FairyTales – Fairly Detailed

Shao-Hua Sun[1], Yao-Hung Tsai[2], and Chi-Wen Cheng[3]

Department of Electrical Engineering, National Taiwan University

No. 1, Sec. 4, Roosevelt Road, Taipei, 10617 Taiwan (R.O.C)

[1]b99901011@ntu.edu.tw

[2]b99901152@ntu.edu.tw

[3]b99901041@ntu.eud.tw

Abstract— **This document gives a detailed explanation about the design of our product and the implementation of it. The name of our product is FairyTales, which is a visual assistant with respect to some extending function such as taking photos and real-time handwriting. The functionalities of FairyTales are realized through DE2-115 board which is a powerful equipment designed for to create, implement, and test digital designs using programmable logic.**

***Keywords*— FairyTales; auxiliary eyeglasses; augmented reality (AR); vision system; DE2-115; Innovate Asia; Altera**

I. INTRODUCTION

Nowadays, visualization is the most simple but direct way we contact with the outside world. That is, perceptions, feelings etc. can easily be transmitted through human eyes. However, the sight for human can't be assured to be precise, and this defect often misleads us to unnecessary mistakes and errors. For instance, lacking enough brightness at night will be a serious problem for pedestrians; furthermore, in some situations, we hope to increase the contrast in colour to have a clear sight. Therefore, we wish to design a product that can resolve the upper problems for a better vision for human. Additionally, the function of immediate snap shot and browsing photos prevent us from regretting the fleeting scene in our life; the function of real-time handwriting make us able to record any information in the world. We name our product "FairyTales", which is "Face And I Refreshed You, Take A Live Efficient Shot", and we believe it is a cutting edge product.

The function of our product is realized on DE2-115, and we give a detailed introduction to our design. In the future, our product will build in an IC chip inside eyeglasses along with a small LCD display. There will be a micro camera beside the eyeglasses to film the scene, and it displays on the LCD display with icon interface.

The method we control our product is via fingertip and gestures rather than voice. With your fingertip pointing to the icon maintaining a few second, it will be recognized as a "click" action. Along with various gestures, we can operate the product in a more clever way. The trivial part will be discussed in section IV.

In this paper, there will be five sections, and they are introduction, related products, functionality, implementation, and conclusion, respectively.

II. RELATED PRODUCTS

Google glass and digital camera are the two famous products that in some aspect are similar to FairyTales can be found in the market. Both of them have their strengths but their weaknesses are required improvement. In the following part, we compare the two products with ours.

A. Google Glass

Google glass is designed for instant sharing images, video, and audio via Internet. The powerful work Google Glass manifests impresses human beings and gives us a newly feelings of information exchanging toward the world. The method Google Glass control the wearable computer is via voice. However, audio signal processing is not yet mature in the present technology, and this often cause misunderstandings between our intuitive and realization of the computer. Furthermore, lacking support of numerous languages is another serious problem, and this limits it to be a worldwide product. The difference in our product is that we communicate with computer via fingertip and gestures detection, and this asserts a more simple but strict approach to utilize the functions.

B. Digital Camera

Invention of digital camera makes us easy to capture memories or capture data. We can take snapshots of our friends or document our family at everywhere, and we can use it to help us remember things such as taking photos of the billing information. However, even though we can do lots of things with a digital camera, there are still some drawbacks we can't neglect. We may often encounter this situation: the moment we would like to capture is transient, so the best time for the photography is too short to grasp. We are not able to always get ready when the opportunity is arriving, and taking an immediate photography is essential. Additionally, having the handwriting in the photos during taking pictures is much more convenient than editing them afterwards. FairyTales not only enables us to photo promptly, but also allows us to record the information through real-time handwriting in the photos.

Comparing with the above two products, FairyTales manifests the conveniences which they can't offer. FairyTales

(a) (b) (c)

Fig. 1 Interfaces in different modes. (a) Initial (b) Interface (c) Display modes

aims at providing a human-friendly and creative usage for the users to better the life experience.

