J Electr Eng Technol.2017; 12(5): 2067-2078
http://doi.org/10.5370/JEET.2017.12.5.2067

ISSN(Print) 1975-0102
ISSN(Online) 2093-7423

A Real-time Face Tracking Algorithm using Improved CamShift with Depth Information

Jun-Hwan Lee*, Hyun-jo Jung* and Jisang Yoo†

Abstract – In this paper, a new face tracking algorithm is proposed. The CamShift (Continuously adaptive mean SHIFT) algorithm shows unstable tracking when there exist objects with similar color to that of face in the background. This drawback of the CamShift is resolved by the proposed algorithm using Kinect's pixel-by-pixel depth information and the skin detection method to extract candidate skin regions in HSV color space. Additionally, even when the target face is disappeared, or occluded, the proposed algorithm makes it robust to this occlusion by the feature point matching. Through experimental results, it is shown that the proposed algorithm is superior in tracking performance to that of existing TLD (Tracking-Learning-Detection) algorithm, and offers faster processing speed. Also, it overcomes all the existing shortfalls of CamShift with almost comparable processing time.

Keywords: Face tracking, Face-TLD, Haar-Feature, CamShift, Kinect

1. Introduction

Real-time face tracking is a topic that is continuously being researched for many purposes in image processing area. However, it still remains a challenge to achieve high accuracy in a dynamic environment [1-5]. Even a real-time face tracking is not an easy problem, especially as many factors are involved. These factors can include skin color, motion information, and various changes in features. Even when a system accurately detects a facial region, it still consumes a long time in tracking, or sometimes even fails to track the detected face.

In these cases, HCI (human computer interaction) system, or surveillance system that requires real-time face tracking, cannot properly operate. The main key to successful real-time face tracking is the amount of computation. No matter how accurate the real-time face-tracker is, complex computations mean that it is bound to be slow and has only limited areas of applications for tracking. Even Face-TLD [6, 7], an algorithm renowned for excellent tracking performance, has a reduced speed of only 5 fps.

Prerequisites for tracking face are first detecting the face region, obtaining its size and position in an image. Detecting a face in real time requires fast operation so that most of the existing algorithms use color and facial feature information [8-10]. The method of using color information is simple, and requires a small amount of computation; however, when an input image has similar color components, the accuracy of the detection decreases. In addition, the method [8-10] using the facial features works well with lighting changes, but has trouble with tracking an object of similar pattern.

To solve these problems, a new face tracking algorithm for real-time implementation in an acquired image is proposed in this paper. It uses color information first to detect the candidate face region, then uses the Haar features to detect the accurate face region [11]. When the face is detected using the Haar features based the AdaBoost (Adaptive Boosting) strong classifier [11, 12], it occasionally detects stripes or texts as faces. But, detecting the face candidate area by using the skin color [13, 14], it reduces the frequency of such false face detection.

In order to track the detected face, CamShift algorithm was proposed, which is a method of using color information for tracking [15]. CamShift is a stable and simple algorithm that compares the color histogram of the target face and that of the candidate face. However, when an object or a background has a color distribution similar to that of the target face, the algorithm shows instability in its search. CamShift algorithm uses variable size of search window, which is advantageous, as it allows objects to be tracked even when their size differs. But, if a background object with similar color distribution comes close to the object being tracked, CamShift may understand the background as the target object. Therefore, even though it may be successful in detecting the face candidate area, it still may be unsuccessful in correctly tracking the detected face [16, 17].

The use of depth information for tracking a face region can enhance its performance. Basically, a method of obtaining depth information is performed by acquiring images from a stereo camera and then calculating a disparity value of each pixel [18-20]. However, the method of calibrating multiple cameras takes time to calibrate after installing the camera, and it is troublesome to recalculate

† Corresponding Author: Dept. of Electronic Engineering, Kwang-Woon University, Korea. (jsyoo@kw.ac.kr)
* Dept. of Electronic Engineering, KwangWoon University, Korea. ({tarje3, guswh7905}@kw.ac.kr)
Received: October 3, 2016; Accepted: May 16, 2017

each time the position of the camera is changed. In addition, it is not suitable for real - time tracking algorithm such as face tracking because it requires considerable computation time to extract disparity map through images acquired from multiple cameras [21]. Microsoft's Kinect has the ability to obtain depth information in real time by acquiring RGB images just as general cameras at the same time [22, 23]. The proposed algorithm improves the tracking performance using the depth information acquired by Kinect, then applying it to the CamShift algorithm to overcome its shortcomings. In addition, even if the target face is disappeared or occluded, re-tracking is possible based on feature matching between the previously saved face template and the current frame [24, 25]. The algorithm proposed in this paper was developed to target a single object, that is, one face in a typical indoor environment.

This paper is composed as follows. Section 2 introduces the basic algorithm used to detect face region, while section 3 proposes the new real-time face tracking algorithm using depth information. Section 4 compares the performance of the proposed algorithm and that of the conventional algorithms, and section 5 concludes with the results of the experiment.

2. Fundamentals for Tracking the Face Region

2.1 Skin detection

In this paper, possible regions for skin color in an image are first detected based on the fact that the object being tracked is a person's face [13, 14]. It is not easy to accurately detect skin color. Finding the optimal algorithm for all skin colors is difficult, as a person's skin color depends on race and even within a race there is a wide range of skin colors. However, regardless of skin color, all humans have red tone in their skin because they all have blood.

The HSV color space, which has better ability to separate brightness and color components in images, is used. The candidate regions are detected using the hue value of color components within the HSV color space. When skin color candidate regions are detected, all the remaining regions are converted to zero value. Fig. 1 shows the results of the skin detection.

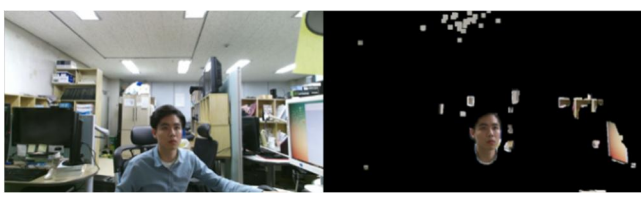


Fig. 1. (a) A given image, and (b) candidate areas detected by the skin detection algorithm

2.2 Face detection using haar features

Before tracking a face, we need to determine whether a face exists in an image, and perceive its location. In this paper, a face detection algorithm that uses Haar features is utilized. The general object detection method based on Haar features and AdaBoost proposed by Viola and Jones has high accuracy and fast processing speed [1]. The Haar feature reduces its processing time by using an integrated image, while the AdaBoost algorithm selects the highest distinguishing feature point to train the classifier. Placing multiple classifiers by complexity in a cascade structure retains highly effective processing speed, while maintaining accuracy. After training Haar features for human face, the cascade structure of Adaboost with the trained Haar features is used to detect the face region [11, 12].

