A Real-time DSP-based Hand Gesture Recognition System

Xuan-Thuan NGUYEN, Lam-Hoai-Phong NGUYEN, Trong-Tu BUI, and Huu-Thuan HUYNH
Faculty of Electronics and Telecommunications
University of Science, Ho Chi Minh City
227 Nguyen Van Cu St., Dist.5, Ho Chi Minh City, Vietnam
{nxthuan, bttu, hhthuan}@fetel.hcmus.edu.vn

Abstract— **In this paper, a real-time hand gesture recognition system implemented on a TMS320DM642 Digital Signal Processor (DSP) of Texas Instruments (TI) is proposed. The highly effective system achieved by combining specialist algorithms and advanced DSP optimization techniques. The proposed algorithms are composed of face detection and hand gesture recognition together with skin segmentation and hand tracking procedures. Cascade classifiers using AdaBoost algorithm are applied in face detection and hand recognition to accelerate the entire system. In addition, many DSP optimization levels such as floating-point numbers to fixed-point numbers conversion, software pipelining etc. are also exploited. The results show that the proposed design achieves up to 50 640x480 16-bit YCbCr frames per second (fps), which is capable of many real-time applications.**

I. INTRODUCTION

Most human-computer interactions (HCI), in general, are based on mechanical devices such as panel, keyboard, mouse, or remote controller. In recent years, there has been a growing interest in the use of hand gestures recognition [1] as the inputs of HCI. Some applications may include hand-free interaction with web browser [2], indoor robot controlling [3], hearing-impaired or deaf people teaching [4] etc.

Hand gesture recognition, in fact, is a complex problem that has been solved by many different ways. S.H. Lee et al. [5] proposed an FPGA-based hand gesture recognition system for controlling the hardware appliance in real-time based on bounding box and Center-Of-Mass computation. However, a red color glove must be used in this system to distinguish the human hand from the environment and its distance to the camera must be unchanging. J. Zhang et al. [6] presented an adaptive HSI complexion model for hand segmentation instead of utilizing uniquely color glove. Although the system can work properly under a variety of lighting conditions, it still cannot meet the real-time requirements due to a lot of intensive computational processes. Viola and Jones [7] proposed an object detection framework in 2001, which utilizes Haar-like features as classifiers and cascade of boosted classifiers. Since then, several Viola Jones-based research on hand gesture recognition has been proposed. For example, an accurate and robust hand gesture recognition system was discussed by Y.Fang et al. [8]. However, the results of approximately 10 fps for 320x240 image indicate the limitation on real-time applications.

Digital signal processor (DSP) and their tools, recently, are powerful enough to develop complex systems effectively, especially those of Texas Instruments (TI). Many of TI's video/image DSPs support advanced Very-Long-Instruction-Word (VLIW), powerful L1/L2 cache architecture, enhanced Direct Memory Access (EDMA) etc. which are very useful for high-speed data transfer between hardware blocks inside the system as well as the low-latency execution time in each block. Moreover, with the help of Code Composer Studio (CCS) [9], the TI's integrated development environment (IDE), many extensive-computation algorithms (i.e. motion detection, face recognition etc.) could be developed and debugged efficiently. It is shown that these DSPs are very capable of real applications.

In this paper, we present a real-time hand gesture recognition system implemented on a TMS320DM642 DSP [10],a TI's platform. Viola Jones framework utilizing Haar-like features and cascade classifiers is the basis of our system. Furthermore, face detection algorithm, skin segmentation algorithm (i.e. histogram back-projection method [11]), and color-based moving object tracking algorithm (i.e. Continuously Adaptive Mean shift - CAMshift [12]) are exploited to improve the whole system performance. Each 640x480 YCbCr422 frame is captured consecutively by a camera as the system input and the recognition results are displayed on a VGA monitor to verify.