III. FUNCTIONALITIES

FairyTales realizes several fantastic tasks. At this part, we're going to introduce each of these functionalities and demonstrate how it works. At **section A,** we show how to launch the product. **Section B** shows how to adjust saturation as well as brightness. **Section C** presents how to do handwriting when you want to record something about the picture. At **section D**, we explain how to take a snap with FairyTales. The final **section E** shows how to browser the saved pictures.

There're three different modes in FairyTales, and they attribute to different functionalities and interface. In Fig.1 lists interfaces in different modes.

C. Launch

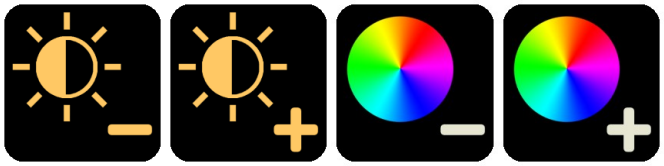
The upper right corner "start" icon is the only user interface when the product starts. Click the "start" icon, and other operating icons would pop out on the screen. If clicking upper right corner "start" icon again, the user interface would be back to the initial one, so that users can decide whether to show the icons as their wish. Besides, when clicking the screen, there will be bright spot at the pointing location. This function shows users if they really click the icons or other position.

(a) (b)

Fig. 2 (a) Launch (b) Close icons

D. Brightness/Saturation Adjustment

Users can adjust screen's saturation and brightness according to different conditions. These icons are on the lower right corner of the screen. The right ones are for brightness and the left ones are for saturation. Users can know the adjustment value by the bars showing above, both have 16 scales, from 0 to 15.

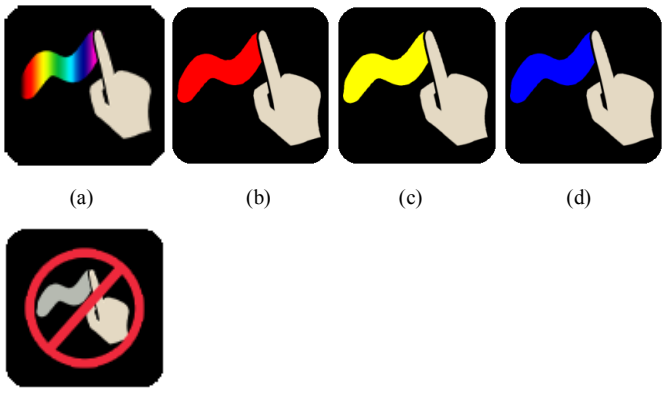


(a) (b) (c) (d)

Fig. 3 (a) Brightness minus (b) Brightness plus (c) Saturation minus (d) Saturation plus icons

E. Handwriting

The lower part of the screen would pop out a handwriting box when click the "handwriting" icon on the left side. There are three kinds of colour for handwriting, including red, blue and yellow, and the red one is the default one. Users can choose the colour and write with hands in the box. If they aren't satisfied with their result, they can delete it by clicking the "delete" icon on the left side.



(a) (b) (c) (d)

(e)

Fig. 4 (a) Handwrite (b) Red pen (c) Yellow pen (d) Blue pen (e) Shut off icons

F. Snap

If users touch the "snap" icon on the lower left corner, the icon would shimmer and take a snap after 2 seconds. In addition, users can also take a snap with gesture. Note that the gesture should occupy whole screen so that it can be detected. And the handwriting can be record via snapping.

Fig. 5 Snap icons

Fig. 6 Snap gesture

G. Display Mode

Users can change to display mode by clicking the "display" icon on the upper left corner. In the display mode, we can turn to next page or previous page by waving hand left to right or right to left. If the picture is the last one, and users wave arms left to turn to next page, the picture would be back to the first one like a circular ring. If users blank the camera lens, system would turn back to interface mode.

Fig. 7 Display icons

Fig. 8 Schematic drawing of hand-waving

IV. Implementations

In this part, we precisely introduce how we design FairyTales and how we implement its functionalities. The rest of this part is organized as follow. We briefly mention FairyTales's operation on different modes in **Section A**. **Section B** presents our framework of sensing icons. **Section C** describes the details about brightness and saturation adjustment. **Section D** and **Section E** explain how we implement snap with sensing icon and intuitive gesture, respectively. **Section F** presents how we execute FairyTales's functionalities in display mode. Implementation of real-time hand-write will presents in **Section G**, followed by the conclusions of this paper.

