2012 IEEE 8th International Colloquium on Signal Processing and its Applications

Real-Time Video Processing Using Native Programming on Android Platform

Khairul Muzzammil bin Saipullah[1], Ammar Anuar[2], Nurul Atiqah binti Ismail[3], and Yewguan Soo[4]

Faculty of Electronic and Computer Engineering
Universiti Teknikal Malaysia Melaka (UTeM)
Melaka Malaysia
muzzammil@utem.edu.my[1], m021110015@student.utem.edu.my[2], m021110036@student.utem.edu.my[3] and soo@utem.edu.my[4]

***Abstract*—As the smartphone industry grows rapidly, smartphone applications need to be faster and real-time. For this purpose, most of the smartphone platforms run the program on the native language or compiler that can produce native code for hardware. However for the Android platform that based on the JAVA language, most of the software algorithm is running on JAVA that consumes more time to be compiled. In this paper the performance of native programming and high level programming using JAVA are compared with respect to video processing speed. Eight image processing methods are applied to each frame of the video captured from a smartphone that is running on an Android platform. The efficiencies of the two applications with difference programming language are compared by observing their frame processing rate. The experimental results show that out of the eight images processing methods, six methods that are executed using the native programming are faster than that of the JAVA programming with a total average ratio of 0.41. An application of the native programming for real-time object detection is also presented in this paper. The result shows that with native programming on Android platform, even a complicated object detection algorithm can be done in real-time.**

Keywords-Android; video processing; object detection; native programming.

I. INTRODUCTION

Real-time video is becoming increasingly popular with the proliferation of low-cost video cameras, camcorders and other facilities. Real-time video processing, however, is among the most demanding computation tasks [1]. Application- specific integrated circuits can deliver implementations optimal in speed and size, but they are often inflexible, expensive and time consuming to develop. Digital signal processing chips and general-purpose microprocessors often include hardware support for processing video or multimedia data, but there may be drawbacks. First, such hardware support may not cover a particular user- specific task: second, the system architecture and low-level software, such as interrupt control, can cause significant delay if not carefully optimized.

The Open Handset Alliance released the Google Android SDK on November 12, 2007 [2]. The conception of the Android platform is attracting more and more programmers in mobile computing fields. Android is a package of software for mobile devices, including an operating system, middleware and core applications. The Android SDK provides powerful tools and APIs necessary to develop applications on the Android platform using the Java programming language. Android platform is of open system architecture, with versatile development and debugging environment, but also supports a variety of scalable user experience, which has optimized graphics systems, rich media support and a very powerful browser. It enables reuse and replacement of components and an efficient database support and support various wireless communication means. It uses a Dalvik virtual machine heavily optimized for mobile devices.

Recently, Android has reached great success in mobile OS especially in smartphones and tablets. New versions of Android are being updated continuously. Due to these circumstances, Android developers introduce new application to satisfy the needs of the smartphone users. Libraries for native programming such as Open Graphics Library (OpenGL) and Open Computer Vision (OpenCv) are used for the development of the application. Android application developers tend to interface built-in devices such as camera, sensors, compass, Bluetooth, Wi-Fi and others into their applications. Application that uses camera usually involves image processing methods such as Gaussian, Median, Mean Laplacian, Sobel filter and others. Developers who have knowledge about image processing can write their own codes to apply those image processing method 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 into their work [3, 5].

In image processing, an open source library that is written in native language known as OpenCV reduces the complexity to apply image and video processing methods for developers. In this paper, we analyze the real-time video processing using native programming assisted with OpenCV library, and apply it on the android platform. Speed per frame test is conducted to measure the frame processing efficiency. All of the experiments are conducted on Samsung's GalaxyS smartphone that is powered by 1 GHz ARM Cortex-A8 processor running with Android 2.3 Gingerbread OS.

