Web camera pulse detection with consideration of ambient light change and motion

Zhang kaizong

Abstract:

Plethysmographic signals are measured remotely using ambient light and a simple web camera from normal laptops in real time, from which we have extracted the heart rate (pulse rate). In the program Open-CV python package is utilized to control web camera and conduct face recognition. The forehead of the face image is used as Region-of-Interest (ROI) to extract plethysmographic signals. All the 3 color channels are averaged up and FFT-analysis is implemented to find the heart rate. The signal is displayed in near real-time. In order to take head's large motion movement into consideration, the face recognition is implemented in an independent python process for the whole running period and the detection results are given as the feed-back to the main process. Large motion will lead to ROI shifting so as to always focus on the forehead. Ambient light change is also measured by averaging up non-face area intensity for each acquired frame, which is assumed to be proportional to the ambient light intensity. This information is finally used to correct the ROI signal for each frame and the measured results indicate that the heart rate is more accurate than non-corrected signal.

1. Introduction

Detection of the cardio-vascular pulse wave traveling through the body is referred to as Plethysmography ('Plethysmos' = increase in Greek) and can be done by means such as variations in air pressure, impedance, or strain. Photo-plethysmography (PPG), introduced in the 1930's [1] uses light reflectance or transmission and is the least expensive method and simple to use. PPG is based on the principle that blood absorbs light more than surrounding tissue so variations in blood volume affect transmission or reflectance correspondingly.

Normally, PPG has always been performed with dedicated light sources and typically red and/or infra-red (IR) wavelengths. Ambient visible light is often considered a source of noise [2-4] when using IR light sources and detectors sensitive for IR and visible light. In this report, I show that PPG signals can be remotely measured on the human face in real time, using a simple digital web camera from normal laptops and normal ambient light to be illuminating source.

2. Method

2.1 Device and setup

The whole program is modified from Github project:

https://github.com/thearn/webcam-pulse-detector

Computers equipped with Web-camera and python3 are capable of running this program. For the moment, Windows system and Mac system are tested to run the program normally. Besides, the web camera should be operating in color mode.

2.2 Plethysmography light acquisition

This application uses [OpenCV](http://opencv.org/) to find the location of the user's face, then isolate the forehead region. Data is collected from this location over time to estimate the user's heart rate. This is done by measuring average optical intensity in the forehead location, in the sub-image's 3 channels. Physiological data can be estimated this way thanks to the optical absorption characteristics of (oxy-) haemoglobin [5].

With good lighting and minimal noise due to motion, a stable heartbeat should be isolated in about 15 seconds. Once the user's heart rate has been estimated, real-time phase variation associated with this frequency is also computed. This allows for the heartbeat to be exaggerated in the post-process frame rendering, causing the highlighted forehead location to pulse in sync with the user's own heartbeat.

2.3 ambient light recognition

Normally the ambient light will not change so significantly in steady environment. However, in order to quantify the ambient light, we take advantage from the non-face area in the image, denoted as NFA, which we assume that the average intensity of NFA should be proportional to the ambient light level. The face-area denoted as FA is recognized with OpenCV package. The whole frame from the camera is denoted as F. Thus the non-face area in the image can be easily obtained by eq.1.

$$\mathrm{NFA} = \mathrm{F} - \mathrm{FA} \qquad \text{eq.1}$$

The average value of NFA is calculated for each frame by eq.2

$$\mathrm{Avg_{NFA}} = \frac{\mathrm{Total_F} - \mathrm{Total_{FA}}}{\mathrm{N_F} - \mathrm{N_{FA}}} \qquad \text{eq.2}$$

where the $\mathrm{Total_F}$ and $\mathrm{Total_{FA}}$ are the summed up intensity of the whole frame and face area, $\mathrm{N_F}$ and $\mathrm{N_{FA}}$ are the number of pixels for whole frame and face area.

As described above, the signal is obtained with Avg_{FA}. In order to correct the ambient light change effect, the signal for each frame is divided by its Avg_{NFA}.

2.4 Motion detection and correction

When the program starts acquisition, the original program from Github uses fixed ROI from the face recognition before the acquisition. However, if the subject moves the head drastically, the ROI does not change according to the new position. In the worst cases it does not remain to be forehead or even not part of the face, which will for sure have an affect on the heart rate detection.

