CS339 Team 11 Project: ChromaCode

Xuehan Sun, Xingrui Yi, Haiwei Deng and Wendong Bi

December 2018

1 Introduction

In this project, we try to embed data imperceptibly into regular videos while remaining unobtrusive to human viewers. If enabled, it will reform various existing applications and foster new possibilities. Firstly, it will change the digital advertising industry by delivering advertising content without tampering primary video. Secondly, it will bring more interesting gameplays when players can perceive multi-dimensional contents for engagement with high throughput and low latency. The main goal of our design is to implement imperceptible, high rate, and reliable communication. Our project includes an outcome-based adaptive embedding scheme, which adapts to both pixel lightness and regional texture and a concatenated code scheme for robust coding.

2 Inspiration

We watch a lot of videos every day in this digital era. Billions of videos are generated and broadcasted through TV, Tik Tok, WeChat, Youku and so on. As digital marketing evolves, people continue to see trends favoring video, especially in the era of Mobile Internet. However, there exists a problem that such videos are mainly for viewing only, which means we can't get information other than video content while watching the video. Many video providers put QR code in the lower right corner of videos and hope viewers scan the QR code for more information such as advertisement, introduction of background and information of relevant cast. But the QR code will affect viewing experience and many viewers are very angry about this. we hope to make advertisements and other information invisible to viewers, to achieve simultaneous viewing and communication without affecting viewing experience.

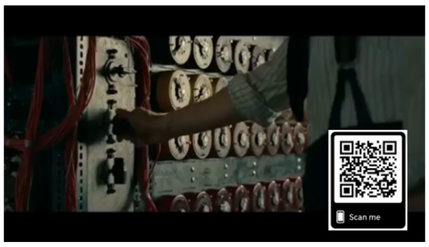


Figure 1 video with QR code

Figure 2 video without QR code

3 ChromaCode Design [5]

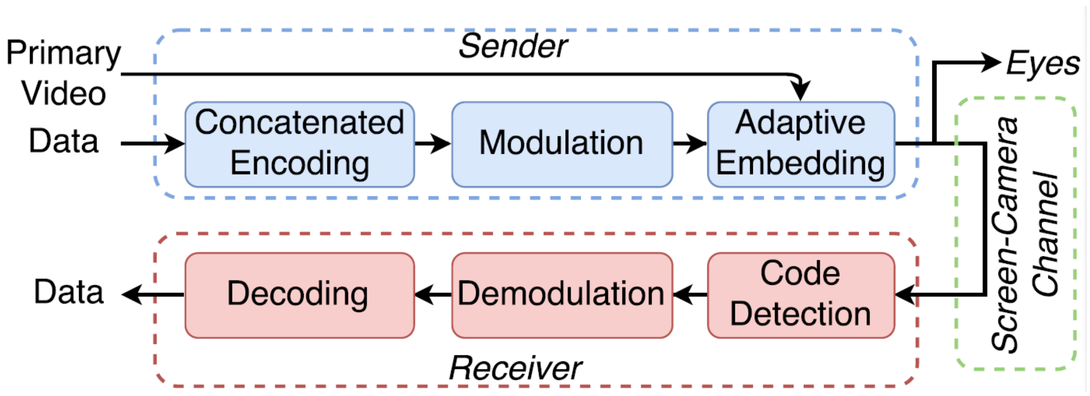


Figure 3 ChromaCode overall architecture

Figure 3 illustrates the overall work flow of *ChromaCode*. As it is said in the introduction part, the full implementation of *ChromaCode* can be divided into two parts: the sender and the receiver. ChromaCode employs commodity screens as transceiver and off-the-shelf smart phones as receiver. In the Sender part, we encode the video, module it, and then give an adaptive embedding into primary video. While watching, human eyes receive the original video, while with screen-camera, the receiver receives lightness changes and decode them to get the information.

