Wen Liang(Wen_Liang@student.uml.edu)

1. a) In Total, there are two major steps:

Step 1: Generating a black-and-white sketch ;

Details:

First we need use the Median filter to reduce noise in the image and the edge detection filter----Laplacian filters to produce edges.

```
medianBlur(gray, gray, MEDIAN_BLUR_FILTER_SIZE);

Laplacian(gray, edges, CV_8U, LAPLACIAN_FILTER_SIZE);
```

The Laplacian filter produces edges with varying brightness, so to make the edges look more like a sketch we apply a binary threshold to make the edges either white or black.

```
threshold(edges, mask, EDGES_THRESHOLD, 255, THRESH_BINARY_INV);
```

Step 2: Generating a color painting and then overlay the edge mask "sketch" onto the bilateral filter "painting" to generate cartoon version of the original photo.

Details:

After obtaining a black-and-white sketch, then in order to obtain a color painting, we will use an edge preserving filter (bilateral filter) to further smooth the flat regions while keeping the edges intact. But bilateral filter is extremely slow when filtering the image, so we will resize the image----reduce the total number of pixels by a factor of four (for example, half width and half height) before applying the filter directly.

```
smallSize.width = size.width/2;
smallSize.height = size.height/2;
resize(srcColor, smallImg, smallSize, 0,0, INTER_LINEAR);
```

when applying many small bilateral filters to produce a strong color effect, we can apply one filter storing a temp Mat and another filter storing back to the input:

```
bilateralFilter(smallImg, tmp, ksize, sigmaColor, sigmaSpace);
bilateralFilter(tmp, smallImg, ksize, sigmaColor, sigmaSpace);
```

Remember that this was applied to the shrunken image, so we need to expand the image back to the original size.

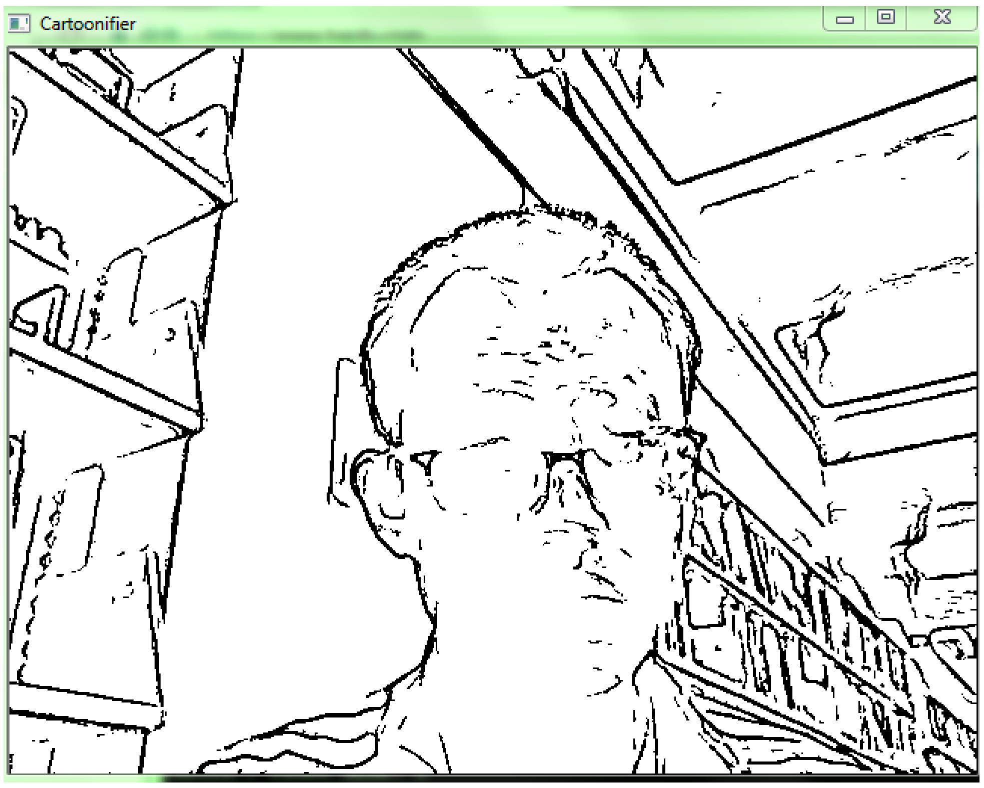
```
resize(smallImg, bigImg, size, 0,0, INTER_LINEAR);
```

To overlay the edge mask "sketch" onto the bilateral filter "painting", we can start with a black background and copy the "painting" pixels that aren't edges in the "sketch" mask, then we obtain a cartoon effect.