Fig. 2 shows that many different types of Haar features are used to extract features from the target objects in an image. Haar features are composed of two or more adjacent rectangular regions. Characteristic values for a feature are defined by subtracting the sum of the pixel values corresponding to the area marked with white by the sum of the intensity of pixels on the area marked with black. Fig. 2 shows that there is nearly an infinite combination of features from inversion of black and white areas, x-axis and y-axis scaling, and zooming in and out operations

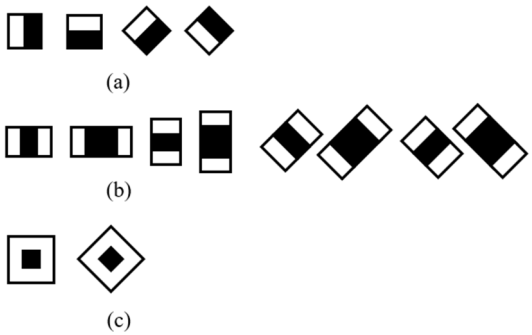


Fig. 2. Various sizes and shapes of Haar features: (a) edge features; (b) line features, and (c) center-surround features

Fig. 3. Face detection through using various Haar features [26]

on these features. From all the options available, it is important to select and use meaningful features that are relevant and proper. The meaningful features are distinguished by having consistently similar values to that of target, that is a face in this case and by showing inconsistent values to irrelevant objects. This meaningful feature selection is done automatically by Boosting learning algorithm [11, 12]. Fig. 3 shows a process for extracting a face by using a variety of Haar features [26].

2.3 CamShift algorithm

The CamShift (Continuously adaptive mean SHIFT) is an improved MeanShift algorithm [27] that adjusts the size of the search window on its own [15]. MeanShift is a nonparametric kernel density estimation algorithm that is repeatedly performed to find the local maximum value of probability distribution. CamShift also tracks an object using a color histogram. When the user inputs the size and position of the initial search area, it repeatedly compares histograms to track the target object. The biggest difference from MeanShift is that by applying variable window size, CamShift easily tracks objects that change size. CamShift processes in accord with the following order [15]:

a. Set the size and initial position of the search window.
b. Calculate the probability distribution of color information and perform the MeanShift to find the center of the search window.
c. Reset the search window by using the center position and size of the color histogram.
d. Using the reset search window, repeatedly perform the MeanShift until the search window converges, or repeat Steps b-d as often as defined. The position and size of the search window is determined through the calculation of the zeroth, 1st, and 2nd moments of the color histogram within the window. The zeroth, 1st, and 2nd moment are obtained by Eqs. (1) - (3), respectively.

$$M_{00} = \sum_X \sum_y I(x,y) \quad (1)$$

$$M_{10} = \sum_X \sum_y xI(x,y),\ M_{01} = \sum_X \sum_y yI(x,y) \quad (2)$$

$$M_{20} = \sum_X \sum_y x^2 I(x,y),\ M_{02} = \sum_X \sum_y y^2 I(x,y) \quad (3)$$

where $I(x,y)$ represents the search window's pixel value at (x,y). Using the zeroth, 1st, and 2nd moment, we can obtain the search window's center position (x_c, y_c) along with Eq. (4).

$$x_c = \frac{M_{10}}{M_{00}},\ y_c = \frac{M_{01}}{M_{00}} \quad (4)$$

3. The Proposed Algorithm

Fig. 4 shows the block diagram of the proposed face tracking algorithm in this paper that is composed of three

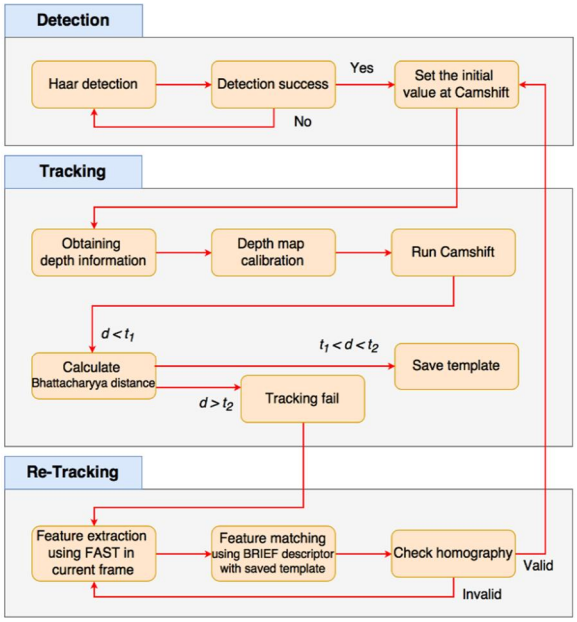


Fig. 4. A block diagram of the proposed algorithm

main parts, detection, tracking and re-tracking of the target face. First, in the Detection part, the face candidate group is generated by applying the skin detection algorithm described in Section 2.1 to the input image obtained from the Kinect. Then, the face is detected using the Haar feature described in Section 2.2. The position and size of the detected face are set to the initial value of CamShift.

In the Tracking part that tracks the detected face, the depth value of the detected face can be obtained through color image obtained from the Kinect and the calibration process of the depth map. We tracking faces with CamShift using depth information as described in Section 3.1. The Bhattacharyya distance [28] is calculated as described in Section 3.2. It saves the face template for the case that the tracking fails later or the failure of the tracking is judged. Finally, in the Re-Tracking part, the feature point matching is performed between the stored face template and the current frame when it is determined that the tracking has failed. If there are more than 4 pairs of correct matches, the validity is checked by homography calculation [29]. If it is valid, CamShift is set to the initial value and the tracking is restarted. Through these processes, we can propose a more stable and robust tracker by overcoming problems of CamShift.

3.1 Improved CamShift with depth information

Even though CamShift [15] are used in many areas of object tracking, there is still room for improvement. Problems of the Camshift include the initial search window settings, incorrect tracking of variable search windows, and tracking target objects apart from the search window. The proposed algorithm solves these problems. First, we solve the problem of having to set the initial search window

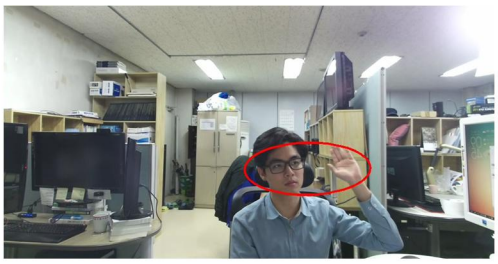


Fig. 5. Unstable tracking in an image due to similar colors

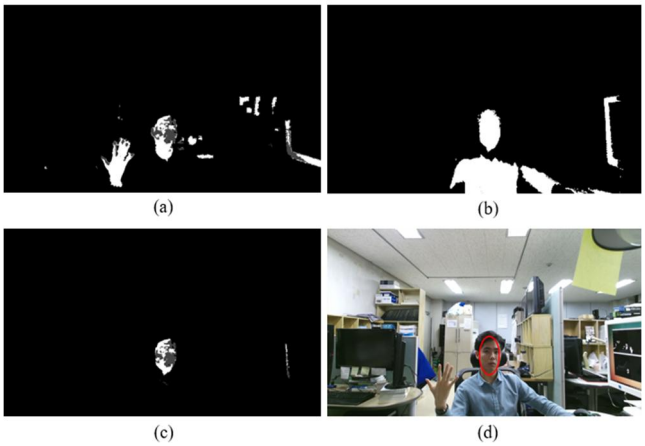


Fig. 6. (a) Visualization of color similarity: (b) pixels with the same depth as the face region tracked in the previous frame, (c) result of AND operation with (a) and (b), and (d) the result of face tracking

every time by face detection with the Haar features. By finding the position of the candidate face area in a given image, automatically setting it as the position of the initial search window. Then user can avoid having to set the search window every time.