3.1 Sender

As the blue part shown in figure 3 The sender end's task is to embed our data frame into the original video implicitly. The sender takes primary video together with secondary side information which is referred as video and data respectively hereafter as inputs. Data are first encoded by the concatenated error correction coding scheme. The encoded data are then modulated as an imagery code, which is adaptively embedded into the primary video in a visually unobtrusive way, producing the ultimate output at the sender. And then it will be displayed on the screen and streamed over the screen-camera channel.

3.1.1 Invisible Embedding

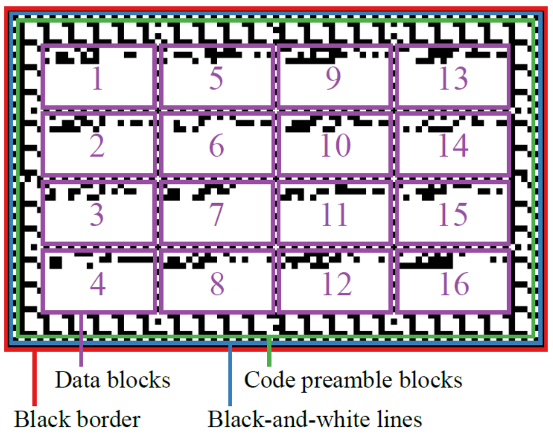


Figure 4 Data Frame

In order to invisibly embed data frames, in the form of image codes as shown in figure 4, into normal videos without affecting the viewing experience, we embed data frames by altering the lightness of carrier video, exerting a series of time-variant high-frequency lightness changes that are imperceptible to human eyes yet detectable to cameras.The generating details of data frame will be illustrated in the Implementation part.Each data frame is rendered by a pair of successive frames with contrary lightness changes $\pm\Delta L$, increasing the pixel lightness of the first frame by a certain value and decreasing that of the subsequent frame by an identical amount.This results in two complementary frames that visually cancel out each other's lightness changes thanks to the flicker fusion property of HVS.

Yet they still can be captured by cameras with high capture rate, allowing extraction of the carried data bits.

3.1.2 Color Space

We compare differences between non-uniform and uniform color spaces with their hue and lightness palettes. There are three annuluses in each color palette. The middle annulus has lightness L. The outer and inner annuluses have lightness $L + \Delta L$ and $L - \Delta L$. For the palettes in non-uniform color space, color differences are not uniform. For example, color differences are more obvious here. While color differences are less obvious here.

For palettes in uniform color space, The color differences are more perceptively uniform.

Uniform color spaces suit human system better. Therefore we choose uniform color spaces for lightness modification.

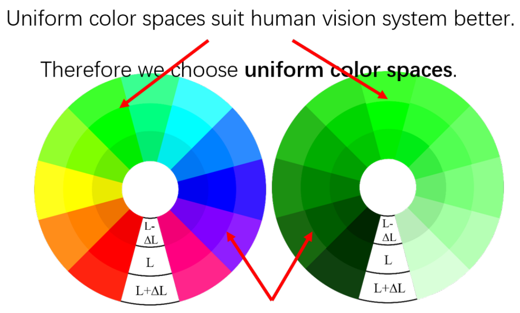

Uniform color spaces suit human vision system better.
Therefore we choose **uniform color spaces**.
L-ΔL
L
L+ΔL
L-ΔL
L
L+ΔL

Figure 5 Color Space Selection

3.1.3 Spatially Adaptive Embedding

CIEDE2000 color difference formula inspires us that, instead of using a fixed lightness changing amount, we should choose an expected color difference amount.And calculate desired lightness changes using CIEDE2000 formula. This leads our outcome-based lightness change derivation based on lightness of single pixel.

$$\Delta L_1^*(i,j) = k_L \left[1 + \frac{0.015\left(L^*(i,j) - 50\right)^2}{\sqrt{20 + \left(L^*(i,j) - 50\right)^2}} \right] \Delta E_{00} \tag{1}$$

What's more, as previous studies show that lightness changes are more noticeable in smooth regions, while less noticeable in textured regions. We further calculate regional texture complexity. And decrease lightness changing amount in smooth regions basing on their texture complexity, to get our final lightness changing amount.