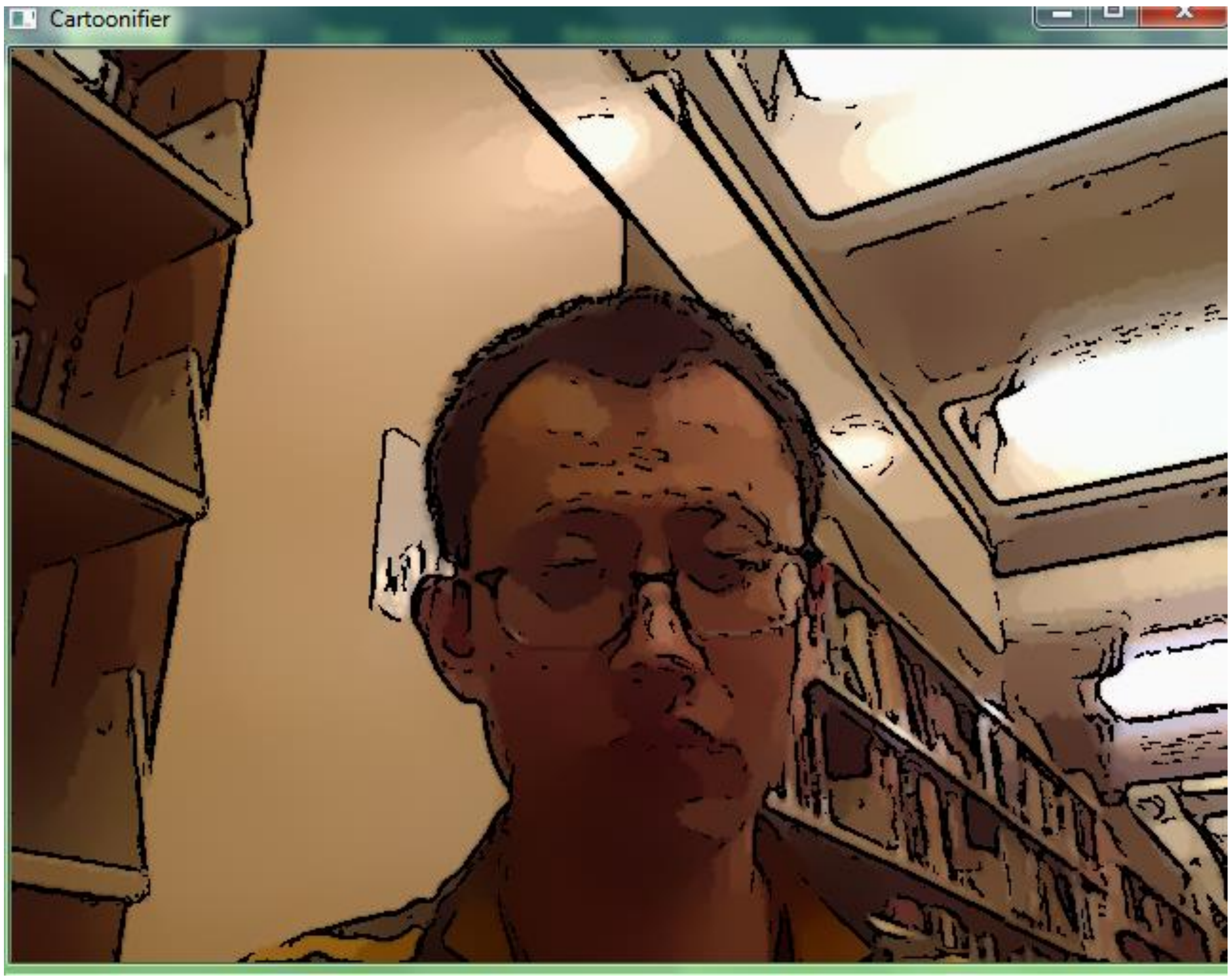
```
bigImg.copyTo(dst, mask);
```

b)

