal Malaysia Melaka under the contract PJP/2011/FKEKK(44B)/S00979 and PJP/2011/FKEKK (6B)/S00840.

## References

[1] OpenCV, Open source Computer Vision library. In http://opencv.willowgarage.com/wiki/, 2009.

[2] W. Meeussen, M. Wise, S. Glaser, S. Chitta, C. McGann, P. Mihelich, E. Marder-Eppstein, M. Muja, V. Eruhimov, T. Foote, J. Hsu, R. Rusu, B. Marthi, G. Bradski, K. Konolige, B. Gerkey, and E. Berger, “Autonomous Door Opening and Plugging In with a Personal Robot,” in Proceedings of the *IEEE International Conference on Robotics and Automation (ICRA),* Anchorage, Alaska, May 3-8 2010.

[3] Industry leaders announce open platform for mobile devices, November 2007.

[4] K. Owen, An Executive Summary of Research in Android & Integrated Development Environments, April 2011.

[5] N.K. Jha. “Low power system scheduling and synthesis”, *IEEE/ACM International Conference on Computer Aided Design,* pages 259 – 263, Nov. 2001.

[6] D. Ortiz and N. Santiago, “Impact of Source Code Optimizations on Power Consumption of Embedded Systems,” June 2008, pp. 133–136.

[7] R. Leupers. Code optimization techniques for embedded processors. Kluwer Academic Publishers, 2000.

[8] A. Sharma and C.P. Ravikumar. “Efficient implementation of ADPCM codec”, *Thirteenth International Conference on VLSI Design*, pages 456– 461, Jan. 2000.

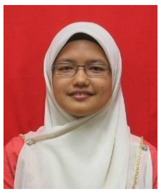
[9] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. “Accurate online power estimation and automatic battery behavior based power model generation for smartphones”, *in Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, CODES/ISSS* ’10, pages 105–114, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-905-3.

[10] T. Simunic, L. Benini, and G. de Micheli. “Energy-efficient design of battery-powered embedded systems”, *IEEE Transactions on Very Large Scale Integration Systems*, 9(1):15 – 28, Feb. 2001.

## AUTHOR’S PROFILE

**Ammar Anuar** received his B.S. degree in Electronic Engineering from Inha University, South Korea. He is currently studying his Master in Electronics Engineering & Computer Engineering at Universiti Teknikal Malaysia Melaka. His research interests are in Real-Time Image Processing for Embedded System, Android Application, Low Power Consumption Embedded System and Image Processing.

**Khairul Muzzammil Saipullah** received his B.S. and M.E. degree in Electronics Engineering from Inha University, South Korea. He is currently working as lecturer at Universiti Teknikal Malaysia Melaka. His research interests are in the area of Real-Time Image Processing for Embedded Device, Texture Image Analysis, Medical Image Processing, Computer Vision, and Embedded System.



**Nurul Atiqah Ismail** was born in 1987 in Pahang, Malaysia and did Bachelor of Electronics Engineering & Computer Engineering from Universiti Teknikal Malaysia Melaka in year 2011. She is currently pursuing her Master in Electronics Engineering at Universiti Teknikal Malaysia Melaka. Her research interests are in Image Processing and Embedded System.

**Yewguan Soo** Yewguan Soo received his B.E. and M.E. in Electrical Engineering from the University of Technology Malaysia in 2001 and 2003, respectively. In 2010, he completed his Ph.D from the University of Tokyo in Precision Engineering. He is presently working on as Senior Lecturer in the University of Tokyo. His research interests include myoelectric signal processing and embedded system design.