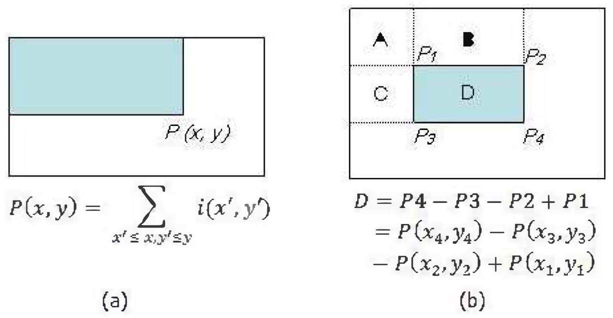
The remainder of this paper is organized as follows. Section 2 describes the hand gesture recognition algorithms. Section 3 illustrates the system architecture implemented in a DSP development kit and the DSP-based optimization methods [13]. Section 4 presents the performance of the proposed system compared with the others. Section 5, finally, gives the conclusion.

II. ALGORITHMS

A. Related Algorithms

The terms introduced in this paper are discussed in the following section and are used to develop a unifying framework for the proposed hand gesture recognition algorithms.

1) Integral Image: The integral image, as shown in Fig.1, is defined as the summation of the pixel values of the original image. The value at any location *P(x, y)* of the integral image is the sum of the image's pixels above and to the left of location *P(x, y)*.



P (x, y)
$P(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y')$
(a)
A
B
C
D
P1
P2
P3
P4
$D = P4 - P3 - P2 + P1 = P(x_4,y_4) - P(x_3,y_3) - P(x_2,y_2) + P(x_1,y_1)$
(b)

Fig. 1. Integral image generation (a) The value of the integral image *P(x, y)* at point *(x, y)* is the sum of all the pixels above and to the left, *i(x, y)* is the original image. (b) The sum of the pixels within rectangle *D* can be computed with four array references. The value of the integral image at location *P1* is the sum of the pixels in rectangle A. The value at location *P2* is A+B, at location *P3* is *A+C*, and at location *P4* is *A+B+C+D*. Therefore, the sum within D can be computed as *P4-P3-P2+P1*.

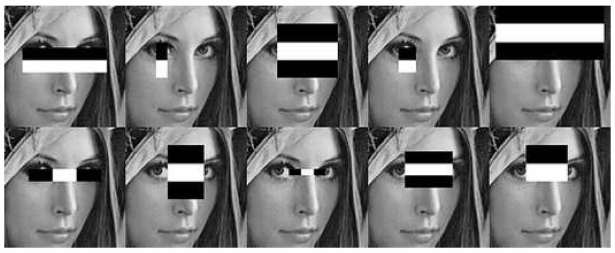


Fig. 2. An example of Haar-like features. In order to get the Haar-like feature value, areas of white and black regions are multiplied by their respective weights and then summed.

2) Haar-like Features: Haar-like features are consisted either two, three, or four rectangles. Object candidates are scanned and searched for Haar-like features of the current stage. A machine learning algorithm from Adaboost is employed to generate the weight and size of each feature and the features themselves. Some Haar-like features for face detection are illustrated in Fig.2, where each feature type can indicate the existence (or absence) of certain characteristics in the image. Each Haar-like feature is computed by taking the area of each rectangle, multiplying each by their corresponding weights, and then summing the results. Note that the integral image is used to find area of each rectangle easily and quickly.

3) Cascade Classifier based on Haar-like Features: The Viola Jones-based object detection algorithm eliminates object candidates quickly using a cascade of stages. A stage is composed of several Haar-like feature classifiers. A stage comparator, then, sums all the Haar-like feature classifier results in a stage and compares this summation with a stage threshold which is obtained by the AdaBoost algorithm. Depending on the parameters of the training data individual stages can have a varying number of Haar-like features. The cascade eliminates candidates by making stricter requirements in each stage with later stages being much more difficult for a candidate to pass. Fig.3 shows an example of a N-stage cascade classifier of face detection algorithm. A face is detected if a candidate passes all stages.

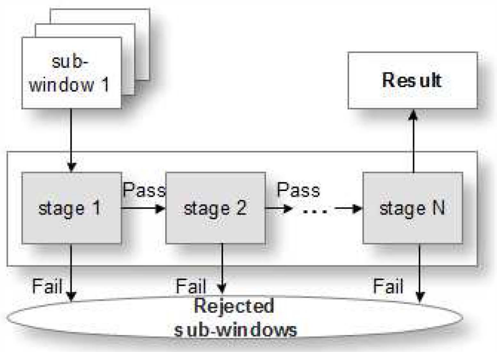


Fig. 3. A N-stage cascade classifier.