Step 1: Generating a black-and-white sketch :



Step 2: Generating a color painting and then overlay the edge mask "sketch" onto the bilateral filter "painting" to generate cartoon version of the original photo.



c) In Total, there are two major steps:

Step1：producing the “evil” mask：

Details:

In order to perform this on a grayscale image with some noise reduction, we need convert the original image to grayscale and applying a 7 x 7 Median filter to get the grayscale Median blur image first.

```
cvtColor(srcColor, gray, CV_BGR2GRAY);
medianBlur(gray, gray, MEDIAN_BLUR_FILTER_SIZE);
```

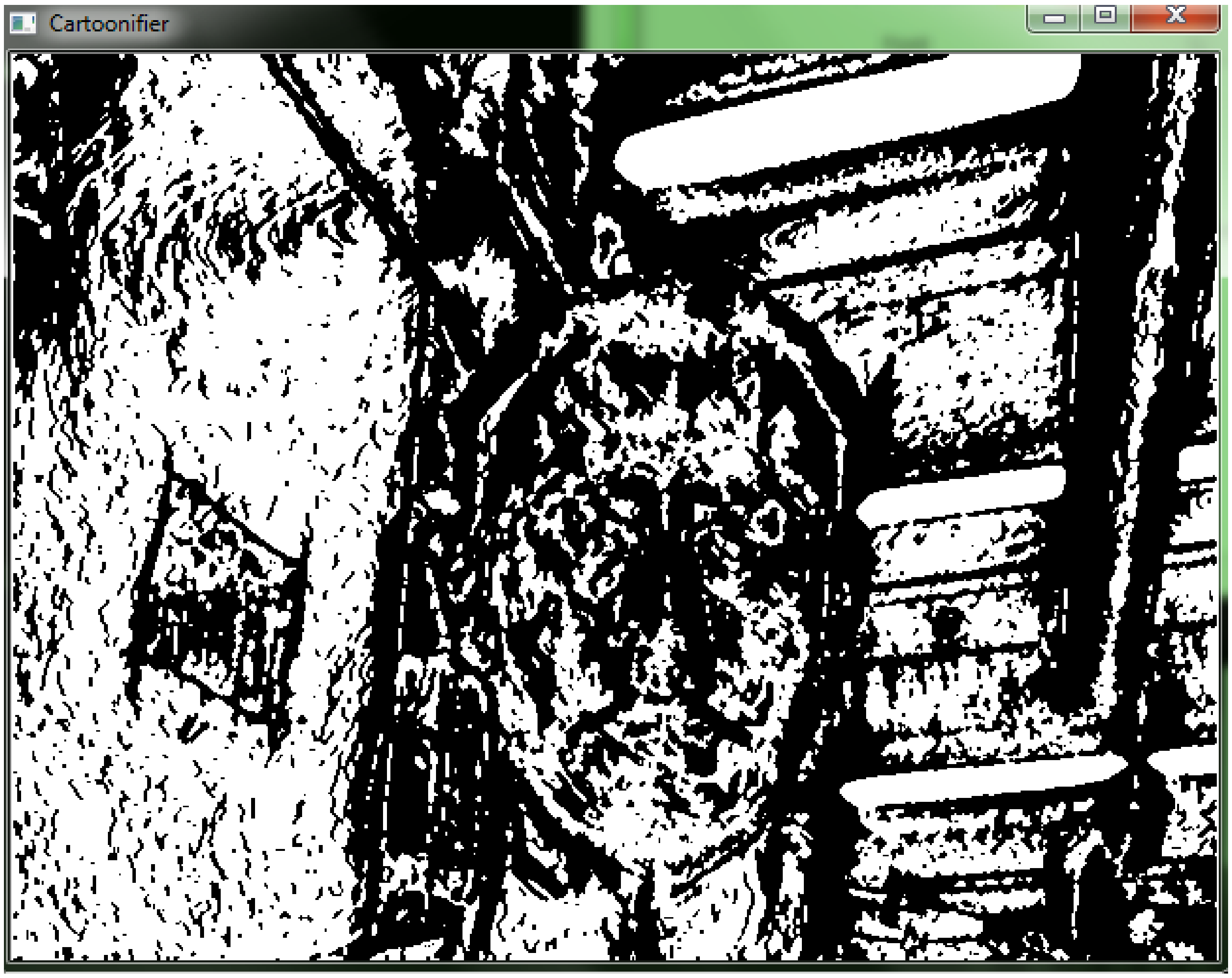
Following it, we apply a 3 x 3 Scharr gradient filter along x and y and then apply a binary threshold with a very low cutoff and a 3 x 3 Median blur to produce the final "evil" mask

```
Scharr(srcGray, edges, CV_8U, 1, 0);
Scharr(srcGray, edges2, CV_8U, 1, 0, -1);

threshold(edges, mask, EVIL_EDGE_THRESHOLD, 255, THRESH_BINARY_INV);

medianBlur(mask, mask, 3);
```