A. State Control

To explain our system's architecture; therefore, at the beginning, we mention FairyTales's different modes and functionalities of these modes. To properly prevent different modes from confusion, we apply two signals to take control. These two control signals named ***change_mode*** and ***write_mode***, respectively. In the following, we simply introduce these two control signals and the modes administered by them.

1) *Change_mode*: This signal manages three modes: initial mode, interface mode, and display mode. Fig. 1 illustrates three modes. The control signal ***change_mode*** manages these three modes by sensing icons and gestures. Besides, ***change_mode*** not only enable or disable operation of sensing icons and gestures, but also determines which icons to be show in different modes. With 2-bit long ***change_mode***, we separate three mode as followed:

change_mode[1] == 1'b0 -> initial mode
change_mode == 2'b10 -> interface mode
change_mode == 2'b11 -> display mode

Switch between these three modes is so intuitive. Click the "start" icon at the upper right corner to switch between initial mode and interface mode. Click the "display" icon at the upper left corner to switch to interface mode. Blank the camera lens with your finger and we can switch to display mode.

2) *Write_mode:* As it's name, this signal determine whether user can utilize handwriting. However, ***write_mode*** can only facilities under interface mode (***change_mode*** == 2'b01). Fig. 9 and Fig. 10 illustrat the two modes. With 1-bit long ***write_mode*** we separate two mode as followed:

write_mode == 1'b0 -> non-handwrite mode
write_mode == 1'b1 -> handwrite mode

Switch between these two modes is so intuitive. Click the "colourful finger" icon at the lower left corner would switch to non-handwrite mode. Click the "forbidden finger" icon at the lower left corner would switch to handwrite mode.

Fig. 9 Non-handwrite mode

Fig. 10 Handwrite mode

B. Sensing Icons

To fulfil our demand of touching buttons above the glass, camera need to sense the vision and determine whether an certain point is touched or not. However, we adopt a genius approach to avoid from consuming enormous memory space to save every pixel of a series of frame (at least 800*600*3*10*n bits, which n is the number of a series of frame). In the following, we briefly explain our implementation.

First, we divide every frame into forty-eight 100 pixels*100 pixels patches to fit our icon size. Schematic drawing is in Fig. 11.

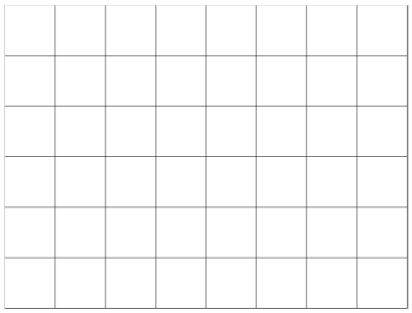


Fig. 11 Schematic drawing of divided patches

Every patch can at most be assigned a sensing icon. So the question is how to determine whether a certain patch is touched or not. In practical, we do not need to detect every point of patches. It's making constrain too strength, and also waste too much memory space. Therefore, we only take the center of patches as samples. Our schematic drawing of sample points is in Fig. 12.

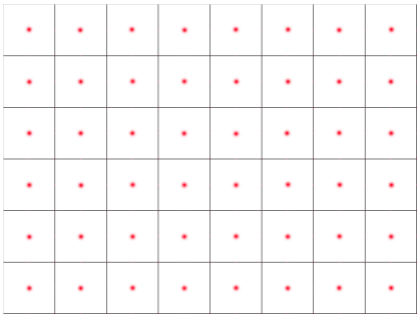


Fig. 12 Schematic drawing of sampling points

Any patch and any other else patch are independent. Once a patch's center detects the attendance of finger, this event would be regard as this sensing icon (if there is one) being touched.

As mentioned above, the issue of sampling is bring to close. However, if we consider only a frame to determine whether a certain patch is touched or not, the environment noise would lead to a terrible problem.

The approach to solve this problem is to consider sequences of frame. We chose to record forty-eight bits (1 means touched, 0 means non-touched) of continuous fifteen frames. Finally we AND these fifteen register of forty-eight bits together to obtain the final answer of whether a certain patch is touched or not during this time interval. The schematic drawing of operation is in Fig. 13.