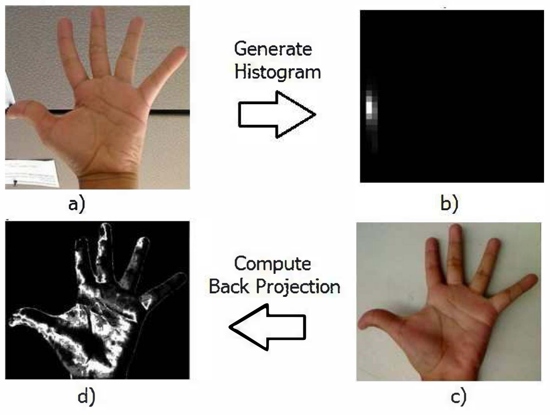


Fig. 4. An example of HBP [11] (a) The given image with a hand as target (b) A model histogram which represents a sample of skin tonality (c) The test image (d) The back-projection image; the values stored in this image represent the probability that a pixel in test image belongs to a skin area, based on the model histogram above. The brighter areas are more probable to be skin area, whereas the darker areas have less probability.

4) Histogram Back Projection (HBP): In real environment, multiple light sources impinge significantly on the human skin. In other words, for robust skin detection, a dynamic skin color model that can cope with the changes must be employed. Histogram Back Projection (HBP), a technique used for creating a probability image (back-projection image) of the region of interest (ROI), as a result, is proposed. An example of this method is depicted in Fig.4. Suppose we create a histogram of a target in an image. If we back-project this histogram onto the image, it will create an image that shows the probability of a pixel belonging to the target.

5) Continuously Adaptive Mean shift (CAMshift): Mean shift is an algorithm that iteratively shifts a data point to the average of data points in its neighborhood, which can be used for visual tracking. However, in most real-time applications, color distribution varies continuously due to the rotation in depth, whereas Mean shift failed to tackle the problem of dynamic-color distribution. CAMshift, a successor of Mean shift, therefore, is proposed instead. In fact, CAMshift can handle dynamically changing color distribution and adapt Mean shift window search size and compute color distri-

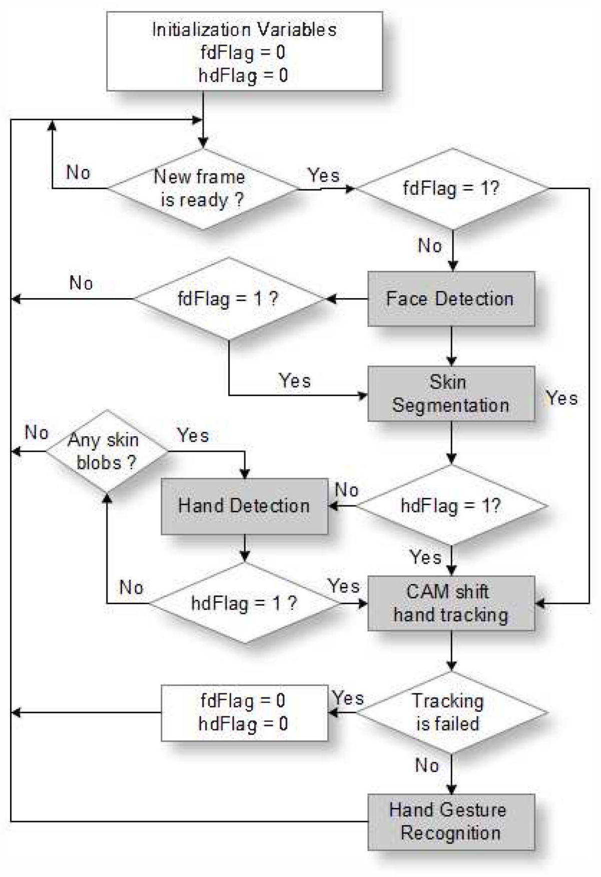


Fig. 5. The flow diagram of the proposed hand gesture recognition algorithm.

bution in search window efficiently. A four-step CAMshift-based hand tracking example [11] is depicted as follows.