However, if we want to detect the face for every frame that is read from the camera, the vast computation time on face recognition will reduce the sampling rates dramatically, even to lower than the heart rates. Thus, the program does not meet Nyquist Sampling theory and eventually it will crash due to out-index in the List.

One proposal would be using multi-threading for parallelizing the data acquisition and face recognition, because the data can be shared within different threads. However, standard Python interpreter does not support multi-threading in an efficient way due to Global Interpreter Lock (GIL). Thus instead, the multi-processing Python module is utilized to realize real-time face-recognition throughout the whole detecting process.

However, this also brings up the question of how to share data between different process without influencing the main process sampling rate too much. The flow-chart of the multi-processing detection is show in fig.1.

Main process and child process share 5 variables to realize the work flow. Each of them is explained below:

- **img_data_share**: Image data shared from Main process to Child process.
- **is_face_im_data_ready**: The main process will provide the image data and set this to true. When this flag is true, the main process will stop provide data. Meanwhile, the child process checks whether this flag is true. If so, it will grab the data and start face recognition. When the recognition is finished, the child process will set this flag to False. Then the main process will continue provide data. So on so forth.
- **is_face_coord_detected**: The child process might fail to detect the face area. This flag indicates whether child process find the face rectangle. If not, the main process will not update the face coordinate.
- **coord_detected**: If the child process detects the face area, the coordinate is stored in this variable and the main process will update its face coordinate to change the ROI.
- **terminate_flag**: If the main process is going to quit, this flag is turned to true. The child process will check this variable and decide whether to quit the loop.

Normally the movement of the head is not so significant and the mean intensity of the ROI is taken as the signal, so I think small movement does not change the results too much and this is tolerable. When the face coordinate has a big shift, then the ROI should be changed according to the detected face coordinate. In the program, we normally check whether the detected coordinate center has a big shift compared with current one. If the shift excesses the pre-defined threshold (defined as the 1/40 of the image frame width in my program), the ROI will be updated. The movement threshold can be set by user.

In order to be compatible with both Mac and Windows systems, in the multi-process program, it is necessary to set the method to start new process as "*multiprocessing.set_start_method('spawn')*". Because it is not possible in Windows to start new process using system Fork() function, which is the fast and a standard way in Unix-like operating system, i.e. , Mac.

Moreover, the data sharing for multi-process program in different operating systems has different speed. It is observed that Mac can have better sampling rate compared with Windows, which I guess is due to the fact that Unix-like operating systems are more adaptable for multi-process programs.

In order to overcome the slow sampling rate on windows system, we reduce the frequency of image data supply from the Main Process. As is demonstrated in fig.1, only after the Main Process has continuously obtained a certain number of frames (set to be 15 frames in our program), it will provide the last one to the Child Process for face recognition.