This document is organized as follows. In section 2, related work in video processing is discussed. Section 3 illustrates the methodology implemented and section 4 shows the results obtained and the analysis performed. In section 5, the application of real-time object detection on Android platform is presented. Finally, conclusions are presented in section 6.

II. ANDROID AND JAVA NATIVE INTERFACE

The Android architecture and its main components are shown in Fig.1. The architecture of Android system [4], similar to other operating systems, use a hierarchical structure. From the chart Android is divided into four layers; from the top to the lower level are the application layer, application framework layer, system layer and the Linux runtime core layer. All applications are done by the Java language. Application Framework provides java class library for Android applications, Android application developers developed applications, like developing those core applications, with full permission to access to the APIs provided by framework. Libraries layer was developed by C/C++ language, those libraries are packaged by Application Framework section. The bottom section is the Linux Kernel which is developed by c language, it provide the core system services, such as security, memory management, file system management, process management, network group, Driver Model and so on. It is also an abstract layer between the hardware and system software.

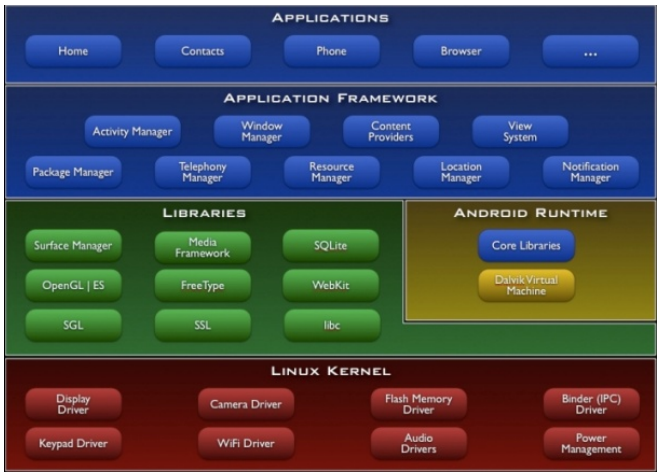


Figure 1. Android architecture

Java platforms are commonly deployed on top of a host environment. For example, the JAVA Runtime Environment (JRE) is a Sun product that supports the Java platform on existing operating systems such as Solaris and Windows. The Java platform offers a set of features that applications can rely on independent of the underlying host environment. JAVA Native Interface (JNI) is JAVA's mechanism for interfacing with native code [14]. It enables native code to have essentially the same functionality as Java code. Through JNI, native code can inspect, modify, and create JAVA objects, invoke methods, catch and throw exceptions, and so on. JNI lets JAVA code use code and code libraries written in other languages, such as C and C++. The Invocation API, which is part of JNI, can be used to embed a JAVA virtual machine (JVM) into native applications, thereby allowing programmers to call Java code from within native code. Java applications are written in the Java programming language, and compiled into a machine-independent binary class format. A class can be executed on any Java virtual machine implementation. Any implementation of the Java platform is guaranteed to support the Java programming language, virtual machine, and API.

The diagram of the JNI in the Android platform is shown in Fig. 2. The JNI is a powerful feature that allows you to take advantage of the Java platform, but still utilize code written in other languages. As a part of the Java virtual machine implementation, the JNI is a two-way interface that allows Java applications to invoke native code and vice versa. The JNI is designed to handle situations where one needs to combine Java applications with native code.

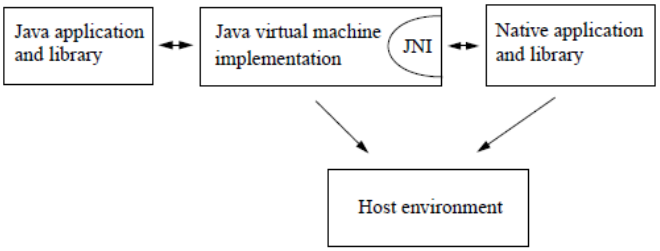


Figure 2. The role of JNI in the Android platform

III. REAL-TIME VIDEO PROCESSING IN ANDROID WITH NATIVE PROGRAMMING

order to compare the efficiency between native programming and JAVE programming on an embedded device namely the smartphone. Each image processing method is iterated 30 times and the average value is recorded for each experiment.