- Create a color histogram to represent the hand.
- Calculate a hand probability for each pixel in the incoming video frames using HBP.
- Shift the location of the hand rectangle in each video frame. The new location is found by starting at the previous location and computing the center of gravity of the hand-probability values within a rectangle.
- Calculate the size and angle that are the best fit to the hand-probability pixels inside the new rectangle location.

B. Proposed Hand Gesture Recognition Algorithm

The flow diagram of the proposed hand gesture recognition algorithm is illustrated in Fig.5. Two variables, *fdFlag* and *hdFlag*, are used to verify whether a face and hand, respectively, have been located or not. In the beginning, *fdFlag*=*hdFlag*=0, which denotes face and hand are not identified yet. If a face or hand is detected, *fdFlag* or *hdFlag* is asserted, respectively, and this value is remained until the system is restarted or hand tracking is unsuccessful.

After the initialization, the Viola Jones-based face detection algorithm scans the entire image with certain sub-windows and denotes each respective section a face candidate. The algorithm looks for specific Haar-like features of a human face. The integral images are utilized to process Haar-like features of a face candidate in constant time. A 22-stage cascade is used to eliminate non-face candidates quickly. Each stage consists of many different Haar-like features and each feature is classified by a Haar-like feature classifier. In total, 2135 Haar-like features are employed in the face detection step. An output is generated by the Haar-like feature classifiers and then is provided to the stage comparator. The stage comparator sums the outputs of the Haar-like feature classifiers and compares this value with a stage threshold to determine if the stage should be passed. If the candidate fails in one of 22 stages, the face detection step will be started again with the next frame. Otherwise, the candidate is concluded to be a face and *fdFlag* is asserted.

Subsequently, each pixel in face region is reduced from 16 bits to 12 bits. A RGB-color histogram of this quantized ROI, then, is created. In the following frames, by using this histogram and HBP technique, the skin-like pixels can be detected rapidly. The output of this step is a binary matrix, called search map or skin map, which contains one or more skin blobs. The search map, besides, is further processed by applying dilation and followed by erosion using a 3x3 square window [15]. This sub-step aims to reduce noise and scratch-like artifacts in the search map. Erosion removes object smaller than the structuring element while dilation connects areas separated by spaces smaller than the structuring element.

Afterward, the Viola Jones framework is applied to those skin blobs once again to detect and recognize hand gestures. In total, 635, 599, 437, and 593 Haar-like features are utilized for four hand gestures, in chronological order as illustrated in Fig.9. Any skin blob will be considered as a hand in case it passes one of four 20-stage cascade classifiers. *hdFlag*, simultaneously, will be asserted. Otherwise, this ROI will be removed from the search map and the next one is processed. If there is no remaining skin blobs, this step will be halted.

Finally, CAMshift is applied to track the hand so that the system does not need to detect face and hand repeatedly. In other words, the system only handle the certain ROIs instead of scanning the entire image. Hand tracking is reactivated as soon as either the hand is disappeared or the area of ROI is exceeded the given threshold. Both *fdFlag* and *hdFlag*, simultaneously, are de-asserted so that the system can execute face detection and hand detection step.

III. IMPLEMENTATION

A. TMS320DM642 Features

The TMS320C64x DSPs including the TMS320DM642 device are the highest-performance fixed-point DSP generation in the TMS320C6000 DSP platform. The TMS320DM642 (DM642) device is based on the second-generation high-performance, advanced VelociTI VLIW architecture (VelociTI.2) developed by Texas Instruments (TI). At a clock rate of 720 MHZ, the DM642 can perform up to 5760 million instructions per second (MIPS).