$$\begin{aligned} Texture(i,j) &= \frac{\text{Contrast }(i,j)}{S(i,j)} \\ \alpha(i,j) &= \frac{Texture(i,j)}{TextureMax} \times (1-k) + k \end{aligned} \tag{2}$$

3.2 Receiver

On the receiver side which is shown in the red part of figure 3, a video clip is captured by camera. The receiver first extracts the embedded imagery code from the captured video frames by code detection. The extracted code is then demodulated and decoded, recovering the embedded bits. During the entire receiving procedure, users can normally watch the video, without feeling affected by the screen-camera transmission.
For this part, our main challenge is to implement high rate and reliable communciation.

4 Our works

4.1 Generating the encoded video

In this part, we ultimate OpenCV 4.0 [1], the newly published open tools for video processing. According to the methods mentionend in paper [5], to ensure both stability and high throughput without disturbing human eyes, we need to consider both lightness and texture.

4.1.1 Lightness

As for the lightness, it is natural that we are more likely to be attracted to images that are brighter while ignore those daker images. But the interesting thing also lies in this, which is, though brighter images obtain more attention, less chances are that we actually notice these changes.

To make changes in lightness more specific, we use a new well-defined color space to ensure, the CIELAB 2000 color space. First we conduct a color space exchange, using the most sophisticated standard CIELAB has provieded.

Listing 1 Color Space Convert

```
void RGBToLab(unsigned char*rgbImg,int*labImg)
{
    long long X=(rgbImg[0] * 199049 + rgbImg[1] * 394494 + rgbImg[2] * 455033 + 524288)>> (
        big_shift);
    long long Y=(rgbImg[0] * 75675 + rgbImg[1] * 749900 + rgbImg[2] * 223002 + 524288) >> (
        big_shift);
    long long Z=(rgbImg[0] * 915161 + rgbImg[1] * 114795 + rgbImg[2] * 18621 + 524288) >> (
        big_shift);

    labImg[0] = Y>ThPara?((ScaleLT * LabTable[Y] - ScaleLC + HalfShiftValue)>>shift):((
        ScaleY* Y)>>shift);
    labImg[1] = (para1*(LabTable[X] - LabTable[Y])+HalfShiftValue+offset)>>shift;
    labImg[2] = (para2*(LabTable[Y] - LabTable[Z])+HalfShiftValue+offset)>>shift;
}
```

Then to fully control the deviation of lightness, we define color difference ΔE_{00} and recalculate each pixel's light-changing throught the following equation:

$$\Delta L_1^*(i,j) = k_L \left[1 + \frac{0.015\,(L^*(i,j) - 50)^2}{\sqrt{20 + (L^*(i,j) - 50)^2}} \right] \Delta E_{00} \tag{3}$$

Related Code is show below, first we encoding the image of information needed, the we recalculate each pixel's light channel, according to the information. Here we meet a problem which will be further explained in Section . The question is, how will can the error be and how much the information influences the original lightness. This is quite a survey and try process, this part consume us about a whole weekend. But sweat pays well, we later conquer it.

Listing 2 Lightness Calculation

```
void Lightness(InputArray _src1, double alpha, InputArray _src2, double beta, InputArray
    _dst, int startrow, int startcol) {
  Mat src1 = _src1.getMat(); //trans origin frame into Mat formula
  Mat src2 = _src2.getMat();
  Mat dst = _dst.getMat();
  double delta=1;
  src1.copyTo(dst);
  for (int i = startrow; i < src2.rows+startrow; i++) {
    for (int j = startcol; j < src2.cols+startcol; j++) {
      for (int k = 0; k < 1; k++) {
        double lightness = src1.at<Vec3b>(i, j)[k];
        double power = pow(lightness - 50, 2);
        double s = 1 + (0.015*power / sqrt(20 + power));
        delta = k_L * delta_E * s ;
        delta = delta / 3;
        dst.at<Vec3b>(i, j)[k] = saturate_cast<uchar>(alpha*src1.at<Vec3b>(i, j)[k] +
            delta * beta * src2.at<Vec3b>(i-startrow, j-startcol)[k]); // Calculate
            lightness through each pixel
      }
    }
  }
}
```

4.1.2 Texture

As has been mentioned above, the lightness changes in the texture are more unnoticeable to human eyes. This part of implementation is quite effort-requiring, for that the main method the paper provided is not that well-formed for texture detecting. It mainly calculates each pixel's neighbor pixel, and use the Contrast of its neighbor window to stand for the textures. The paper give a 9×9 region, to compare the main changes with the max one and calculate the relevant contrast.