The description the eight image processing methods are shown in Table I. The input's 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

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

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are 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:

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

where, $f(x,y)$ is original frame and $g(x,y)$ is thresholded frame.

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

To remove the noise from the frame using median filter, each 3x3 window of the original frame is process by calculating the median value of the whole pixels in 3x3 windows. 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 Fig. 3(a). For mean filter, the frame will be convolved with 3x3 masks as shown in Fig. 3(b)

1/16	2/16	1/16	1/9	1/9	1/9	1	1	1
2/16	4/16	2/16	1/9	1/9	1/9	1	-8	1
1/16	2/16	1/16	1/9	1/9	1/9	1	1	1

Figure 3. (a) Gaussian mask, (b) Mean filter mask, (c) Laplacian mask.

For Laplacian, the frame will be convolved with the 3x3 mask as shown in Fig. 3(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. 4.

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

Figure 4. (a) The x-direction Sobel 3x3 mask, (b) The y-direction Sobel 3x3 mask.

Fig. 5and Fig. 6 shows the image processing methods using native programming and the JAVA programming (CamTest). Those images were captured in real-time using the Samsung Galaxy S camera at same position for the eight different methods as explained above. The output images were quite same for the gray scale, mean, Gaussian, median processing. One of the significant differences between the native programming and CamTest can be seen in the resulting image of Laplace edge detection. In native programming, the background region is darker whereas in the case of the CamTest, the edge at the background region is clearer. The reason is at the background regions, the Laplacian() function from the OpenCV that is running on the native programming 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.

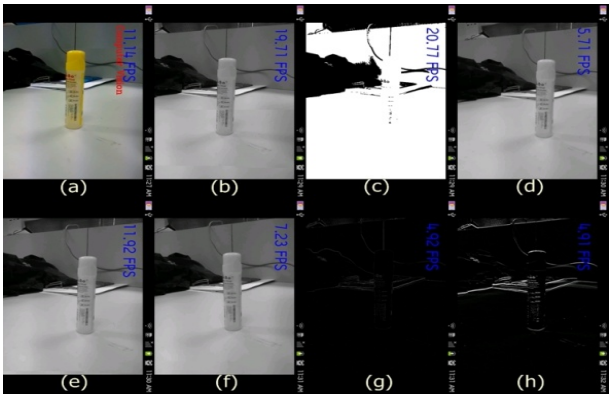


Figure 5. Native programming implementation. Those images were captured when it processes images in real-time video processing. (a) is RGB image, (b) is Greyscale image, (c)is Threshold image, (d) is Mean filter image, (e) is Gaussian image, (f) is Median filter image, (g) is Laplacian filter image, (h) is Sobel filter image.

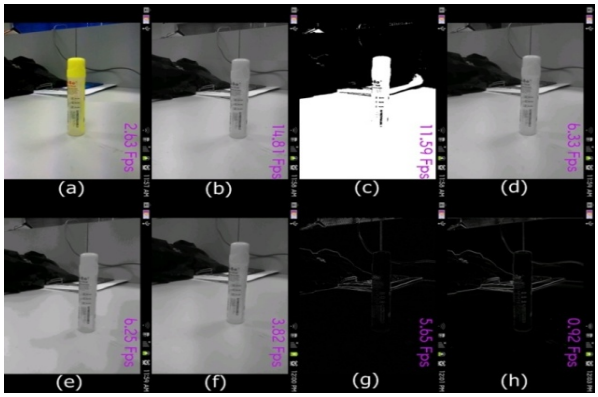


Figure 6. CamTest implementation. Those images were captured when it processes images in real-time video processing. (a) is RGB image, (b) is Greyscale image, (c) is Threshold image, (d) is Mean filter image, (e) is Gaussian image, (f) is Median filter image, (g) is Laplacian filter image, (h) is Sobel filter image

IV. EXPERIMENTAL RESULT

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

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

The unit for the FPR is frame per second (*fps*). It is the number of frame can be processed one second using the image processing algorithm. The higher value of the FPR, the more

efficient the method is. Fig. 7 shows the FPR of eight image processing methods using native programming and CamTest. As it can be seen, the chart shows a significant FPR differences between the native programming 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 the native programming. This may be caused by the similar algorithm applied for both mean and Laplacian methods in the native programming and CamTest.