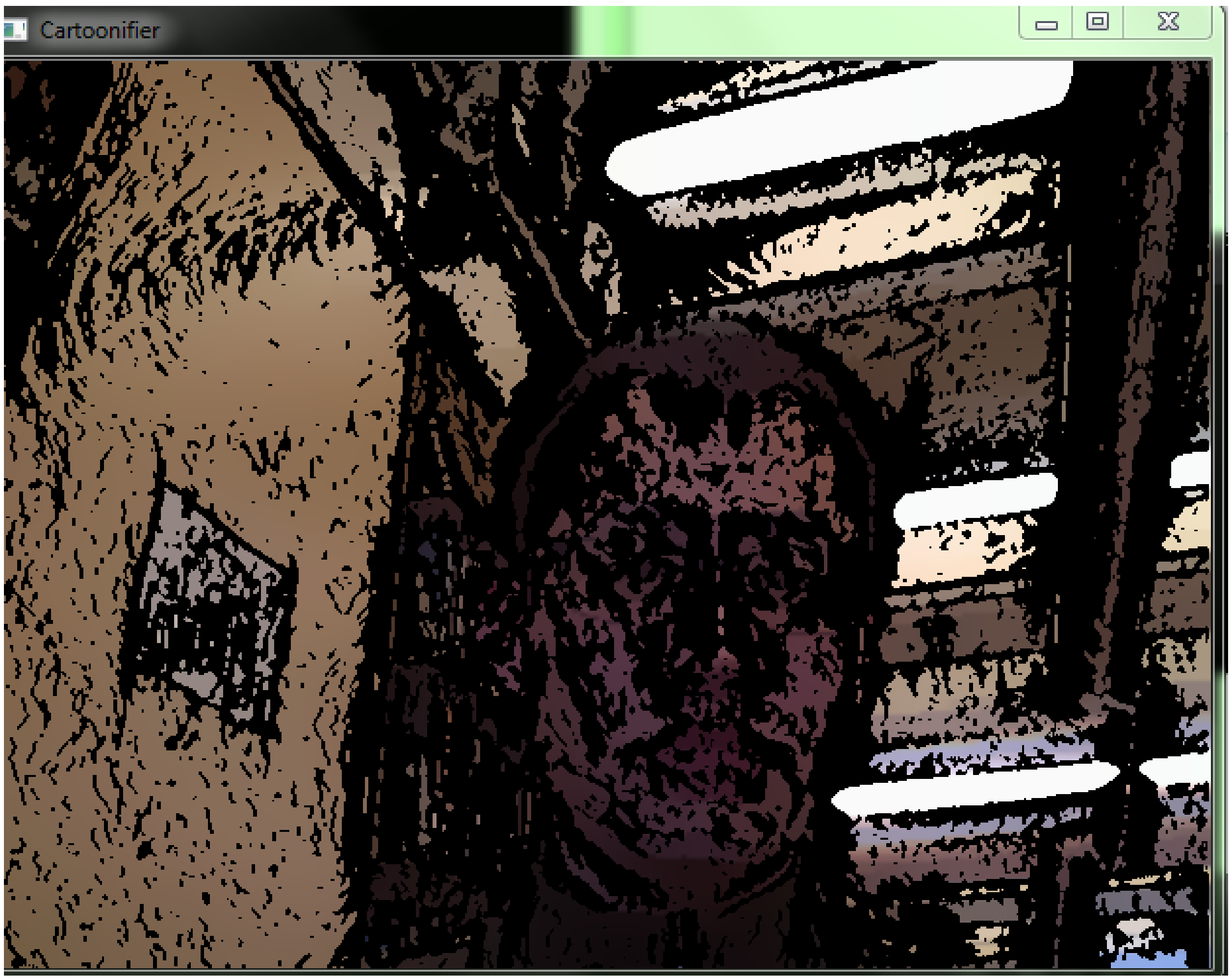
Step2：overlaying the “evil” mask onto the cartoonified “painting” image directly just like we did with the regular "sketch" edge mask