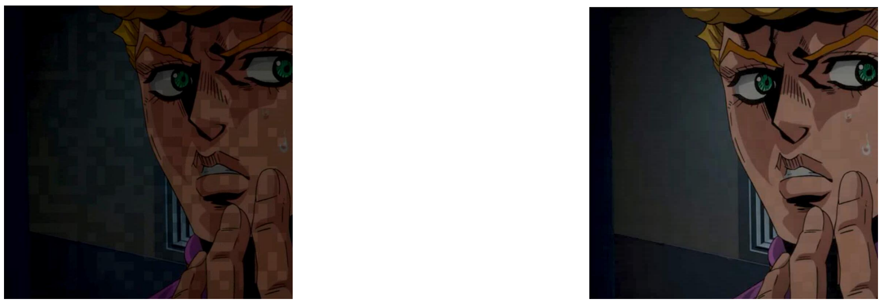

(a) Contrast (b) Ours

Figure 6 Comparison of Simply using Contrast with Our Methods

In the texture part, we actually think differently, because they do not give a convincing reasons for why they use Contrast. We have tested use lightness, pixel changes and contrast. We find that use the pixel's changes is a better one. Its main idea is to compare each pixel with their neighbor ones, and use the Hamming Distance or Euclid Distance to show the differences (actually Either two distance works pretty well).The comparison is demonstrated in Figure 6.

So in Figure 6 you can find that, contrast is not a that good one. This question will be further developed in Section 6. The implementation is show below:

Listing 3 Texture calculate

```
void Texture(InputArray _src1, double alpha, InputArray _src2, double beta, InputArray _dst,
    int startrow, int startcol) {
  Mat src1 = _src1.getMat();
  Mat src2 = _src2.getMat();
  Mat dst = _dst.getMat();
  for (int i = startrow; i < src2.rows+startrow; i++) {
    for (int j = startcol; j < src2.cols+startcol; j++) {
        WindowGenerate(i,j,dst,src1,src2);
        if( Max <= Distance(i,j,dst)
            Max = Distance(i,j,dst); //find the Max Distance in each frame
      }
    }
  }
  for (int i = startrow; i < src2.rows+startrow; i++) {
    for (int j = startcol; j < src2.cols+startcol; j++) {
        for(int k = 0 ; k < 1 ; k++){
            dst.at<Vec3b>(i, j)[k] = saturate_cast<uchar>(dst.at<Vec3b> * Distance(i,j,dst
                )/Max; // Calculate lightness through each distance
            }
      }
    }
  }
}
```

4.2 Decoding with screen-camera

In this part, we are trying to design a simple app on Android devices by using Android Studio.

4.2.1 Overview

The whole code flow involves the brightness analysis of frame data returned by Camera in YUV format coded by byte stream. Then the lightness difference of two consecutive frames is calculated and normalized. Next, the result picture is obtained according to the corresponding threshold. Finally, we decode the result image to get the data we want.

In order to minimize the delay of decoding, resulting in low recognition rate of data and hope to extract data as soon as possible, we have adopted Algorithm 1.

Algorithm 1: Decoding with screen-camera

```
byte[][] $tmp\_data$;
int $count = 0$;
if $count \leq max\_count$ then
    $tmp\_data[i] = Camera\_frame\_message$;
    $count++$;
    Ask for another $Camera\_frame\_message$;
for $i = 0; i < count - 1; i++$ do
    if $tmp\_data[i] != null \ \& \ tmp\_data[i+1] != tmp\_data[i]$ then
        $out\_data$ = compareFrame($tmp\_data[i], tmp\_data[i+1]$);
        decode($out\_data$);
    $count = 0$;
```

4.2.2 Lightness acquisition

In this part, we would like to give a brief introduction to YUV color space. In YUV color space, the Y component determines the lightness of the color (referred to as luminance or luma), while the U and V components determine the color itself (the chroma). Y ranges from 0 to 255 in digital formats, while U and V range from -128 to 127 in signed digital form, or 0 to 255 in unsigned form [4].