Unlike MeanShift [27] that uses a fixed search window, CamShift uses variable search windows, which has the advantage of tracking objects that change size. However, a problem occurs if the tracking face comes close to a background with similar color distribution. The algorithm then recognizes the whole background with the similar color as the target object. Therefore, even though it may be successful at detecting the face region, it frequently fails to track the face correctly. Fig. 5 is an example of an unstable tracking case due to similar colors in the background.

In the proposed algorithm, the depth information along with color information is used to solve this problem. Fig. 6 (a) shows an image that visualizes the brightness value in the range 0 to 255 to represent the color similarity with the target face by computing the color distribution of the object that is selected in the initial search window. Fig. 6 (b) is also a visualized image of the pixels with brightness value of 255, which have the same depth value as that of the central pixel of the tracked face region in the previous frame. Fig. 6 (c) is the result of taking AND operation on Figs. 6 (a) and (b). Finally, Fig. 6 (d) shows the result of

(a) (b)

Fig. 7. Results of (a) the proposed algorithm and (b) of the CamShift

face tracking in a red ellipse. The proposed algorithm tracks the color using the depth information given in Fig. 6 (c), so that it improves the performance of CamShift and it is robust to backgrounds with similar colors.

In Fig. 7, it is shown that the proposed algorithm successfully tracks the face even when it is near a background with similar colors while the CamShift failed to do so.

As we already said, the CamShift algorithm tracks objects based on the specific kernel called the search window by using the distribution of color within the kernel. In particular, when the tracked object moves rapidly, processing the image takes a long time or the object's position is suddenly changed by the camera characteristics, then the target object does not exist within the search window so that the search is not possible any longer. In addition, tracking is also impossible if the object being tracked moves out of the camera's field of view. However, the proposed algorithm has the advantage of being able to re-track the target object, face by a feature matching even for these cases.

3.2 Determining the motion of face

In pattern recognition, the concept of distance generally represents the distance between the features, and is used as the standard to measure similarity between patterns. Two patterns positioned closer to a feature space usually have a greater degree of similarity. Many tools are used to measure the distance, such as Euclidean distance [30], Dynamic Time Warping(DTW) algorithm [31], and Bhattacharyya distance [28].

In this paper, the Bhattacharyya distance is used to measure the change in the tracked face, and to confirm failure in tracking. The Bhattacharyya distance d is defined by histogram comparison of the previous frame with the current frame as in Eq. (5). [28]

$$d(H_1, H_2) = \sqrt{1 - \sum_I \frac{\sqrt{H_1(I) \cdot H_2(I)}}{\sqrt{\sum_I H_1(I) \cdot \sum_I H_2(I)}}} \quad (5)$$

where, H_1 and H_2 are the histograms of two images being compared. I indicates the index of the histogram bin, while $H_1(I)$ and $H_2(I)$ mean histograms of the Ith bin of H_1 and H_2, respectively.

Fig. 8. Various size and angles of saved face templates

The histograms of two image are first normalized, then compared using Eq. (5). When two images being compared are perfectly the same, the Bhattacharyya distance should be 0, and approaches a value of 1 as the degree of similarity decreases. By comparing the results of CamShift for each frame, the frame with a similarity value greater than a threshold is saved to the template. A Bhattacharyya distance greater than the threshold means that the object has not come to a stop, but is still moving or rotating, causing changes in brightness distribution in an image. Fig. 8 illustrates face templates saved in various sizes and angles.

In the proposed algorithm, two threshold valuest_1, t_2 $(0 < t_1 < t_2 < 1)$ are used. t_1 and t_2 are determined through experiments and can change depending on the performance of hardware, camera, and light condition. Three cases are considered by comparing the Bhattacharyya distance d in Eq. (5), with t_1 and t_2. First, when the value of the Bhattacharyya distance d is less than t_1, the movement of the face is little to none, as there is a small difference between histograms of the previous and the current frames. In this case, by not processing anything, template duplications of a stationary object can be prevented.

Second, when the value of Bhattacharyya distance d is greater than t_1 and less than t_2, it can be assumed that there is movement of the face. In this case, face being tracked is saved to the template. In the proposed algorithm, if tracking is successful and a large number of templates are saved, it increases the possibility for re-tracking.

In other words, running the algorithm many times leads to better performance because the algorithm learns more about the object as the number of trials increases. However, template matching is based on features and the increase in the number of saved templates leads to longer process time for re-tracking. Therefore, appropriately controlling the maximum number of saved templates is crucial in determining the performance and process time of the algorithm.

Finally, when the value of the Bhattacharyya distance d is greater than t_2, it can be assumed that the histogram change of the face being tracked is great. In this case, it fails to track due to occlusion, or because the face is moving fast. When this case occurs, re-tracking is proceeded.

3.3 Re-tracking using feature matching

When failure occurs due to occlusion and/or due to fast movement of the face, the face can be tracked again through feature matching between stored templates and the current frame. The feature extraction algorithm and descriptor are determined in order to reduce the processing time because the proposed algorithm must be applied in real-time. FAST (Features from Accelerated Segment Test) algorithm is applied [24], because it has a fast processing time and excellent repeatability. In the case of the feature point descriptor, the SIFT algorithm [32] is most detailed in 128 dimensions and can be expected to have high accuracy. However, since the dimension of descriptors increases, the computational complexity increases. Therefore, in the proposed algorithm, the feature point matching is performed using the BRIEF descriptor [25]

3.4 Fast feature extraction by FAST

The feature points from FAST (Features from Accelerated Segment Test) algorithm [24] is better fitted for real-time because it extracts features very fast compared to that of basic feature extraction algorithms such as Harris Corner Detector [33], SIFT [32], and SURF [34].

Fig. 9 below illustrates the relationship between the center and nearby pixels in deciding the feature point. In a given image, a circle with distance 3 from the reference pixel P is first defined. If 16 pixels on the blue circle defined above are represented as $p \rightarrow x_k$, the feature points can be found by comparing the result of adding or subtracting the threshold value from either the intensity values of $I_{p \rightarrow x_k}$ or the intensity value of the reference pixel, I_p [24].