d) In Total, there are two major steps:

Step1：producing the “evil” mask：



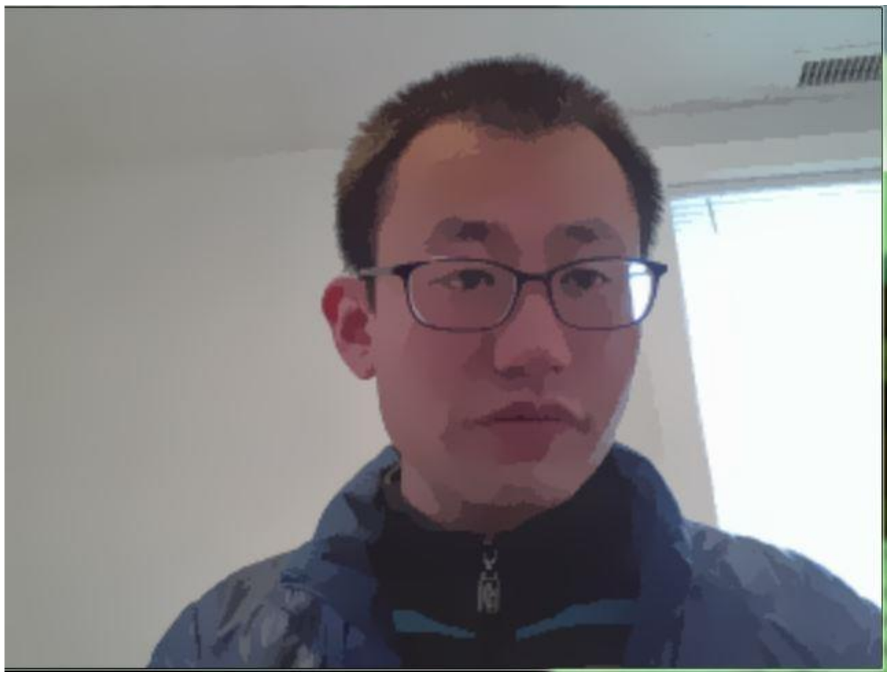
Step2：overlaying the “evil” mask onto the cartoonified “painting” image



e) We use a MedianBlur filter because it is good at removing noise while keeping edges sharp; also, it is not as slow as a bilateral filter.