Because the frame data returned by Camera is encoded in YUV format, we can extract Y information directly from byte stream to get the lightness information of each frame quickly.

Algorithm 2: Get Lightness

```
double[$width$][$height$] $brightMap$;
for $j = 0; yp = 0; j < height; j++$ do
    for $i = 0; i < width; i++, yp++$ do
        $first\_y = (0xff \ \& \ firstFrame[yp]) - 16$;
        if $first\_y < 0$ then
            $first\_y = 0$;
        $second\_y = (0xff \ \& \ secondFrame[yp]) - 16$;
        if $second\_y < 0$ then
            $second\_y = 0$;
        $brightMap[i][j] = first\_y - second\_y$;
```

4.2.3 Decoding

We read normalized lightness of each data cell in the same order as interleaving during encoding process, leading to a sequence of data values between [0, 1]. Then we obtain the corresponding decoded data according to the reverse operation of coding.

4.3 Problem and Conquer

4.3.1 Different Videos need different parameters

Problem Although this paper has actually proposed a new formula function as is shown in Equation 3, the choices for ΔE_{00} can significantly influence the quality for each video.In practice, generating different video need to re-choose these parameters. That means, the recalculation function may not work that well.

Solution: Propose A Training Part The way to reduce the reliability on these parameters, is to change these parameters according their original video type. This part probably can be conducted by some machine learning strategies, but due to lack of data set, we cannot afford to pay those training prices. I still feel that this training way can work well if we have enough human resources to label these videos. In the end, we turn to experimentalism, as is stated in the origin paper, to manually control each parameter for each video.

4.3.2 Lightness change caused by the refresh of the screen

Problem According to our actual test, we found that the lightness change caused by the refresh of the display screen will also be captured by the camera (even if the content of the display screen remains unchanged like Figure 7).

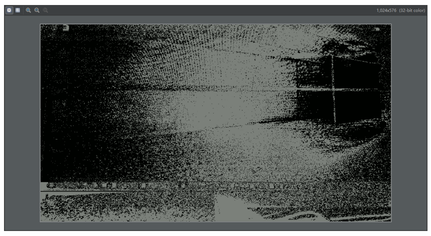


Figure 7 Lightness change of the unchanged screen

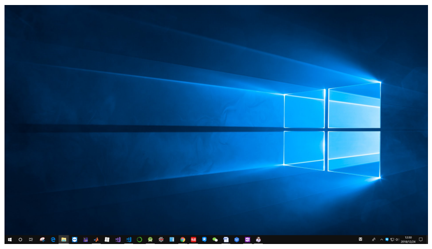


Figure 8 The original screen

Solution: Normalization In order to reduce unnecessary lightness effects caused by screen flicker, we tried normalization method to solve this issue. After demodulation, we have the received data frame with corrected lightness for each cell. Ideally, each positive lightness value shall represent bit '1' while negative represents bit '0' (or vice verse). In practice, however, due to screen-camera channel distortions, there may be significant lightness offsets, making this simple strategy in-feasible. To combat lightness distortions, we normalize the subtracted lightness values to [0, 1] and employ soft-decision decoding to leverage their tendency to bit '1' or '0'.

4.3.3 The impact of lightness of different frames

Problem Take QR Code for example, we want to show some outputs during the running time. We can see in Figure 9, the QR Code we have are influenced by different animation textures. The reason is the lightness change caused by the change of picture in two consecutive frames of animation.