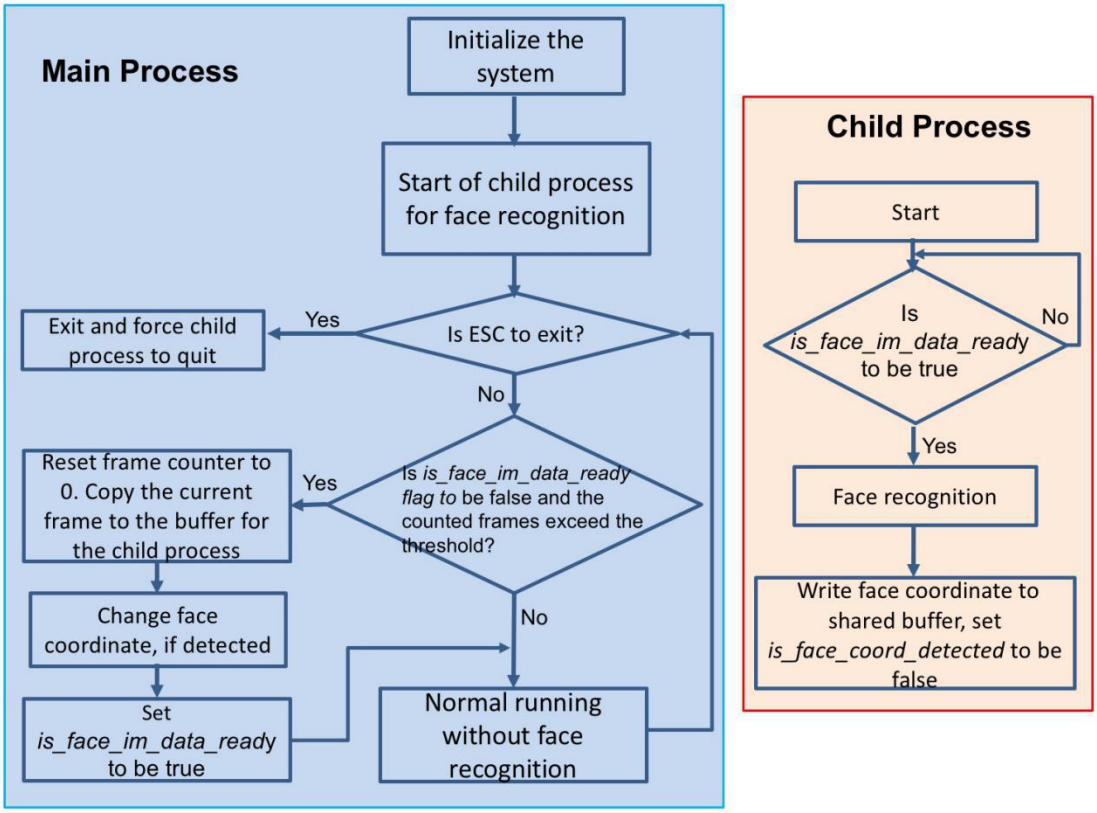

Fig.1 Flow chart of the multi-process program

3. Experiments and Results

3.1 Ambient light evaluation

3.1.1 Ambient light intensity

We measured the non-face area average intensity for half minute and the measured signal is shown in fig.2. It is easy to find that the ambient light in a steady environment does not have so much fluctuations. The maximum change is around 0.345% compared with the mean intensity. It seems that to be not necessary to correct. But in non-steady environment, it might be still worthy doing this.

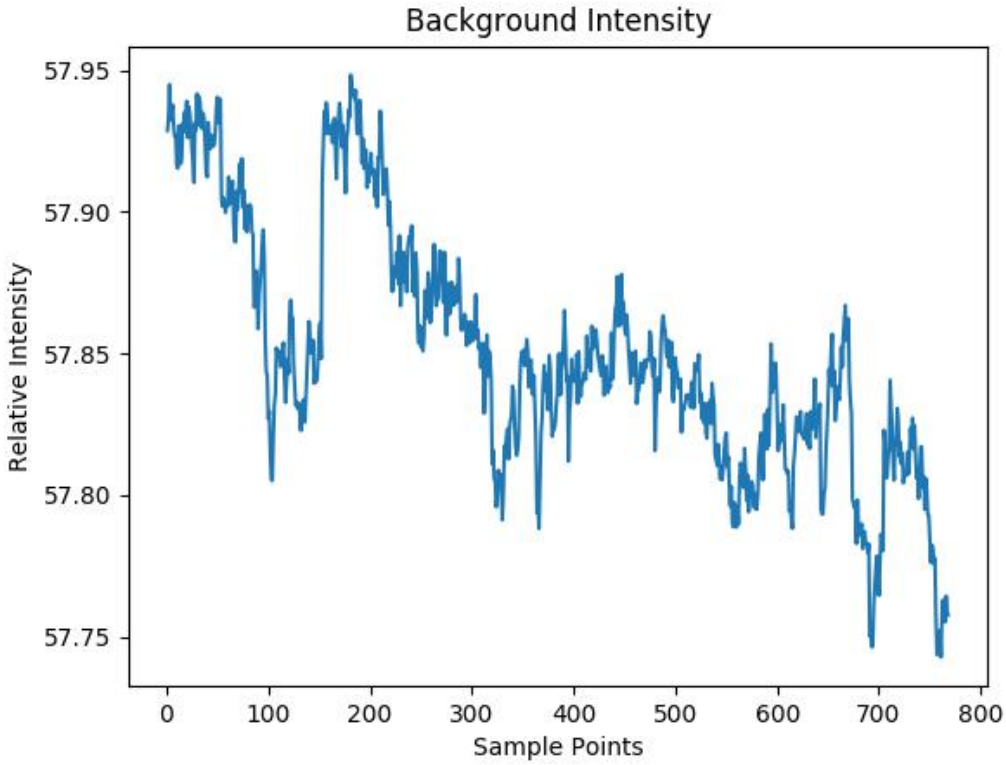

Figure. 2 The background intensity measure for each frame. The total time lasts around half minute.