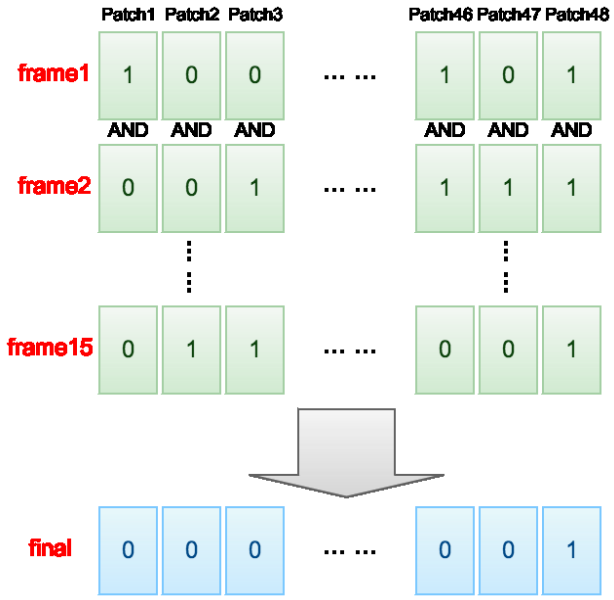
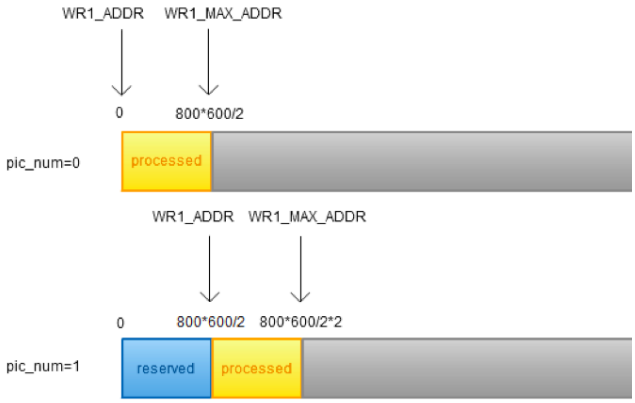


Fig. 13 Schematic drawing of frame-sequence operation

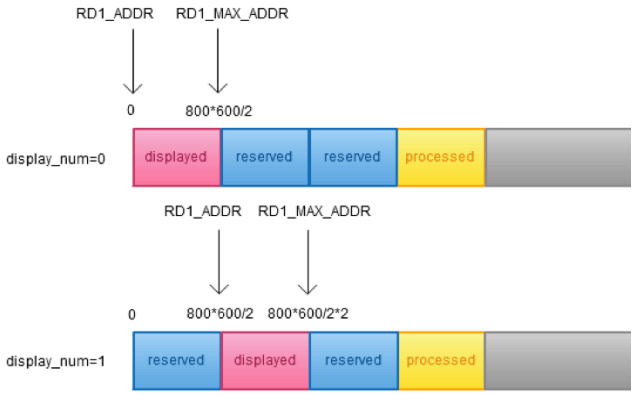
As a consequence, the register "final" would be the indicator to decide if sensing icons are touched. Different control signals would be triggered by different touched sensing icons, and different operations would be triggered be

different control signals. The control signals are listed as followed:

b_lum_add = ~change_mode[1] || KEY_signal || ~final[39];
b_lum_sub = ~change_mode[1] || KEY_signal || ~ final [47];
b_sat_add = ~change_mode[1] || KEY_signal || ~final [38];
b_sat_sub = ~change_mode[1] || KEY_signal || ~final [46];
b_screen_shot = ~change_mode[1] || KEY_signal || ~final [40];



(a)



(b)

Fig. 14 Schematic drawing of SDRAM memory space

b_write_green = change_mode[1] && (~write_mode || change_mode[0] || KEY_signal || ~o_isFinger[33]);
b_write_blue = change_mode[1] && (~write_mode || change_mode[0] || KEY_signal || ~o_isFinger[41]);

The ***KEY_signal*** is made according to the signal that generated by DE2-115 board when the buttons on the board are pressed on. Besides, ***change_mode*** plays a role of supervisor that determine which signal is valid to be generated depend on different modes.