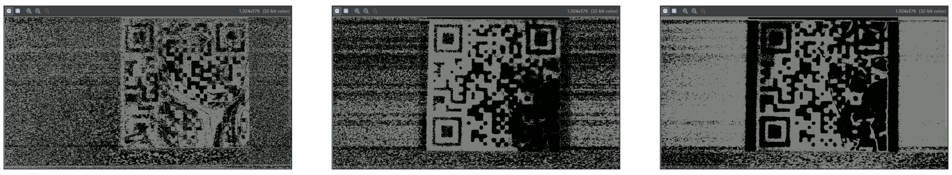

(a) pic1. (b) pic2. (c) pic3.

Figure 9 The Impact of Different Frames of Animation

Also, even though some outputs are very clear enough, it's just on the opposite sequence of differences, and it will be like Figure 10. But luckily, this situation can be solved by post-reflex treatment.

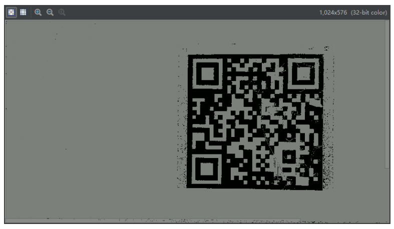
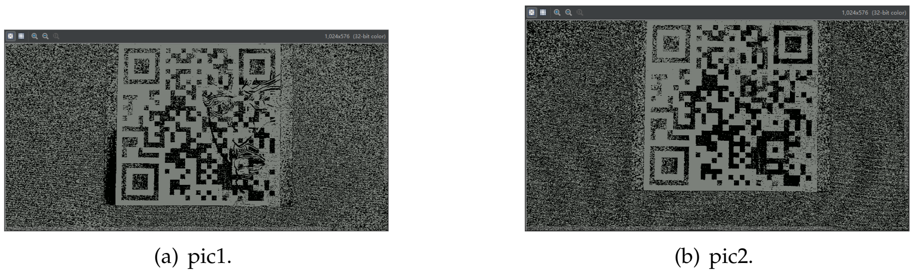


Figure 10 Opposite output

And we will also show some outputs with better situation in Figure 11. These outputs can be decoded by decoder.



(a) pic1.

(b) pic2.

Figure 11 Good outputs

Solution: Increase the frame rate If the refresh rate of the code is higher than the original refresh rate of the video, the impact of this problem can be reduced. So, increasing the frame rate can be a very simple solution. However, because the screen refresh rate dependents on the screen performance, updating the screen should be considered. Also, with the increase of the refresh rate, screen-camera needs higher acquisition rate. Due to the hardware limitations, we can't get further improvement.

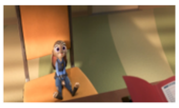
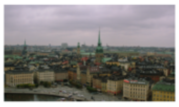
5 Result

We design a system that achieves all three goals on unobtrusive, high-rate, reliable screen-camera communication. As shown in the table [1], the results demonstrate its superior performance over previous schemes in terms of invisibility, throughput and other performance. ChromaCode is on the way of changing video advertising industry, game industry and other interactive application. Furthermore, we anticipate even more imaginary spaces for novel applications.

Table 1 comparison with previous schemes

	Texture [3]	HiLight [3]	Implicit [2]	Chroma [5]	Ours
Texture Detecting	✓			✓	✓
High throughput			✓	✓	✓
Colorful video		✓		✓	✓
High processing speed					✓
Better Measuring ways					✓

We not only implement each video using ChromaCode's encoding method, but also Embedding QR Code into other kinds of video for interaction with classmates during our presentation, so that they can download our app, and scan it in real transmission.