If intensity value of any of 16 pixels is greater than that of the summation of the intensity value of the reference pixel P and the threshold value, or it is smaller than the difference of the intensity value of the reference pixel P and the threshold value, and if this occurs more than N

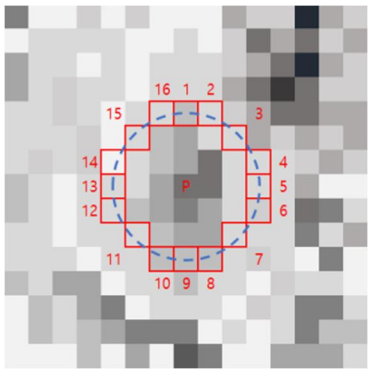


Fig. 9. Pixels on a circle centered at P with distance of 3

times, reference pixel *P* is determined to be a feature point. The value of *N* is usually 9, 10, 11, 12, etc., but when the value is 9, the repeatability of the feature is the highest [24]. Similar to the value of *N*, the threshold can be selected accordingly by the user. Selecting a low threshold means that more feature points are extracted, while selecting a high threshold means that less feature points are extracted.

A decision tree can also be used for faster processing. There are 3 cases: intensity values of 16 pixels are all greater than that of *P*, they are all less than *P*, or similar to *P*; by using this classification, the intensity distribution of pixels can be described as a ternary vector. Determining whether a pixel is a feature point or not can be done efficiently by applying this vector to the decision tree [24].

Because the above feature point extraction process only compares the intensity of corresponding pixels, feature points may be clustered. Therefore, choosing a representative point from the clustered features is necessary and done by NMS (Non-Maximum Suppression) method. The performance of FAST is sensitive to the image size: a bigger image means more feature points, and a smaller means less. Through NMS, it becomes robust to size change, decreasing the number of feature points in a big image by replacing multiple feature points into one representation.

3.5 Fast matching by the binary descriptor, BRIEF

BRIEF (Binary Robust Independent Elementary Features) [25] has a huge advantage in the memory efficiency by using a binary descriptor. The descriptor uses relatively small bits in comparison to ordinary descriptors, but still shows good performance. BRIEF descriptors use Hamming distance [35], rather than Euclidean distance [30], to provide faster and better efficiency than that of other descriptors. The Hamming distance between two binary strings of equal length is defined as the number of positions at which the corresponding bits are different. By comparing SURF [34] to BRIEF, BRIEF has better or equal recognition and matching performance with significantly faster processing [25].

3.6 Judgement validity of homography

A homography is a 3x3 matrix that represents the projection relationship between two images. In the proposed algorithm, homography is used to determine if the projection relationship is normal during the re-tracking process with a saved face templates and current frame [29].

Minimum of 4 pairs of feature points are required in order to find the homography matrix. Through feature-based matching between the saved templates and current frame, if 4 or more pairs correctly match, then the homography matrix can be calculated. At this time, RANSAC (RANdom SAmple Consensus) [36], an algorithm which reduces errors by predicting correct models within data mixed with errors and noise, is used to remove outliers (incorrect feature points). There are few cases of RANSAC failing so the result of project relationship is not always correct. Therefore, an additional process for double-checking the obtained homography matrix has correct relationship [29].

Normalized homography matrix H with 3 rows and 3 columns is defined by Eq. (6). The elements of the matrix are expressed as h_1~h_8 in order from row 1 and column 1 and the last one has 1.

$$\mathrm{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{bmatrix} \tag{6}$$

D in Eq. (7) is the determinant value of the 2x2 submatrix of H; if *D* is less than 0, the rotating order is not correct. If reflection or twist occurs, Eq. (7) is the crucial factor to determine the incorrect homography.

$$\mathrm{D} = h_1 h_5 - h_2 h_4 \tag{7}$$

In Eq. (8), X_s and Y_s are the scale factor that measures the amount of zoom-in and zoom-out of the *x* and *y* directions respectively. *P* is a perspective factor that represents the trapezoidal degree. If *P* has 0, it represents a rectangle, while as the value of *P* grows, it represents a shape further from a rectangle, i.e. a trapezoid.

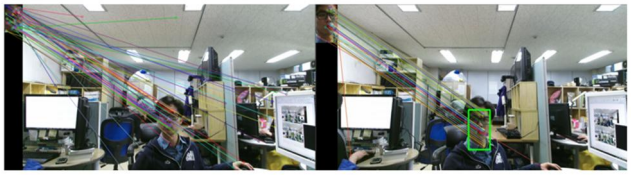
$$\begin{aligned} X_s &= \sqrt{h_1^2 + h_4^2} \\ Y_s &= \sqrt{h_2^2 + h_5^2} \\ \mathrm{P} &= \sqrt{h_7^2 + h_8^2} \end{aligned} \tag{8}$$

Various factors as shown above are used to determine whether or not the obtained homography is not proper.

Especially, 6 discriminants in Eq. (9) are used for this purpose. If any of the 6 givens are applicable, it is an incorrect transformation [29].

$$\mathrm{D} \le 0 \parallel X_s < 0.1 \parallel X_s > 3 \parallel Y_s < 0.1 \parallel Y_s > 3 \parallel P > 0.002 \tag{9}$$

If there is no case in Eq. (9) occurred, the current frame's projected position of the target face is reset as the



(a) (b)

Fig. 10. Mismatching result with various templates, and (b) success in re-tracking the face with the correct homography conversion

Table 1. Comparison of algorithm performance

Sequence	Algorithm	Max fps(fps)	Min fps(fps)	Average(fps)	Object miss frame (frame)	Total frame (frame)	Tracking success rate (%)
self-shooting	TLD	5	2	2.362	61	726	91.597
	Proposed	23	9	19.336	43	726	94.077
	CamShift	30	28	28.500	377	726	48.071
face_move1	TLD	6	4	4.223	0	469	100.000
	Proposed	33	32	32.147	0	469	100.000
	CamShift	34	32	32.850	0	469	100.000
face_occ2	TLD	8	3	5.829	8	387	97.933
	Proposed	33	7	22.608	59	387	84.755
	CamShift	34	33	33.094	0	387	100.000
face_occ3	TLD	10	5	8.231	40	262	84.733
	Proposed	33	7	25.848	39	262	85.115
	CamShift	33	32	32.707	0	262	100.000
face_occ5	TLD	8	4	4.529	0	330	100.000
	Proposed	33	7	28.161	22	330	93.333
	CamShift	33	32	32.439	0	330	100.000
face_turn2	TLD	7	4	4.207	0	600	100.000
	Proposed	33	7	30.468	10	600	98.333
	CamShift	51	40	45.386	0	600	100.000

initial position, and re-tracking is started. Fig. 10 illustrates the re-tracking process after it has stopped, due to occlusion of the target face. The face on the top left corner in Fig. 10 is one of the templates that are saved and the right one show current frame. Fig. 10 (a) represents the mismatching case with a saved template and consequently results in incorrect homography matrix while Fig. 10 (b) represents success in re-tracking the face with correct homography after correct feature point matching.