By means of above procedures, we can easily use finger to press on sensing icons above the glasses.

C. Brightness/Saturation Adjustment

Because of the difference of implementations, we separate this section into two parts to explain how we design the system of luminance adjustment and saturation adjustment, respectively.

1) Brightness:

According to concepts and principals of image processing, we know that the so-called "brightness" of a pixel is definite as followed:

$$Y = 0.299*R + 0.587*G + 0.114*B \quad (1)$$

By this formula, we can not only convert RGB values of a pixel to brightness but also know that we can increase the brightness of a pixel by increasing the RGB values.

Therefore, our implementation is based on this concept. We declare a register named "lum", which is 4-bit ling. And then the luminance adjustment is as followed:

output_R = input_R + lum * lum_scale
output_G = input_G + lum * lum_scale
output_B = input_B + lum * lum_scale

The parameter "lum_scale" is determined by try-and-error, and it's most proper value is about 50.

Moreover, the RGB values above are all 10-bit long register, so these values range from 0 to 1023. To prevent RGB values from overflowing, we set the value that more then 1023 to be 1023.

2) Saturation:

According to concepts and principals of image processing, we know that the so-called "saturation" of a pixel is definite as followed:

$$V = 0.5*R - 0.419*G - 0.081*B + 128 \quad (2)$$

By this formula, we can not only convert RGB values of a pixel to brightness but also know that we can increase the saturation of a pixel by increasing the R value and decreasing the G value and B value.

However, in order to simplify the computation, we approximate the formula above to:

$$V = R - G \quad (3)$$

Therefore, our implementation is based on this concept. We declare a register named "sat", which is 4-bit ling. And then the saturation adjustment is as followed:

output_R = input_R + lum * lum_scale
output_G = input_G + lum * lum_scale - sat * sat_scale

output_B = input_B + lum * lum_scale + sat * sat_scale

The parameter "sat_scale" is determined by try-and-error, and it's most proper value is about 20.

Moreover, the RGB values above are all 10-bit long register, so these values range from 0 to 1023. To prevent RGB values from overflowing, we set the value that more then 1023 to be 1023 and set the value that smaller then 0 to be 0.

D. Snap with Sensing Icon

Because of the difference of implementations, we separate this section into two parts to explain how we design the system of luminance adjustment and saturation adjustment, respectively.

We knew that SDRAM would buffer the newest frame every time. Thus we decide to save picture by reloading new address, which is next to the processed one. Remember that both read and write addresses are the same when snapping (interface mode in other word), so both needs to be reloaded simultaneously. Note the reloading needs to be executed between one frame and another, preventing picture from aliasing. More details are shown in Fig. 14 (a).

Now we can save pictures. Besides, if we consider how to read the saved pictures, the idea is similar to the previous one. For example, if we have snapped three pictures already, and now we want to see these pictures, the read address will jump to a certain address (first picture at first time by default). Note that the write address doesn't change along with the read address. More details are shown in Fig. 14 (b).

E. Snap with Gesture

To prevent users finger from being captured on the screen and let the product being more user-friend, we add a 2-second delay from clicking "snap" icon to FairtTales really taking a snap. The controlling finite state machine is shown in Fig. 15.

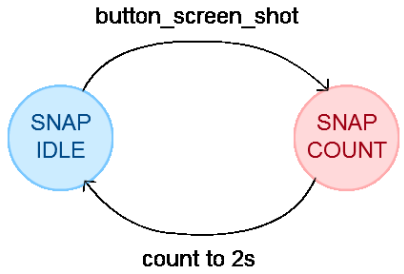


Fig. 15 Schematic drawing of state of gesture detecting

In order to knowing if we press the icon or not, we use the signal **snap_counter**[23] in state SNAP COUNT to control the icon shimmering. Therefore, if we see the icon shimmering, we know that FairyTales is about to snap after 2 seconds.

Besides snapping by clicking the sensing icon, we can also snap by a specific gesture. The idea is from Pranav Mistry, a MIT orator in TED. Now we implement one of his ideas in our product.

However, we don't want to wear fingerstalls. The reason is we think it's not user-friendly enough if there're too many fingerstalls. Thus we choose to recognize complexion instead of detecting fingerstalls.