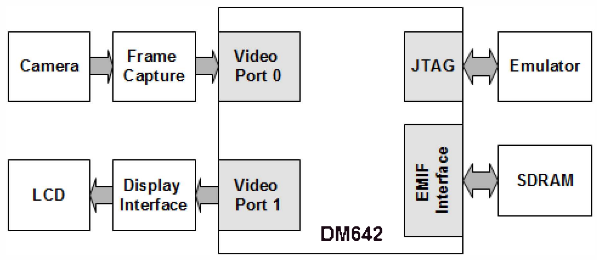


Fig. 6. Block diagram of the DM642-based hand gesture recognition system.

Furthermore, the C64x DSP core processor has 64 general-purpose registers of 32-bit word length and eight highly independent functional units two multipliers for a 32-bit result and six arithmetic logic units (ALUs) with VelociTI.2 extensions. The VelociTI.2 extensions in the eight functional units include new instructions to accelerate the performance in video and imaging applications and extend the parallelism of the VelociTI architecture.

The DM642 uses two-level cache-based architecture for program and data and possesses a powerful and diverse set of peripherals. The Level 1 program cache (L1P) and Level 1 data cache are a 128-Kbit direct mapped cache and a 128-Kbit 2-way set-associative cache, respectively. The Level 2 memory/cache (L2) consists of an 2-Mbit memory space that is shared between program and data space. The peripheral set is comprises of a 10/100 Mbps Ethernet MAC (EMAC), a 64-bit external memory interface (EMIFA), three configurable video ports (VP0, VP1, and VP2) etc. Those video ports support multiple resolutions, multiple video standards, and either capture or display mode.

For this reason, the DM642 is very fit for the video and imaging applications due to its powerful capability of data processing and interfaces.

B. DM642-based Hand Gesture Recognition System

The general block digram of DM642-based hand gesture recognition system is depicted in Fig.6. This processing unit consists of the DM642, the SDRAM memory, two video ports (VP0 and VP2), and the JTAG emulator. The SDRAM is configured as 64Mx64-bit memory and used to store the intermediate data as well as the feature sets. The VP0 and VP2 are defined as capture and display port, respectively. The acquired frames are decoded by TVP51xx interface while the results are displayed on a VGA by the SAA7121 one.

The block diagram of the proposed hand gesture recognition core is described in Fig.7. Many components have been developed to achieve high performance system including YCbCr to RGB Converter (*M1*), Scale to 320x240 (*M2*), Face Detection (*M3*), Skin Segmentation (*M4*), Hand Detection (*M5*), Hand Tracking (*M6*), Hand Gesture Recognition (*M7*), Post Processing (*M8*), and VGA Display (*M9*).

In the beginning, each 640x480 YCbCr422 frame is captured in turn from the VP0 port, which is driven by FVID library and configured by VPORTCAP_Params function. This frame, then, is converted into RGB565 by *M1* with the

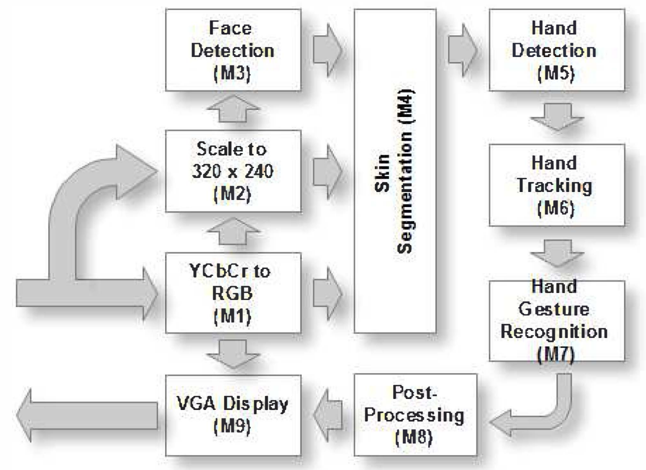


Fig. 7. Block diagram of a hand gesture recognition core.

formula illustrated in Eq.1.

$$R = 1.1644Y + 1.596Cr - 222.92 \quad (1a)$$
$$G = 1.1644Y - 0.3918Cb - 0.813Cr + 135.58 \quad (1b)$$
$$B = 1.1644Y + 2.0122Cb - 276.83 \quad (1c)$$

The Y component, simultaneously, is scaled down to 320x240 by *M2* and moved to the next stage. The Gaussian pyramid [14] is employed to reduce the image resolution. *M3*, then, locates the face in the 320x240 grayscale image by using Viola Jones framework. If a face is detected, its RGB-color histogram is calculated and back projection technique is applied in the following frames to function a skin map, or search map. Besides, this search map, or *M4* output, is further processed by employing dilation and erosion to reduce such noises.