4. Experimental Results

The proposed algorithm receives RGB images of 3 channel with 1920x1080 resolution and depth image of 1 channel with 512x424 resolution through Microsoft's Kinect-v2. After resizing RGB images to 960x540, we performed experiments on Intel i5-4690 3.50GHz CPU, 16GB RAM and Visual Studio 2013 environment without acceleration of the graphics card (GPU). Since the proposed algorithm uses depth image acquired from Kinect, it is assumed that an object moves within 0.5m ~ 4.5m range of Kinect's depth acquisition. The proposed algorithm is compared with Face-TLD: Tracking-Learning-Detection [6, 7], CamShift [15].

Due to the nature of the proposed algorithm, color images and depth images with depth information are needed at the same time. Therefore, we can't experiment on all public datasets. A total of 726 frames were taken from Kinect-v2 self-shooting sequence. In addition, we used five sequences of human faces among sequences [37] that provide both color and depth information. The sequences were taken using Kinect-v1, both color and depth images are provided with a 640x480 resolution per frame PNG file format [37]. The self-shooting sequence captures color and depth images in real time from Kinect-v2, so it shows a slightly slower speed than that of other five sequences by loading the stored image

Fig. 11 compares the processing speed of the three algorithms. Since all three algorithms start at 0 fps, it is the speed before setting the initial position of an object to be tracked, so some frames with 0 fps rate are excluded in calculating the minimum fps and average fps. There was an increase in TLD algorithm's processing speed because when it was unable to track a candidate face area due to occlusion. On the other hand, the proposed algorithm showed a decrease in processing speed for the same case. In general, CamShift algorithm using depth information improves the processing time; but in situations when an object cannot be tracked, performing feature point matching between the face template and current frame demands more computations.

The CamShift algorithm shows fast and stable speed at a speed of at least 28 fps, but there is no processing speed as shown in Fig. 11 (a) after 349 frames in the self-shooting sequence due to the problem that tracking fails and the object can't be retraced. There is no case where retrace fails for the remaining 5 sequences except the self-shooting sequence. However, resulting images show that they are all false traces.

Table 1 compares the performance of TLD, CamShift, and the proposed algorithm. The order in terms of speed was CamShift algorithm, proposed algorithm, and then TLD algorithm with the CamShift being the fastest; and the order for accuracy was the proposed algorithm, TLD, then CamShift with the proposed algorithm being the most

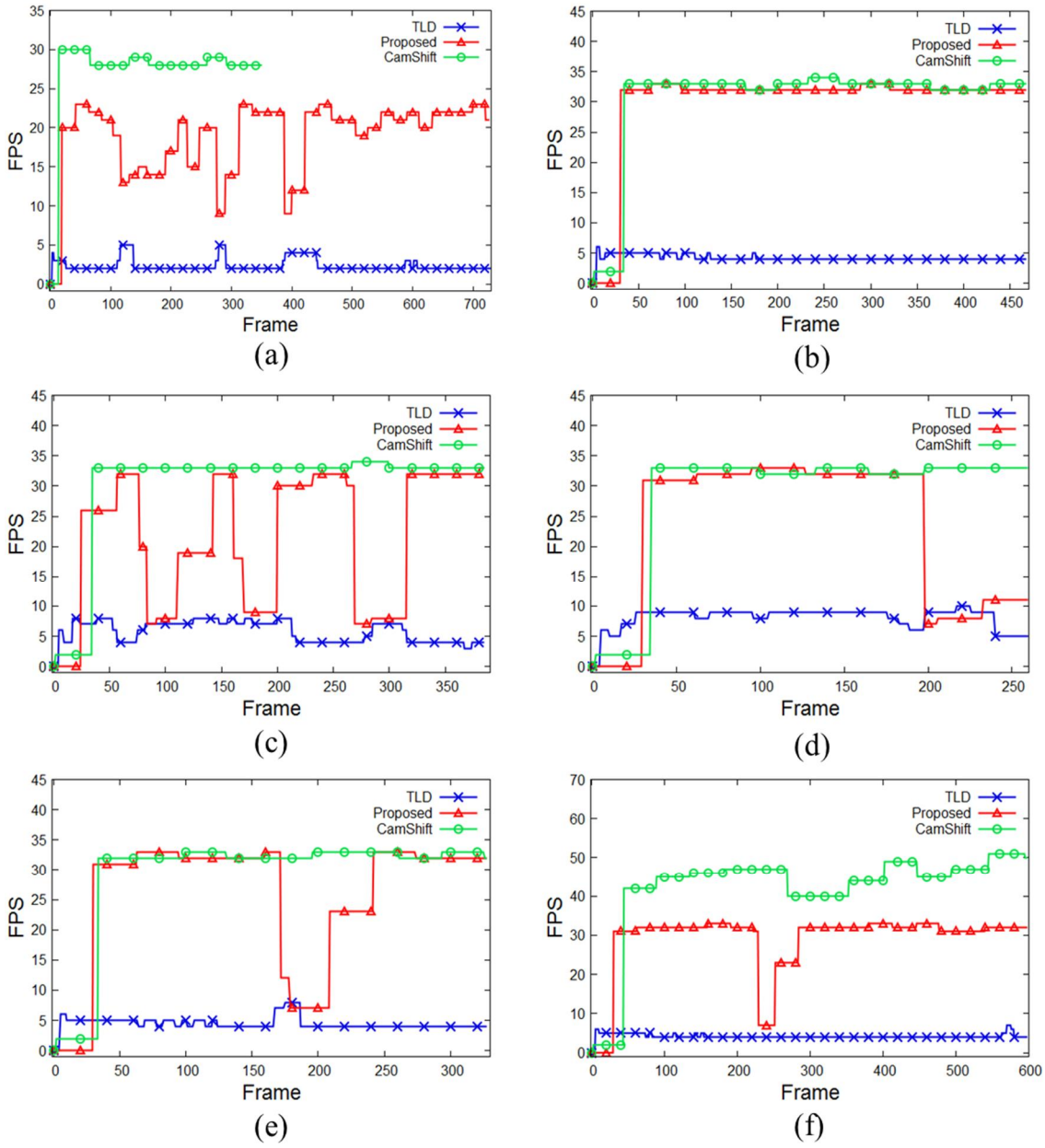


Fig. 11. Comparison of the processing speed of three algorithms. (a) self-shooting sequence (b) face_move1 sequence (c) face_occ2 sequence (d) face_occ3 sequence (e) face_occ5 sequence (f) face_turn2 sequence.

precise. It has already been shown that CamShift is unsuitable for tracking at self-shooting sequence, as it is unable to re-track the face. Other five sequences shows incorrect tracking result because CamShift uses only simple color information.

When the object being tracked is occluded, the entire image is regarded as an object so that the tracking success rate is high. TLD algorithm and proposed algorithm showed high tracking success rate. However, the problem of tracking a non-real object region in the TLD algorithm was found in 4 sequences out of 6 sequences, but 1 out of 6 sequences in the proposed algorithm. The TLD algorithm has an average processing speed of 4.897 fps while the proposed algorithm has an average of 26.428 fps. If we use GPU to speed up, it seems to be no problem in real time implementation.