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 there methods. However, in the case of native programming, 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 native programming compared to CamTest, the FPR ratio is calculated. The FPR ratio is calculated using the following formula:

$$FPR\ ratio = \frac{Native\ FPR - CamTest\ FPR}{Native\ FPR} \quad (4)$$

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

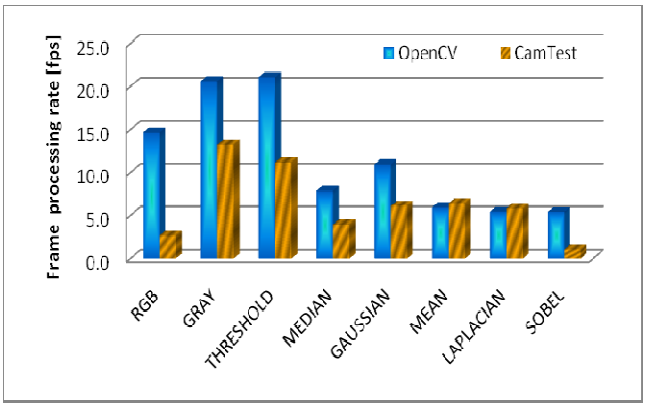


Figure 7. Frame Processing Rate using the native programming and CamTest for the eight image processing methods.

V. APPLICATION

A. Object detection algorithms

Object detection and recognition is becoming one of the major research areas in computer vision. Many applications are widely use especially in human-computer interaction, visual surveillance, robot navigation and many more. Object detection is use to detect the main point of object in an image. Generally, object detection is divided into three stages. In the first stage, representation of feature requiring for object recognition is examined based on local or global image information. Local image is for detecting object in certain part and global image is use to detect object in general image. Second stage is classification of image based on extracted features. The last stage is recognition of the new image based on learning machine which is performed with training images.

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

The first step of object recognition is feature extraction that is used to detect the interest point of the image. The Scale-Invariant Feature Transform (SIFT) method is use to detect feature local image. SIFT is invariant to image scale, noise and illumination. SIFT algorithm can be divide into four feature information stage which are scale-space extrema detection, keypoint localization, orientation assignment and keypoint descriptors. The scale-space extrema detection is used to detect the interest point and also known as keypoint. Then the image will convolve with Gaussian filter with different scales of image. Keypoint is taken from the maxima or minima of Difference of Gaussian (DoG) that is shown in Fig. 8. The second stage is keypoint localization. Among the keypoint candidates, the selections are made by using the comparison between each pixel. In orientation invariant, each pixel is assign on local image gradient directions. The last stage is keypoint descriptor which is used to find the location of the objects with different orientation and scale. The keypoint descriptor is invariant to the image location, scale and rotation. The SIFT utilizes Harris corner detector and have a good performance but not effective due to real-time of object recognition because expansion computation of the feature detection and keypoint descriptor [6, 11].

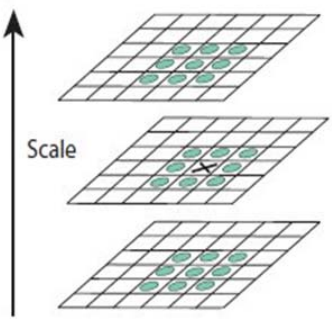


Figure 8. The process of extract DOG values.