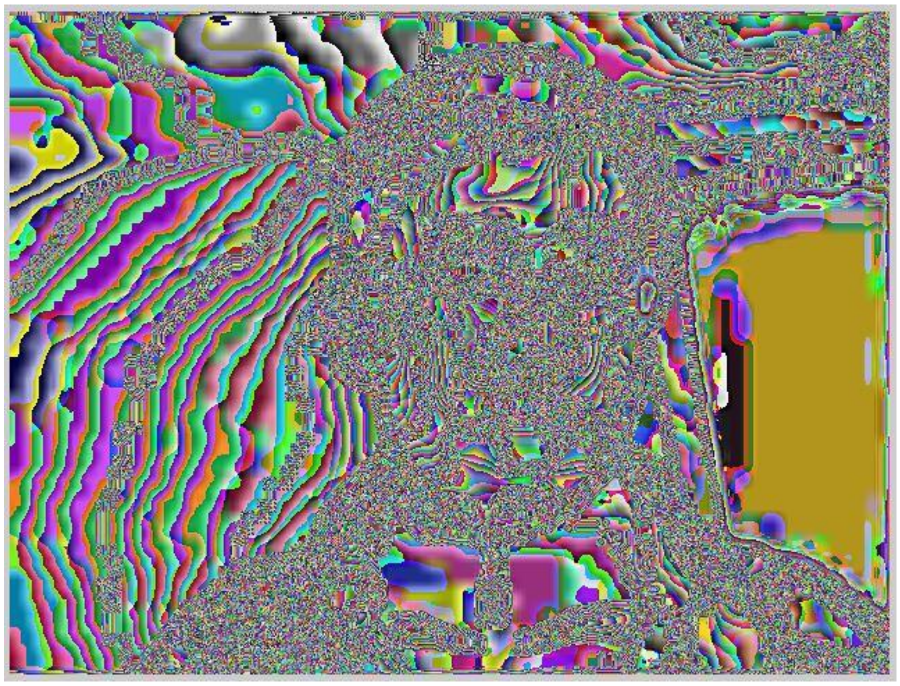
2. a. Oil painting Effect

Step 1: Use almost the same filter and process like what I did for the evil mode, except that I did not apply the function **threshold(edges, mask, EVIL_EDGE_THRESHOLD, 255, THRESH_BINARY_INV);**
a binary threshold with a very low cutoff to the original image.



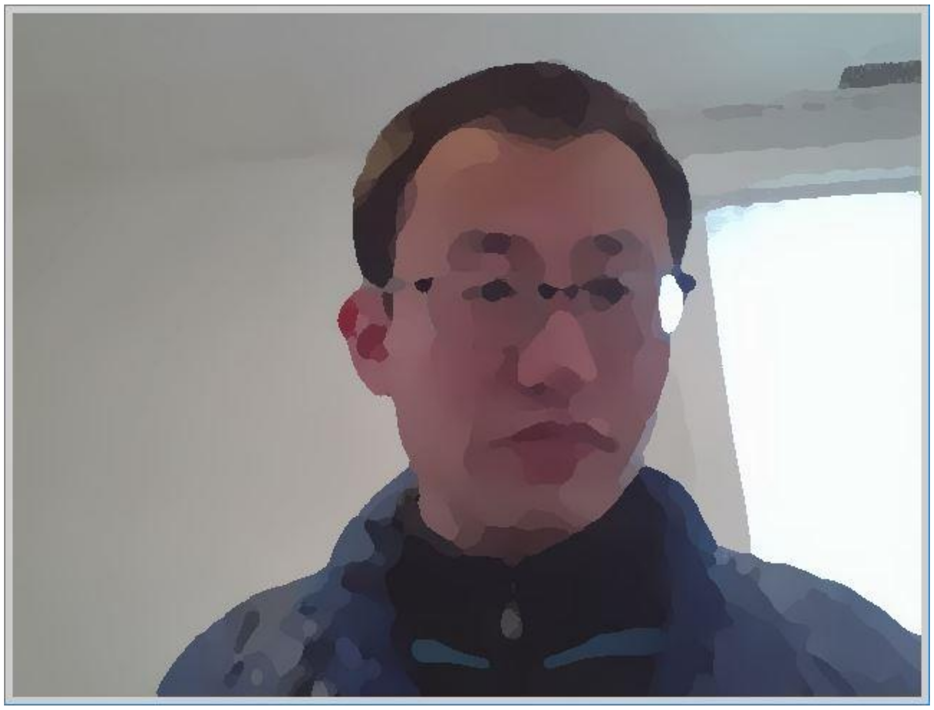
Step 2: For each pixel in the whole image, average the Red, Green, Blue channel value.

```
for (each pixel)
{
        for (each pixel, within radius r of pixel)
        {
                int curIntensity = (int)((double)((r+g+b)/3)*intensityLevels)/255.0f;
                intensityCount[curIntensity]++;
                averageR[curIntensity] += r;
                averageG[curIntensity] += g;
                averageB[curIntensity] += b;
        }
}
```