Fig. 12 shows how all 3 algorithms are successful in tracking, even when the face is half-occluded with the 112th frame of self-shooting sequence. Unlike the TLD algorithm in Fig. 12 (b), the proposed algorithm in Fig. 12 (c) and the CamShift algorithm in Fig. 12 (d) track only non-occluded facial regions due to the variable size of the search window.

Fig. 13 shows the situation when the face is completely occluded at the 133th frame of the self-shooting sequence. Fig. 13 (b) and (d) show that the TLD algorithm and the CamShift algorithm are tracking incorrectly, even though the object to be tracked has disappeared. The proposed algorithm in Fig. 13(c) determines that the tracked object

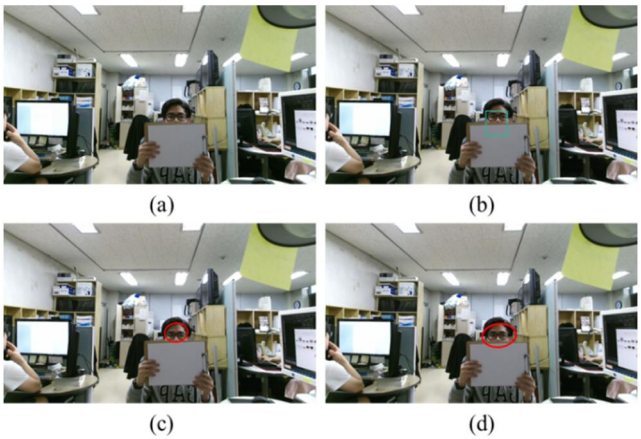


Fig. 12. The result of face tracking for frame #112: (a) reference image, (b) TLD algorithm, (c) the proposed algorithm, and (d) CamShift algorithm

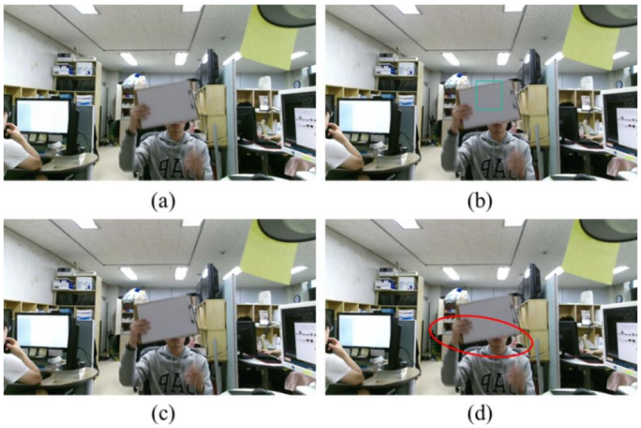


Fig. 13. The result of face tracking for frame #133: (a) reference image, (b) TLD algorithm, (c) the proposed algorithm, and (d) CamShift algorithm

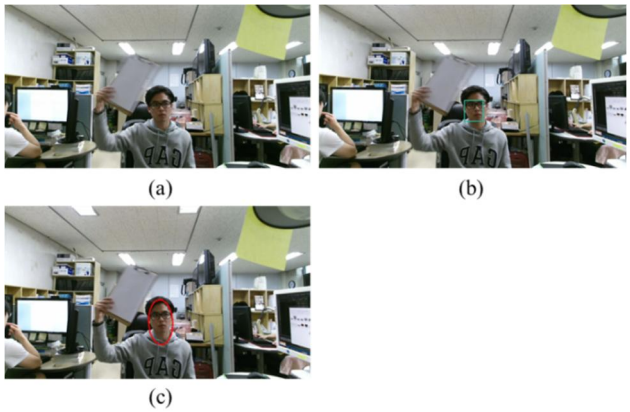


Fig. 14. The result of face tracking of frame #360: (a) reference image, (b) TLD algorithm, and (c) the proposed algorithm

is occluded by Bhattacharyya distance calculation and switches to the re-tracking mode.

Fig. 14 illustrates the situation when the occluded face

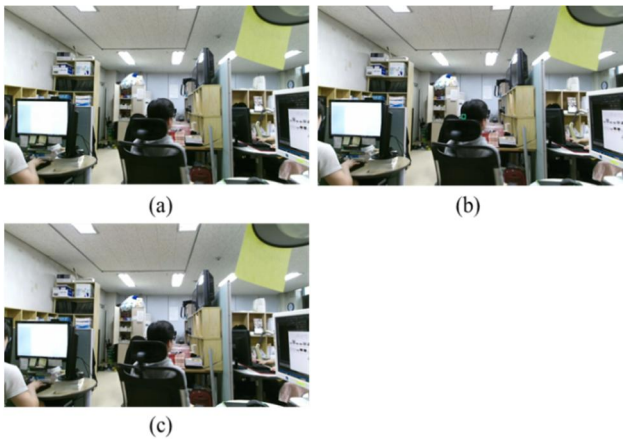


Fig. 15. The result of face tracking of frame #403: (a) reference image, (b) TLD algorithm, and (c) the proposed algorithm

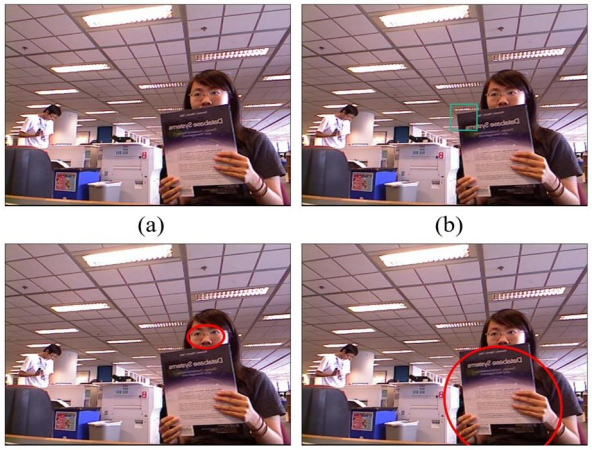


Fig. 16. The result of face tracking for face_occ2 sequence frame #24: (a) reference image, (b) TLD algorithm, (c) the proposed algorithm, (d) CamShift algorithm

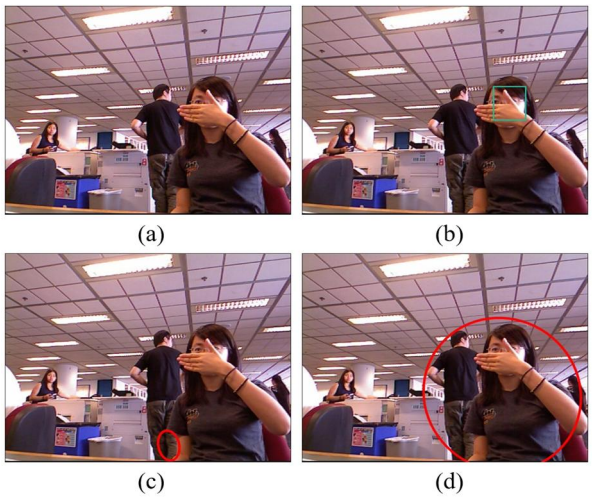


Fig. 17. The result of face tracking for face_occ2 sequence frame #175: (a) reference image, (b) TLD algorithm, (c) the proposed algorithm, (d) CamShift algorithm

reappears as in the 360th frame. Fig. 14(b) and (c), each shows success in re-tracking for the TLD algorithm and the proposed algorithm, while CamShift algorithm is unable to do so, as it failed in re-tracking after occlusion from the 349th frame onwards.