3.1.2 Experimental results

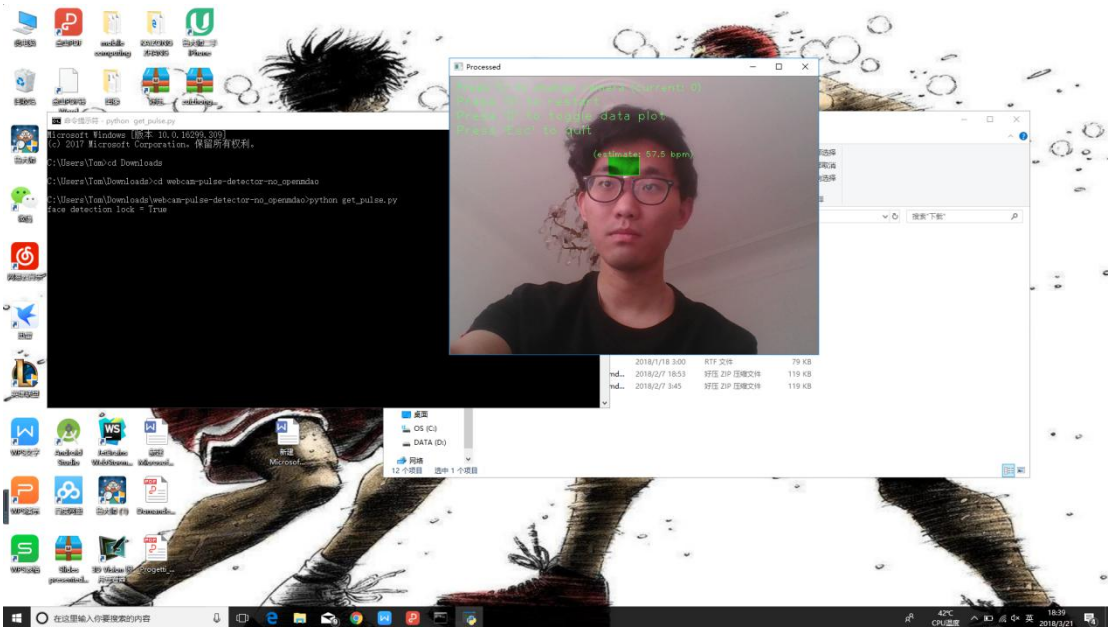

Figure. 3 Initial code test data

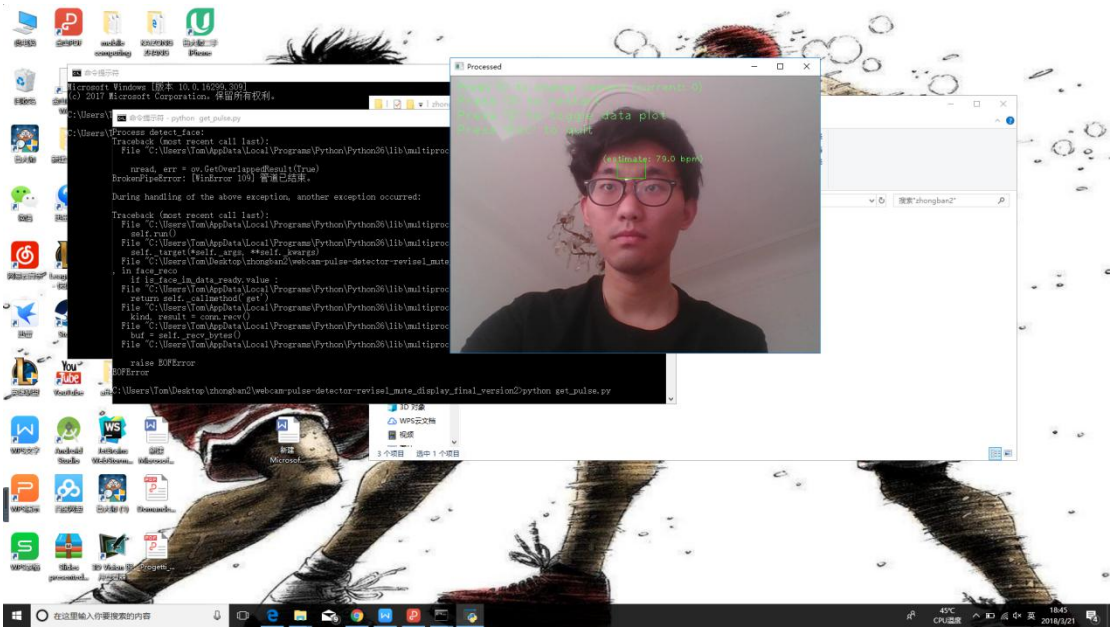

Figure. 4 Optimized code test data

According to the test results, the optimized code improves the accuracy of the background correction.