For the faster feature matching, Speed up Robust Feature (SURF) algorithm has a similar performance with SIFT but is much faster than SIFT. SURF builds image pyramid and does filtering for each layer with Gaussian of increasing sigma by taking the difference between the layers. Since image pyramid are used in the multi-resolution image, the Gaussian of different scale is made using a constant filter size. SIFT looks for extrema in Difference of Gaussian filtered versions of an image. This computation is done for many image sizes, or octaves, and with a variety of different strength blurs, or scales. Simplified scale-space extreme detection in SURF algorithm speed up feature extraction speed, therefore it is being faster than SIFT. SURF algorithm is also has difficulties to produce real-time object recognition [6].

FAST corner detector is based on the corner information. It is widely used to track object in different corner. FAST corner detector is unlike SIFT and SURF where the FAST detector does not utilize the descriptor. Even though the FAST corner is 10 times faster than those of SIFT and SURF, it is able to get accurate interest point information. FAST corner detector is possible to recognize simple markers using template matching because affine transformations (changes in scale, rotation and position) are limited in such a case. FAST detector is less applicable for object detection and recognition because it reduces the time for feature extraction [7].

Good Features to Track (GFTT) is a feature detector that is based on the Harris corner detector. The main improvement is that it finds corners that are good to track under affine image transformations. Maximally Stable Extremal Regions (MSER) is used as a method of blob detection in images. This method is use to find correspondence between two image with different viewpoint. MSER is applied with binary image. All pixels inside MSER have 'extremal' where it refers to the higher or lower intensity than all the pixels on its outer boundary. Meanwhile, MSER regions are 'maximal stable' in the threshold selection process [8][13]. Oriented FAST and rotated BRIEF (ORB) is very fast binary descriptor based on BRIEF descriptor. ORB is combination of FAST detector and BRIEF descriptor. BRIEF is a feature descriptor that uses simple binary tests in a smoothed image patch. It is similar to SIFT regarding to invariant to lighting, blur and distortion but the weakness is very sensitive to the rotation [9][12].

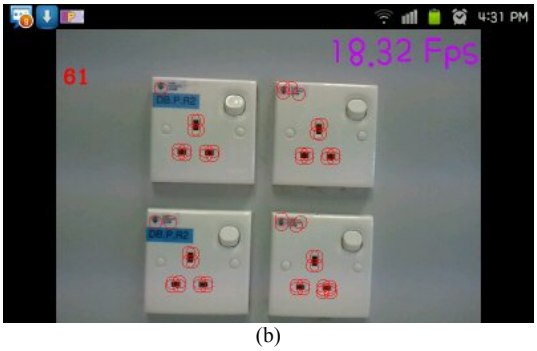
Center Surrounded Extrema (CenSurE) uses polygon, hexagon and octagon filters as more computable alterative to circle filter. First, CenSurE computes all the location and scales to find the local extrema in a neighborhood by simplified center-surround filter. Then Harris detector is use to eliminate the entire weak corner. CenSurE applies simple approximations where it uses bi-level center surround filter by multiply the image value to 1 and -2. Fig. 9 shows the bi-level Laplacian of Gaussian and other examples of approximations that are use to conjugate with integral images [10].

Figure 9. Center-Surround bi-level filters approximating the Laplacian.

B. *Performance evaluation*

To measure the speed per frame (fps) test, the processing time of each object detection method for one frame is recorded. From this information, the number of frame that can be processed in one second can be calculated. Each object detection method is executed on the video captured by the camera. 10 continuous frames are selected and the average processing speed for one frame is used to measure the fps on the two different objects. The higher value of the fps is the higher speed of the method to process the frame. Two objects that are applied in this test are the glue and the power socket that are shown in Fig. 10.

(a)



(b)

Figure 10. Object is used in the experiment. (a) Glue image, (b) Power socket.