Fig. 15 also shows a situation where the face cannot be seen as in the 403th frame. The TLD algorithm of Fig. 15 (b) shows the result of incorrect tracking.

In self-shooting sequence, the calculated successful tracking-rate for TLD algorithm is 91.597%, as it failed to track 61 frames from a total of 726 frames. The algorithm may recognize it as successful tracking, when in reality there was occasionally incorrect object tracking in frames as in Figs. 13(b) and 15(b). Consequently, we predicted that the success rate of correct face tracking in the TLD algorithm be a bit lower than the calculated 91.597%.

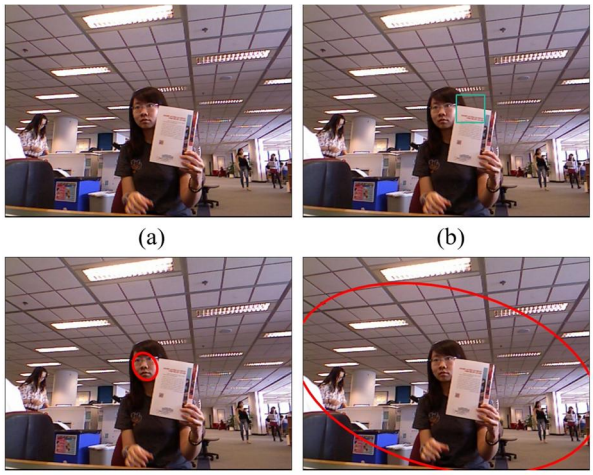


Fig. 18. The result of face tracking for face_occ3 sequence frame #104: (a) reference image, (b) TLD algorithm, (c) the proposed algorithm, (d) CamShift algorithm.

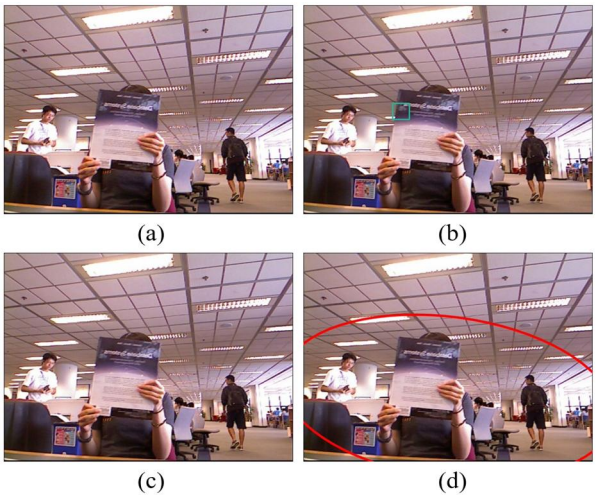


Fig. 19. The result of face tracking for face_occ5 sequence frame #178: (a) reference image, (b) TLD algorithm, (c) the proposed algorithm, (d) CamShift algorithm

In self-shooting sequence, the proposed algorithm had a high tracking rate of 94.077%. There were no signs of incorrect object tracking frames. Even though CamShift algorithm has fast and stable processing, it had various issues, as it failed to track correct objects in occlusion, and to re-track after occlusion, as shown in Fig. 13(d). CamShift had a low success rate in tracking of 48.071%.

Fig. 16 shows that TLD algorithm and CamShift algorithm do incorrect tracing in the 24th frame of the face_occ2 sequence. In the case of the TLD algorithm, the case of incorrect tracking immediately before the object being tracked was found in four of the six sequences. For the remaining 5 sequences except the self-shooting sequence, the CamShift algorithm shows false traces that recognize all backgrounds other than the face to be tracked as objects. This is because it uses color information and variable windows that are the characteristics of CamShift.

Fig. 17 shows the 175th frame of the face_occ2 sequence. The TLD algorithm keeps track of the position of the face in the previous frame, although the face is covered by the hand as shown in Fig. 17 (b). The algorithm shows the case of false tracking when the tracking face disappears due to occlusion. The proposed algorithm shows false tracking only in one of six sequences.

Fig. 18 is the 104th frame of the face_occ3 sequence and Fig. 19 is the 178th frame of the face_occ5 sequence. As mentioned above, the TLD algorithm incorrectly tracks objects after they were hidden. The proposed algorithm distinguishes the occlusion by calculating the distance between the frames. CamShift failed in tracking for all frames.

The video results of these experiments are also available at https://www.youtube.com/channel/UCrNCHOY3CUhWpndRbP6O7Yg

5. Conclusion

In this paper, we propose a face tracking algorithm in indoor environment, which improves the disadvantages of CamShift using depth information obtained from Microsoft's Kinect. In this paper, a new face tracking algorithm that uses depth information from Microsoft's Kinect to overcome drawbacks in the existing CamShift is proposed. The proposed algorithm solved the following issues in the old CamShift: incorrect tracking when the object lies adjacent to similar colors, failing to re-track, and cumbersomeness for users to have to manually input the targets being tracked. There are many researches being conducted for face tracking. But there are problems for algorithms with high efficiency, as they generally have greater calculation and processing time, which causes difficulty in real-time implementation. The proposed algorithm showed a high processing speed with Intel i5-4690 3.50GHz CPU, 16GB RAM, Visual Studio 2013,

even without acceleration of the GPU. It also shows a similar or better face tracking success rate than the TLD algorithm for self-shooting sequences and also shows superior tracking speed compared to TLD algorithm.

Acknowledgements

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. 2015-0-00258, Development of hybrid audio contents production and representation technology for supporting channel and object based audio)

References

[1] Viola, Paul, and Michael J. Jones. "Robust real-time face detection." *International journal of computer vision* 57.2 (2004): 137-154.

[2] Rowley, Henry A., Shumeet Baluja, and Takeo Kanade. "Neural network-based face detection." *IEEE Transactions on pattern analysis and machine intelligence* 20.1 (1998): 23-38.

[3] Osuna, Edgar, Robert Freund, and Federico Girosit. "Training support vector machines: an application to face detection." *Computer vision and pattern recognition, 1997. Proceedings., 1997 IEEE computer society conference on*. IEEE, 1997.

[4] Hsu, Rein-Lien, Mohamed Abdel-Mottaleb, and Anil K. Jain. "Face detection in color images." *IEEE transactions on pattern analysis and machine intelligence* 24.5 (2002): 696-706.

[5] Hjelmås, Erik, and Boon Kee Low. "Face detection: A survey." *Computer vision and image understanding* 83.3 (2001): 236-274.