3.2 Movement detection

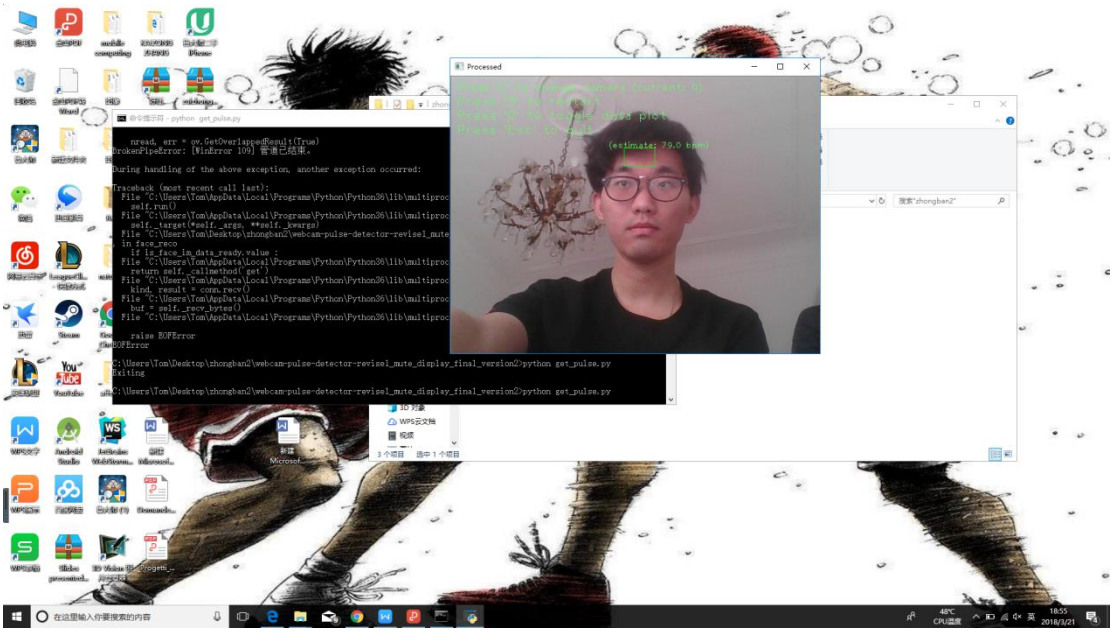

Figure. 5 start testing(0:05)