	Zootopia (Z)	Football (F)	GTA V (G)	Minecraft (M)	Town (T)
					
Texture	Plain	Textured	Textured	Textured	Plain&Textured
Switching	Gradual	Gradual&Sharp	Sharp	Gradual	Gradual
Luminance	Bright	Bright	Dark	Bright	Dark
Quality	SD	SD	HD	HD	SD

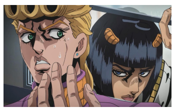
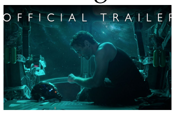
	JOJO	Avengers	Dark Souls	Christmas	Two Blossom
					
Texture	Textured	Textured	Textured & Plain	Textured	Plain
Switching	Sharp & Gradual	Sharp	Gradual	Gradual	Gradual
Luminance	Bright	Dark	Dark	Bright	Dark
Quality	HD	BD	BD	HD	SD

Table 2 Our Results

6 Beyond the Paper

6.1 Criticism

6.1.1 Too much redundant statement for color space selection

The paper gives about two pages to explain why the color space selection is so important that we initially believe that this part is of some significance, while the truth is this part may be provided to extend the paper to page-requirement. According to their own referrence for CIELAB, CIELAB has already had a well-formed function change from RGB to LAB, so whether you choose LAB or RGB, which color space is not important at all. If you change some parameters in CIELAB space, you can one hundred percent without any loss trans these in RGB space. So I am quite confused about why they spend so much effort to emphasize this.

Later I also implement this in RGB color space and can also be detected for the receiver. I feel that the main reason is part of the fact that you can have a more easily parameter adjusting debugging process. But because the original video is read in RGB forms, trans it to LAB consume redundant calculation time.

6.1.2 Consider the Texture without quite simple factors

It conforms out intuition that objects with more textures can hold a wider tolerance of lightness variance. But the method ChromaCode provides, frankly speaking, is so intuitive and kind of simple. By only computing the relevant contrast, the paper holds that difference from the Max Contrast can stand for texture difference. This also results in unnecessary computation, for implementing this needs extra comparison, window-size computing. But by using calculation form Hamming Distance or Euclid Distance, the workload for calculation reduces much but enhances stability in lightness.

6.2 New Methods

6.2.1 Lightness noise elimination

To eliminate these lightness noise, generally speaking, there are two ways: one is to elevate the quality of hard wares, enabling the screen for a better demonstration and the camera for a better high frame-rate detecting; the other is to increase the threshold when detecting lightness changes. Besides these two ways, I thing that there can also use some strategies when generating the video. As is acknowledged, detecting the movements in a video is not that hard in temporary times, using some famous neural networks in Computer Vision. But this trade-off for better stability also results in more calculation. When during real-time transmission, this part may can be tolerated by long-time detecting and if things come to High Quality, maybe pre-detecting is a better way.

6.2.2 New Measures for Texture Detecting

In Section 4.1.2, we hold that the methods raised by the paper simply using Contrast is not that satisfied, besides, it also need a lot of extra work to figure out the proper parameters. So we use Distance to measure each pixel's differences and complete our results. As is show in figure 6, this will not only provide a better embedding, but also enable us to make the changes in lightness a little more, thus contributing to more stable transmissions.

References

[1] Joel. Opencv 4.0. `https://opencv.org/opencv-4-0-0.html`.

[2] Tianxing Li, Chuankai An, Xinran Xiao, Andrew T. Campbell, and Xia Zhou. Real-time screen-camera communication behind any scene. In *International Conference on Mobile Systems, Applications, and Services*, pages 197–211, 2015.

[3] Viet Nguyen, Yaqin Tang, Ashwin Ashok, Marco Gruteser, Kristin Dana, Wenjun Hu, Eric Wengrowski, and Narayan Mandayam. High-rate flicker-free screen-camera communication with spatially adaptive embedding. In *IEEE INFOCOM 2016 - the IEEE International Conference on Computer Communications*, pages 1–9, 2016.

[4] Shiming Shi, Lingxue Wang, Wei-qi Jin, and Yuanmeng Zhao. Color night vision based on color transfer in yuv color space. In *International Symposium on Photoelectronic Detection and Imaging 2007: Image Processing*, volume 6623, page 66230B. International Society for Optics and Photonics, 2008.

[5] Kai Zhang, Chenshu Wu, Chaofan Yang, Yi Zhao, Kehong Huang, Chunyi Peng, Yunhao Liu, and Zheng Yang. Chromacode: A fully imperceptible screen-camera communication system. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, pages 575–590, New York, NY, USA, 2018. ACM.