The next stage, *M5*, decides whether the ROI of skin is a hand or not. The correct ROI is passed to next stage while the false ones are removed from the search map. Hand tracking using CAMshift is applied in *M6* so that the system is no need to scan the entirely following frames. It is just reactivated as soon as either the hand is disappeared from screen or the ROI area is exceeded the given threshold. *M7* decides whether the input hand matches to one of four given hand gestures or not. Both hand detection and hand recognition utilize Haar-like features and cascade classifiers to achieve the performance.

When a hand region is identified in an search map, a check for overlap with previously found hand regions is done by *M8*. If no overlap is found, the corresponding object window location is added to the hand regions list and the object map bits covered by the object window are cleared. Lastly, *M9* displays the current hand gesture matching with the available database on a VGA monitor for verification. FVID library drives the video output port, VP2, and the parameters referred to video displaying are configured in VPORTDIS_Params function.

C. Optimizations

In order to improve the system performance, many DSP-based optimizations are deployed as follows.

1) Compiler option selection: The Code Composer Studio (CCS) integrates abundant compiler options to control the operation of the compiler. For instance, -o3 option is always strongly recommended because it represents the highest level of available optimization. In fact, when -o3 option is enabled, various loop optimizations are performed (i.e. software pipelining, single instruction multiple data (SIMD) etc.), which reduces the execution time.

2) Floating-point number to fixed-point number conversion (FFC): The DM642 is a fixed-point DSP; hence, all floating-point arithmetic must be converted into fixed-point ones. For example, instead of computing 0.3918*Cb, (401*Cb)/1024 or ((0x191*Cb) SHL 0xa) is calculated. The test data show that float-type data should be converted from 24- to 32-bit integer to guarantee the precision. The experiments prove that FFC code is about three to eight times faster than the floating-point one.

3) Memory dependencies: In order to maximize the efficiency of the code, the compiler schedules as many instructions as possible in parallel. *restrict* keyword, thus, is utilized to help the compiler determine the relationships, or dependencies, between instructions so that the compiler can schedule instructions in parallel effectively.

4) Intrinsics: The CCS provides intrinsics, special functions which are optimized internally in DSP and are reduced to single assembly instructions. All instructions that are not easily expressed in C code are supported as intrinsics. For example, the *_sadd()* intrinsic is applied for the saturated addition instead of writing a multicycle function C code. It is recommended to utilize intrincis to optimize the C code quickly.

5) Software pipelining: The proposed algorithms are composed of many loop procedures which take lots of DM642's useful time; for this reason, software pipelining techniques are utilized to increase the performance. This technique aims to schedule instructions from a loop so that multiple iterations of the loop execute in parallel. According to our experiments, the system speed is improved up to three times by using this optimization.

6) Linear assembly: Some functions require to execute many times while some may not make full use of the two separate sets of registers and operation units. In this case, liner assembly language is used instead of C one for optimization. For instance, YCbCr422 to RGB565 Converter is written in assembly to condense the code size and decrease the run time.

IV. EXPERIMENTAL RESULTS

The proposed algorithms are first implemented in a personal computer (PC) with the help of Visual Studio C and Intel's OpenCV library. After successful PC-based testing and debugging, Code Composer Studio (CCS) v3.3 is utilized for real-time verification. The target platform is a TMS320DM642 development board of Texas Instruments (TI) as illustrated in Fig.8.