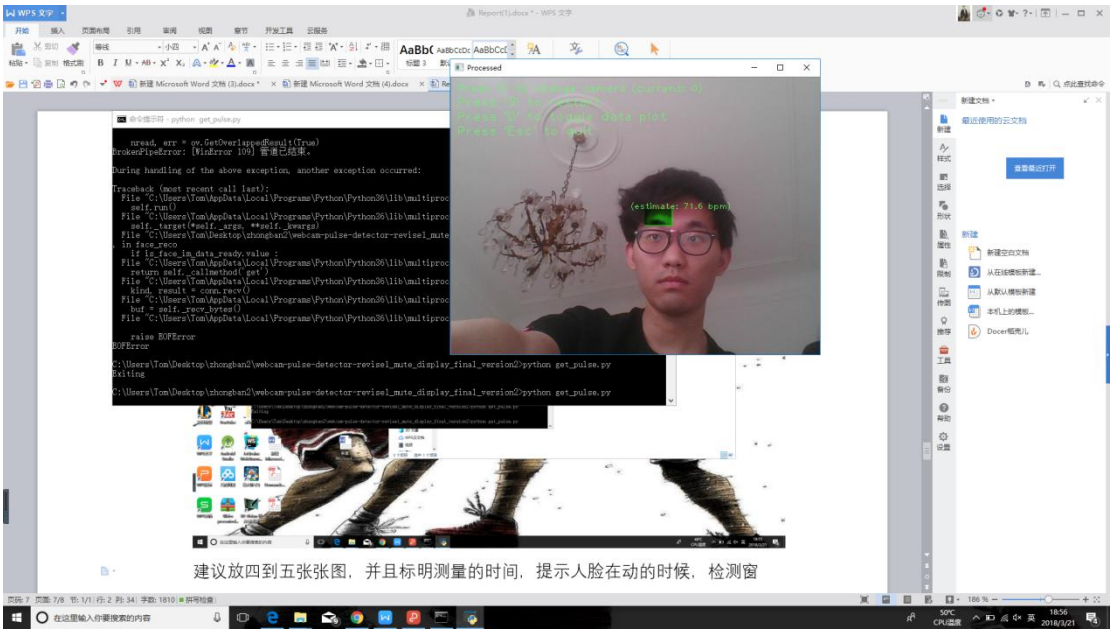

Figure. 6 Head vertical movement(0:09)

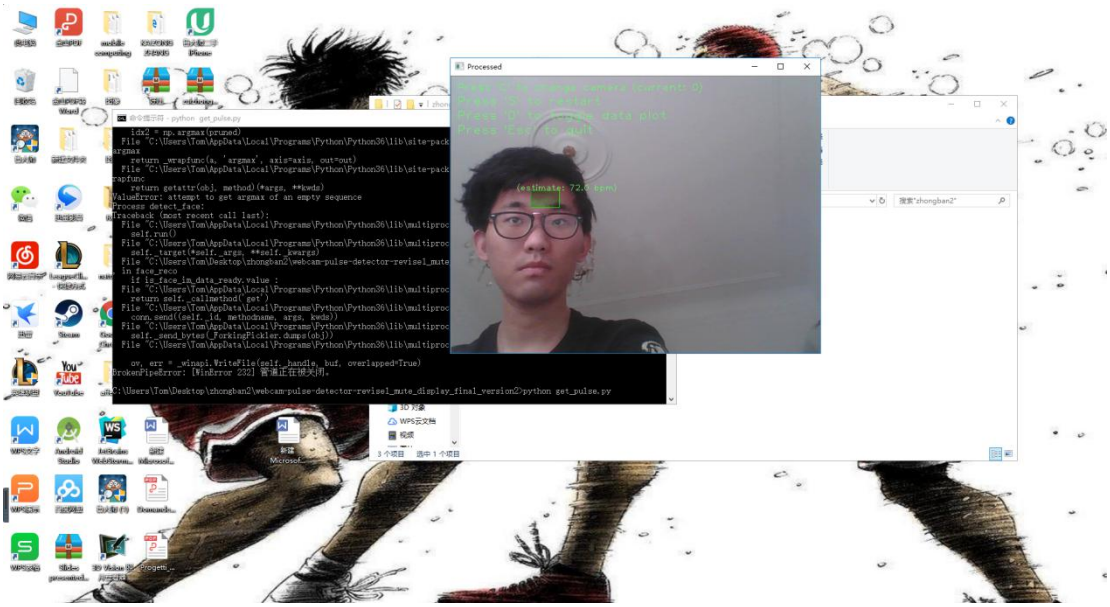

Figure. 7 Head Horizontal movement(0:13)

According to test results, as the face moves, the detection window ROI also changes, So it is concluded that the detection window is self- adaption.

4. Discussion and conclusion

In this report, I demonstrate a web-camera pulse detection application. Comparing with the original one, the ambient light change is taken into consideration to increase the detection precision. Moreover, the motion movement is detected in a separate running process. The update of the movement information is nearly in real time and can be adjusted with different system. I think we enhance the original program with more robust features.

Reference

[1] Hertzman A B. Photoelectric plethysmography of the fingers and toes in man[J]. Proceedings of the Society for Experimental Biology and Medicine, 1937, 37(3): 529-534.

[2] Cheang P Y S, Smith P R. An overview of non-contact photoplethysmography[J]. Dept. of Electronics & Electrical Engineering, Loughborough University, LE, 2003, 1(1).

[3] Hummler H D, Engelmann A, Pohlandt F, et al. Accuracy of pulse oximetry readings in an animal model of low perfusion caused by emerging pneumonia and sepsis[J]. Intensive care medicine, 2004, 30(4): 709-713.

[4] Trivedi N S, Ghouri A F, Shah N K, et al. Effects of motion, ambient light, and hypoperfusion on pulse oximeter function[J]. Journal of clinical anesthesia, 1997, 9(3): 179-183.

[5] Verkruysse W, Svaasand L O, Nelson J S. Remote plethysmographic imaging using ambient light[J]. Optics express, 2008, 16(26): 21434-21445.