Fig. 11 shows the performance of speed per frame test for seven object detection methods. As shown in the graph, the FAST algorithm achieves the highest fps value while the SIFT algorithm achieves the lower fps than the other algorithms. We also can see that FAST is 10 times faster than SIFT and

SURF. The minimum fps rate for real-time video is 15 fps. This shows that FAST achieves the optimum real-time video performance while executing the object detection algorithm.

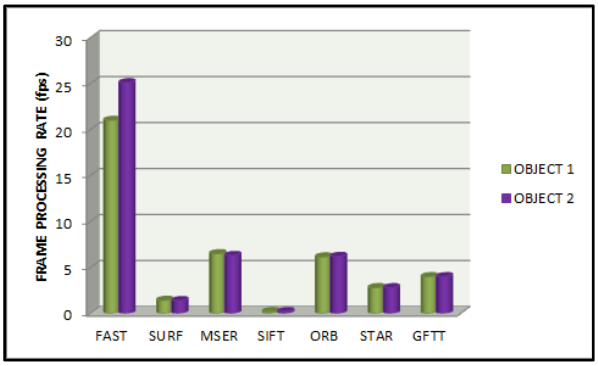


Figure 11. Frame Processing Rate for the seven objects detection methods

VI. CONCLUSIONS AND FUTURE WORK

Majority of the image processing methods that using native programming assisted with OpenCV library has high performance than the self-made algorithm built in Android library using JAVA language that we called as CamTest. Based on the experiment results, we can conclude that native programming give more attention to the efficiency compared to the CamTest. From the object detection application, even though not all of the algorithms achieved the 15 fps threshold, one of them manage to extract the keypoint fast enough even the human eyes could not see the delay. This shows the superior of the native programming compared to the JAVA programming. For future work, we would like to enhance the performance of object detection algorithm of FAST so that it will not only process fast but also produce high accuracy detection. We also would like to consider a low power consumption technique in the algorithm since the computer vision and image processing algorithm consumes a lot of power and this will burden the hardware that is only powered by a battery.

ACKNOWLEDGEMENT

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

REFERENCES

[1] Athanas, P. M. and A. L. Abbott (1995). "Real-time image processing on a custom computing platform." Computer 28(2): 16-25.

[2] Open Hanset Alliance, http://www.openhandsetalliance.com/

[3] 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.

[4] Xianhua, S., D. Zhenjun, et al. (2009). Research on Mobile Location Service Design Based on Android. Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on.

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

[1] Paul Viola, Michael Jones, Robust Real-time Object Detection, Second International Workshop on Statistical and Computational Theories of Vision, July 2001.

[2] KanghunJeong, Hyeonjoon Moon, Object Detection using FAST Corner Detector based on Smartphone Platforms, *Computers, Networks, Systems and Industrial Engineering (CNSI),* May 2011.

[3] Donoser, M.; Bischof, H. "Efficient Maximally Stable Extremal Region (MSER) Tracking," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* pp.553-560, June 2006.

[4] Ethan Rublee Vincent Rabaud Kurt Konolige Gary Bradski, ORB: an efficient alternative to SIFT or SURF.

[5] Agrawal, M., K. Konolige, et al."CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching,"*Computer Vision – ECCV 2008*, Springer Berlin / Heidelberg.vol.5305, pp.102-115, 2008.

[6] David G. Lowe, "Object Recognition from Local Scale-Invariant Features,"The Proceedings of the Seventh IEEE International Conference on Computer Vision,pp. 1150-1157, 1999.

[7] Clemens Arth, Christian Leistner, Horst Bischof, "Robust Local Features and their Application in Self-Calibration and Object Recognition on Embedded Systems,"*Computer Vision and Pattern Recognition,2007. IEEE Conference,* June 2007.

[8] Calonder, M., V. Lepetit, et al. "BRIEF: Binary Robust Independent Elementary Features," *Computer Vision – ECCV 2010*, Springer Berlin / Heidelberg. Vol. 6314, pp.778-792, 2010.

[14] Liang, S . Java Native Interface: Programmer's Guide and Reference. Addison-Wesley, 1999