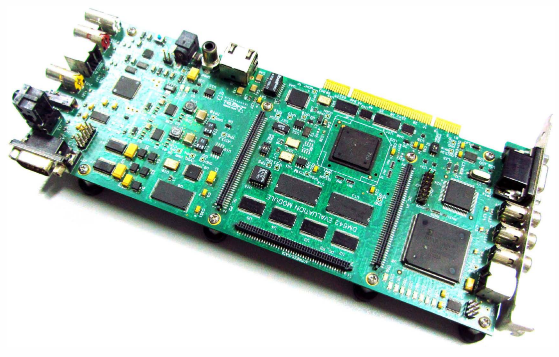


Fig. 8. The DM642 development board.

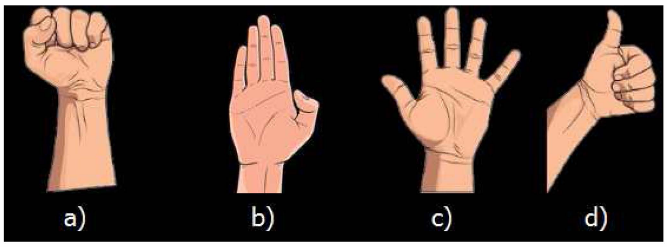


Fig. 9. Definitions of four hand gestures (a) *LCLICK* denotes an action to make a computer do something by pressing a left button on the mouse (b) *DRAG* denotes an action of moving of something across a computer screen using the mouse (c) *FREE* denotes an action to move the pointer freely on a computer screen (d) *RCLICK* denotes an action to make a computer do something by pressing a right button on the mouse.

Four hand gestures are used in our experiments including *LCLICK*, *DRAG*, *FREE*, and *RCLICK* as defined in Fig.9. They can be considered as some basic tasks of a PC. For example, to select any object on a screen, at first, *FREE* is used to move a pointer to the right position. *DRAG*, then, can be used for holding and dragging the object to a different location on a screen. Besides, *LCLICK* and *RCLICK* are used as left and right click on a given item, respectively.

In order to assess the performance, a large number of experiments are conducted under different light conditions, various distances between a hand and camera, and different gesture sizes performed by alternative users. The proposed system proves the robustness due to its highly correct recognition percentage. An example of the recognition results is shown in Fig.10.

Table I compares the system speed between the DM642 with PC. The DSP-based optimized design achieves about 50 fps, which is two times and five times faster than that implemented on a PC and non-optimized DSP, respectively. The execution time, undoubtedly, is quite appropriate to real-time applications. Furthermore, the proposed system provides a fairly easy expansion ability. In fact, by supplying additional feature sets in off-chip memories (i.e. SDRAM), many more hand gestures can be recognized.

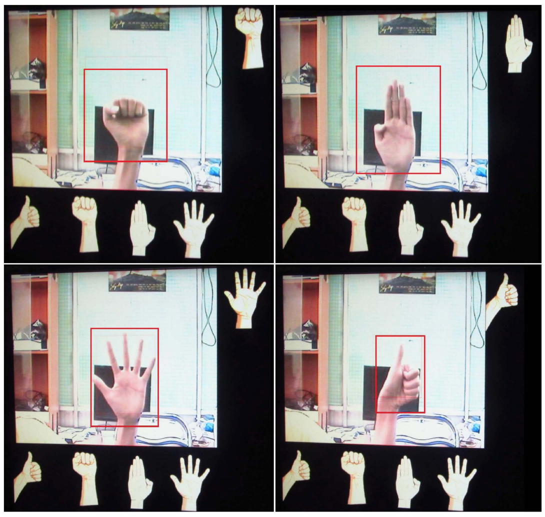


Fig. 10. An example of hand gesture recognition results. The top-left quarter depicts the experimental result of *LCLICK*. The system searchs for the matching hand gesture and displays the corresponding one on a screen. A set of given hand gesture and the recognized result are shown at the bottom and right of the top-left corner, respectively.

TABLE I
THE COMPARISON OF THE PROPOSED SYSTEM IN A DM642 AND PC.