The algorithm is simple. First we divide the screen into a 3 by 3 grid as in Fig. 16.

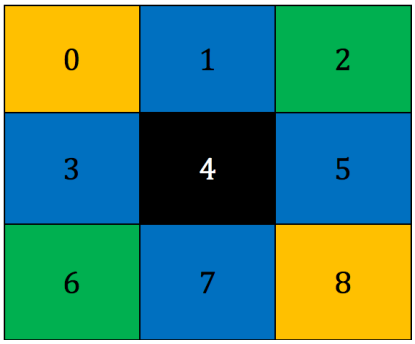


Fig. 16 Schematic drawing of divided screen

We find that if we pose the gesture, two palms are surely diagonal, that is 2&6 or 0&8. Note that patch 4 must not be complexion and the fingers (both thumb and index finger) are at patch 1, 3, 5, 7. However, neither the fingers are big enough to occupy the patch, nor the fingers are at certain location. As a consequence, the condition is looser than the others. In conclusion, when sensing complexion at the condition of (0&8|2&6)&(!4)&(1&3&5&7), snap icon will shimmer and FairyTales will be about to snap after 2 seconds.

F. Display mode with Gesture

As mentioned above, SDRAM writing address can be changed at any time, but the reload signal should be updated during the interval between a finished frame and next frame.

Though it's very easy to do this work by just pressing on the button, we designed intuitive FairyTales display mode that change viewed picture just with your hand throwing.

In order to detect user's hand, we adopt simple version of complexion detection algorithm to turn every frame into a binary frame (1 means skin, 0 means non-skin). We also designed two finite-state machines (FSM) to record the state of hand throwing. FSM description is drawn in Fig. 17.

We choose the event "TurnRight" as example to explain. Initial state called "IDLE". FSM at "IDLE" waits to be triggered. By detecting user's hand attending at left side of screen, FSM then move to the next state "R1". FSM at "R1" is expecting user's hand appearing at the middle of screen and then move to the next state "R2". Likely, when user's hand finally arrives at right side of screen, FSM then move to the next state "R3". At the next clock falling edge, FSM move to next state "TurnRight", and change SDRAM writing address at the same time. As the writing address of SDRAM being changed, the viewed picture would be change. Finally, FSM move to "IDLE" state and begin to wait to next "TurnRight" event.

As mentioned above, "TurnLeft" event operates the same mechanism as "TurnRight" event.

With "TurnRight" and "TurnLeft", user can intuitively changed picture which displayed on the screen as he or she want.

Moreover, we designed system to count the number of black pixels in each frame on display mode. Once the number of black pixels is more then a threshold, FairyTales would change display mode into interface mode.

G. Handwrite

We design the real-time handwrite algorithm in the module ***RAW2RGB***. The reason we design it in ***RAW2RGB*** not in other module is because the handwriting must be memorized in SDRAM not only displayed on the screen, so the prior module is needed rather than the posterior module, and the process of our module can be illustrated in Fig. 18.