[6] Kalal, Zdenek, Krystian Mikolajczyk, and Jiri Matas. "Face-tld: Tracking-learning-detection applied to faces." *Image Processing (ICIP), 2010 17th IEEE International Conference on*. IEEE, 2010.

[7] Kalal, Zdenek, Krystian Mikolajczyk, and Jiri Matas. "Tracking-learning-detection." *IEEE transactions on pattern analysis and machine intelligence* 34.7 (2012): 1409-1422.

[8] Kim, Young-Gon, Rae-Hong Park, and Seong-Su Mun. "Face Detection Using Adaboost and Template Matching of Depth Map based Block Rank Patterns." *Journal of Broadcast Engineering* 17.3 (2012): 437-446.

[9] Kim, Hoo Hyun, et al. "Rotation Invariant Face Detection with Boosted Random Ferns." *Proceedings of the Korean Society of Broadcast Engineers Conference*. The Korean Institute of Broadcast and Media Engineers, 2013.

[10] Lee, Kyong-Ho. "Face Tracking Using Face Feature and Color Information." *Journal of the Korea Society of Computer and Information* 18.11 (2013): 167-174.

[11] Viola, Paul, and Michael Jones. "Rapid object detection using a boosted cascade of simple features." *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. vol. 1. IEEE, 2001.

[12] Viola, Paul, and Michael Jones. "Fast and robust classification using asymmetric adaboost and a detector cascade." *Advances in neural information processing systems*. 2002.

[13] Jones, Michael J., and James M. Rehg. "Statistical color models with application to skin detection." *International Journal of Computer Vision* 46.1 (2002): 81-96.

[14] Vezhnevets, Vladimir, Vassili Sazonov, and Alla Andreeva. "A survey on pixel-based skin color detection techniques." *Proc. Graphicon*. vol. 3. 2003.

[15] Bradski, Gary R. "Computer vision face tracking for use in a perceptual user interface." (1998).

[16] Allen, John G., Richard YD Xu, and Jesse S. Jin. "Object tracking using camshift algorithm and multiple quantized feature spaces." *Proceedings of the Pan-Sydney area workshop on Visual information processing*. Australian Computer Society, Inc., 2004.

[17] Wang, Zhaowen, et al. "CamShift guided particle filter for visual tracking." *Pattern Recognition Letters* 30.4 (2009): 407-413.

[18] Zhang, Zhengyou. "A flexible new technique for camera calibration." *IEEE Transactions on pattern analysis and machine intelligence* 22.11 (2000): 1330-1334.

[19] Tsai, Roger. "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses." *IEEE Journal on Robotics and Automation* 3.4 (1987): 323-344.

[20] Weng, Juyang, Paul Cohen, and Marc Herniou. "Camera calibration with distortion models and accuracy evaluation." *IEEE Transactions on pattern analysis and machine intelligence* 14.10 (1992): 965-980.

[21] Mühlmann, Karsten, et al. "Calculating dense disparity maps from color stereo images, an efficient implementation." *International Journal of Computer Vision* 47.1-3 (2002): 79-88.

[22] Zhang, Zhengyou. "Microsoft kinect sensor and its effect." *IEEE multimedia*19.2 (2012): 4-10.

[23] Pagliari, Diana, and Livio Pinto. "Calibration of kinect for xbox one and comparison between the two generations of microsoft sensors." *Sensors* 15.11 (2015): 27569-27589.

[24] Rosten, Edward, and Tom Drummond. "Machine learning for high-speed corner detection." *Computer Vision–ECCV 2006* (2006): 430-443.

[25] Calonder, Michael, et al. "Brief: Binary robust

independent elementary features." *Computer Vision–ECCV 2010* (2010): 778-792.
[26] https://www.flickr.com/photos/unavoidablegrain/6884354772/in/photostream/ (Image by Greg Borenstein)
[27] Comaniciu, Dorin, and Peter Meer. "Mean shift: A robust approach toward feature space analysis." *IEEE Transactions on pattern analysis and machine intelligence* 24.5 (2002): 603-619.
[28] Bhattacharyya, Anil. "On a measure of divergence between two multinomial populations." *Sankhyā: the indian journal of statistics* (1946): 401-406.
[29] Trzcinski, Tomasz, and Vincent Lepetit. "Efficient discriminative projections for compact binary descriptors." *European Conference on Computer Vision*. Springer, Berlin, Heidelberg, 2012.
[30] Danielsson, Per-Erik. "Euclidean distance mapping." *Computer Graphics and image processing* 14.3 (1980): 227-248.
[31] Müller, Meinard. *Information retrieval for music and motion*. vol. 2. Heidelberg: Springer, 2007.
[32] Lowe, David G. "Distinctive image features from scale-invariant keypoints." *International journal of computer vision* 60.2 (2004): 91-110.
[33] Harris, Chris, and Mike Stephens. "A combined corner and edge detector." *Alvey vision conference*. vol. 15, no. 50. 1988.
[34] Bay, Herbert, et al. "Speeded-up robust features (SURF)." *Computer vision and image understanding* 110.3 (2008): 346-359.
[35] Hamming, Richard W. "Error detecting and error correcting codes." *Bell Labs Technical Journal* 29.2 (1950): 147-160.
[36] Fischler, Martin A., and Robert C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography." *Communications of the ACM* 24.6 (1981): 381-395.
[37] http://tracking.cs.princeton.edu/index.html
[38] Song, Shuran, and Jianxiong Xiao. "Tracking revisited using RGBD camera: Unified benchmark and baselines." *Proceedings of the IEEE international conference on computer vision*. 2013.
[39] http://darkpgmr.tistory.com/80

Jun-Hwan Lee He received the B.S degree at department of electrical engineering, KwangWoon University, Wolgye-dong, Nowon-gu, Seoul 01897, Republic of Korea in 2016. He is currently pursuing the M.S degree at Kwangwoon University. His research interests include computer vision, image processing, signal processing and deep-learning.

Hyun-jo Jung He received the B.S degree at department of electrical engineering, KwangWoon University, Wolgye-dong, Nowon-gu, Seoul 01897, Republic of Korea in 2015. He received the M.S. degree at department of electrical engineering, KwangWoon university, Wolgye-dong, Nowon-gu, Seoul 01897, Republic of Korea in 2017. His research interests include computer vision, image processing, signal processing and deep-learning.

Jisang Yoo He received the B.S degree at department of electrical engineering, Seoul National University, 1, Gwanak-ro, Gwanak-gu, Seoul 08826, Republic of Korea in 1985. He received the M.S. degree at department of electrical engineering, Seoul National University, 1, Gwanak-ro, Gwanak-gu, Seoul 08826, Republic of Korea in 1987. He received the Ph.D degree at department of electrical engineering, Purdue University, 610 Purdue Mall, West Lafayette, IN 47907, the United States of America in 1993. He is currently a professor with the department of electronics engineering, KwangWoon University, Wolgye-dong, Nowon-gu, Seoul 01897, Republic of Korea. His research interests include computer vision, image processing, signal processing and deep learning.