PLATFORM	CPU USAGE (%)	FRAMES PER SECOND (FPS)
Intel Core i3-370M 2.4 GHz, 4 GB RAM, Windows 7 x64	30	22
TMS320DM642 (non-optimized)	100	10
TMS320DM642 (optimized)	80	50

V. CONCLUSIONS

This paper presents the algorithms and architectures of a real-time high-reliability hand gesture recognition system implemented on a TMS320DM642 DSP of Texas Instruments (TI). The Viola Jones-based face detection and hand recognition are combined with HBP-based skin segmentation and CAMshift-based hand tracking procedures to achieve the high performance. Furthermore, the optimization of the proposed architecture which fully exploits such the DSP hardware is discussed in detail to accelerate the whole system. The results of 50 fps at the resolution of 640x480 show that the proposed design is thoroughly capable of many real-time applications. Last but not least, the design can recognize additional hand gestures easily by supplying neccessary feature parameters to off-chip memories.

REFERENCES

[1] V.I. Pavlovic, R. Sharma, and T.S Huang, ”Visual interpretation of hand gestures for human-computer interaction: A Review,” *Proc. Int. IEEE Pattern Analysis and Machine Intelligence*, pp. 677 - 695, 1997.

[2] Y. Shi, R. Taib, and S. Lichman, ”GestureCam: A Smart Camera for Gesture Recognition and Gesture-Controlled Web Navigation,” *Proc. Int. Conf. Control, Automation, Robotics and Vision (ICARCV)*, pp. 1 - 6, Dec. 2006.

[3] T. Shiraishi, A. Ishitani, M. Ito, S. Karungaru, and M. Fukumi, ”Operation Improvement of Indoor Robot by Gesture Recognition,” *Proc. 4th Int. Conf. Modeling, Simulation and Applied Optimization (ICMSAO)*, pp. 1 - 4, Apr. 2011.

[4] Z. Vmossy, A. Tth, and B. Benedek, ”Virtual Hand Hand Gesture Recognition System,” *Proc. Int. Symp. Intelligent Systems and Informatics (SISY)*, pp. 97 - 102, Aug. 2007.

[5] S.H. Lee, S.N. Cheong, C.P. Ooi, and W.H. Siew, ”Real Time FPGA Implementation of Hand Gesture Recognizer System,” *Proc. 15th WSEAS Int. Conf. Computers*, pp. 217 - 222, 2011.

[6] J. Zhang, H. Lin, and M. Zhao, ”A Fast Algorithm for Hand Gesture Recognition Using Relief,” *Proc. 6th Int. Conf. Fuzzy Systems and Knowledge Discovery*, pp. 8 - 12, 2009.

[7] P. Viola and M. Jones, ”Rapid Object Detection using a Boosted Cascade of Simple Features,” *Proc. Int. Conf. Computer vision and Pattern Recognition*, 2001.

[8] Y. Fang, K. Wang, J. Cheng, and H. Lu, ”A Real-Time Hand Gesture Recognition Method,” *Proc. IEEE Int. Conf. Multimedia and Expo*, pp. 995 - 998, 2007.

[9] Texas Instruments, Code Composer Studio, `http://www.ti.com/tool/ccstudio`, Accessed Day: Jan. 2012.

[10] Texas Instruments, ”SPRU615, TMS320DM642 Technical Overview: DSP Video and Imaging Digital Applications,” `http://www.ti.com/lit/ug/spru615/spru615.pdf`, Accessed Day: Jan. 2012.

[11] OpenCV Documents, `http://docs.opencv.org/` Accessed Day: Apr. 2012.

[12] John G. Allen, Richard Y. D. Xu, Jesse S. Jin, ”Object tracking using CamShift algorithm and multiple quantized feature spaces,” *Proc. Visual Information Processing Workshop*, pp. 3 - 7, Australia, 2004.

[13] Texas Instruments, ”SPRU198i, TMS320C6000 programmer’s guide:,” `http://www-s.ti.com/sc/techlit/SPRU198`, Accessed Day: Jan. 2012.

[14] C. H. Anderson , J. R. Bergen , P. J. Burt , J. M. Ogden, ”Pyramid Methods in Image Processing,” `http://citeseerx.ist.psu.edu`, Accessed Day: Mar. 2012.

[15] Dwayne Phillips, ”Image Processing in C, Second Edition,” pp. 153 - 182, 2000.