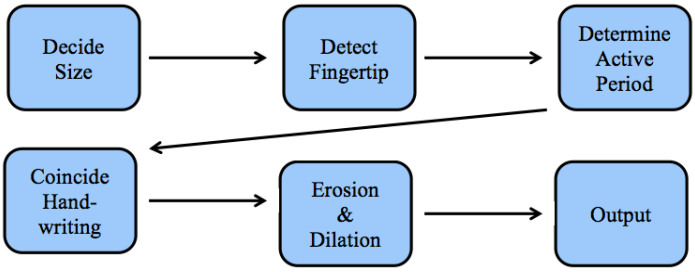


Fig. 18 Process of handwrite module

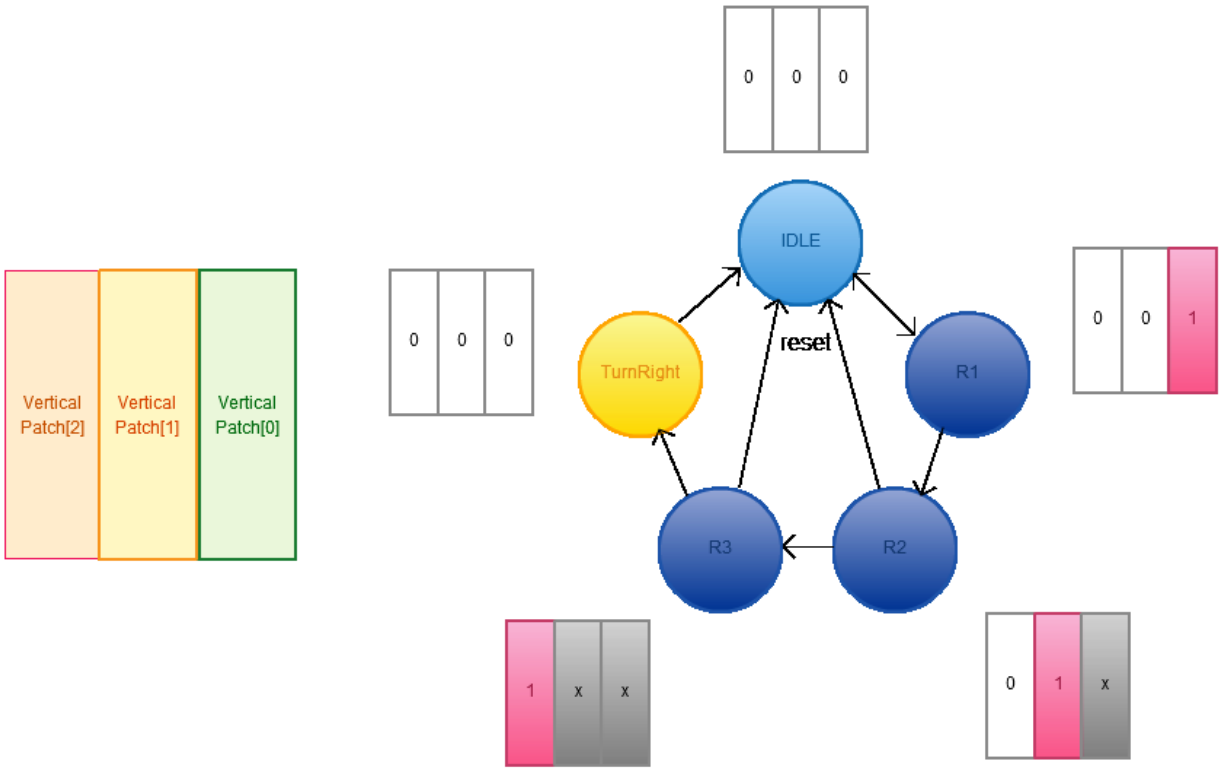


Fig. 17 FSM description

1) *Decide Size*: First, we select the size of our output handwriting patches. If we choose the size to be 10 pixels, than the handwriting patches will be bigger; otherwise, if we choose the size to be 5 pixels, than the patches will be smaller. Usually, the smaller patches, the smoother handwriting would be. In our module, we choose the size to be 5 pixels.

2) *Detect Fingertip*: In the 5*5 pixels, we define that if the middle 3*3 pixels are detected to be green, than we say that the patch is chosen; otherwise, we say that the patch isn't chosen. In Fig. 19, if the 3*3 pixels are detected in the 5*5 pixels, than the patch is considered to be an effective patch.

3) *Determine Active Period*: Different from the setting of the sensing icon, the active period of our handwriting function is 1 frame, not 15 frames. That is, if the patch viewed as the effective patch within 1 frame, it would be coloured.

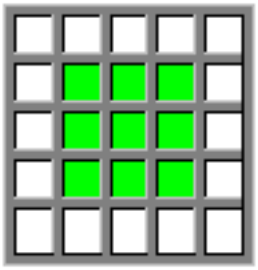


Fig. 19 5*5 pixels patch with medium 3*3 pixels detected

4) *Coincide Handwriting*: There are two registers in our module, ***tmp_handwrite*** and ***final_handwrite***, respectively. These two registers represent the above patches. At the beginning, all the patches in ***tmp_handwrite*** are all given value 1. If one patch not considered as green, than that patch in ***tmp_handwrite*** would be given background colour. After one frame is finished, we coincide ***tmp_handwrite*** and ***final_handwrite***. Fig. 20 explains our algorithm.

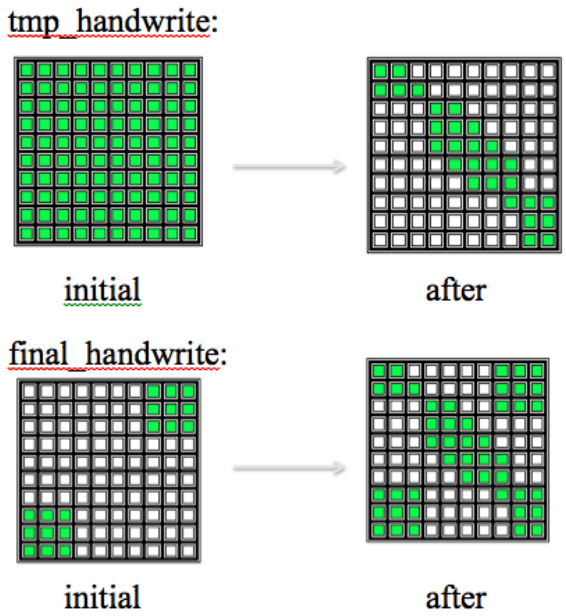


Fig. 20 Coincde tmp_handwrite and final_handwrite

5) *Erosion & Dilation*: After ***final_handwrite*** finishes its change, we refine the value with erosion and dilation. The adjustment would be: if one patch has five surrounding coloured patches, than whether or not this patch is coloured, this patch would be regarded as coloured patch. On the other hand, if one patch has at most two surrounding coloured patch, than it would be regarded as non-coloured patch. After we finish erosion and dilation, the values of pixels in ***final_handwrite*** are given to ***processs_handwrite***. Fig. 21. explains the process.

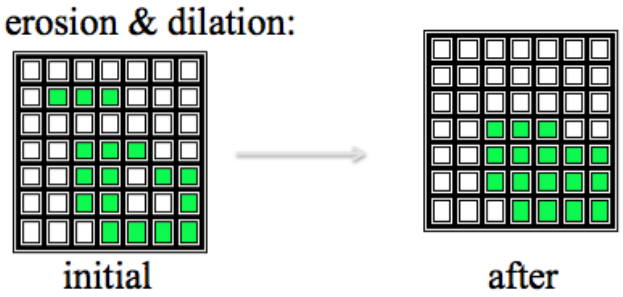


Fig. 21 Erosion and dilation after final_handwrite

6) *Output*: Due to the fact that camera processes the image in a sequential manner, we can't compute both ***final_handwrite*** and ***processs_handwrite*** in the demand of real-time. That is, we have to compute them in different time sequence. We compute ***final_handwrite*** in the bottom of A-frame, and we compute ***processs_handwrite*** in the upper half of (A+1)-frame. After the computation of ***processs_handwrite*** is finished, the result is given to ***o_handwrite***, which is the output on the screen. In this mean, we can assert to have a real-time handwriting rather than a delayed one. Fig. 22. explains the process.

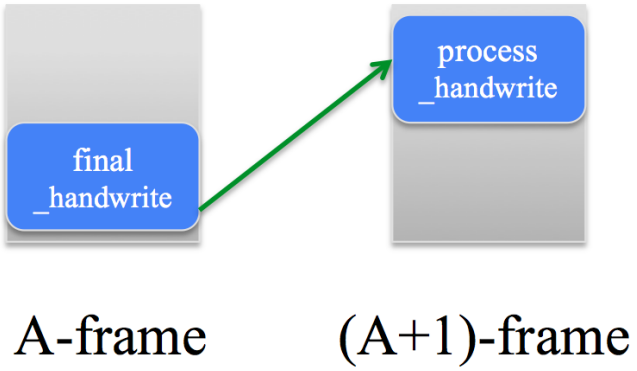


Fig. 22 Compute final_handwrite and process_handwrite at different time

V. Conclusions

The product we design aims at altering humans' vision and realizing real-time hand-wring and taking photos. We compare our product to Google glass and digital camera and also give a detailed explanation to our product. In our design, we utilize numerous techniques especially in hardware to apply on various functionalities. Wholeheartedly, we wish our product may ameliorate humans' life and we would never stop improving it.