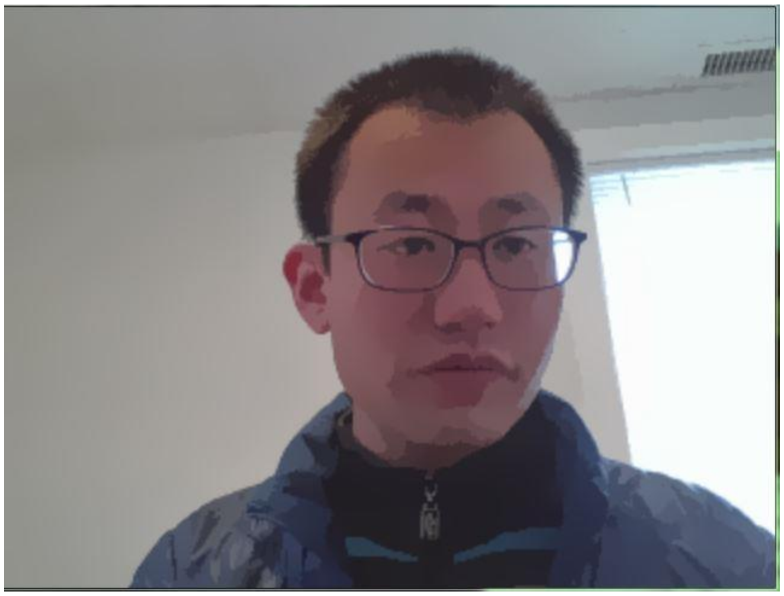
Step 3: For each pixel, determine which intensity bin has the most number of pixels in it, find the maximum level of intensity. After we determine which intensity the pixel represents, as determined by the intensity bin with the most pixels, we can then determine the final color of the pixel by taking the total red, green, and blue values in that specific bin, and dividing that by the total number of pixels in that specific intensity bin.

```
for (each level of intensity)
{
        if (intensityCount[i] > curMax)
        {
                curMax = intensityCount[i];
                maxIndex = i;
        }
}

finalR = averageR[maxIndex] / curMax;
finalG = averageG[maxIndex] / curMax;
finalB = averageB[maxIndex] / curMax
```



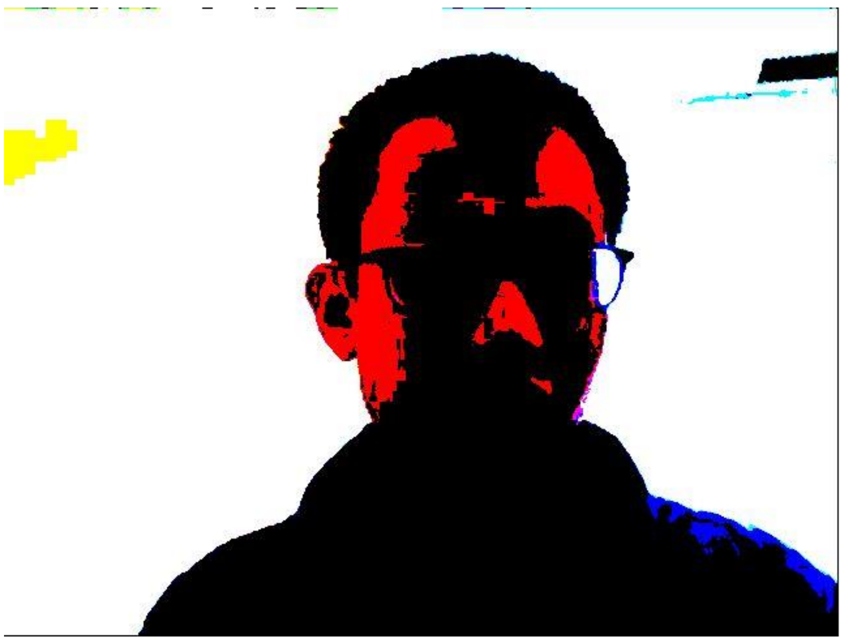
b. Floyd–Steinberg dithering effect

Step 1: Use almost the same filter and process like what I did for the evil mode, except that I did not apply the function **threshold(edges, mask, EVIL_EDGE_THRESHOLD, 255, THRESH_BINARY_INV);** a binary threshold with a very low cutoff to the original image.



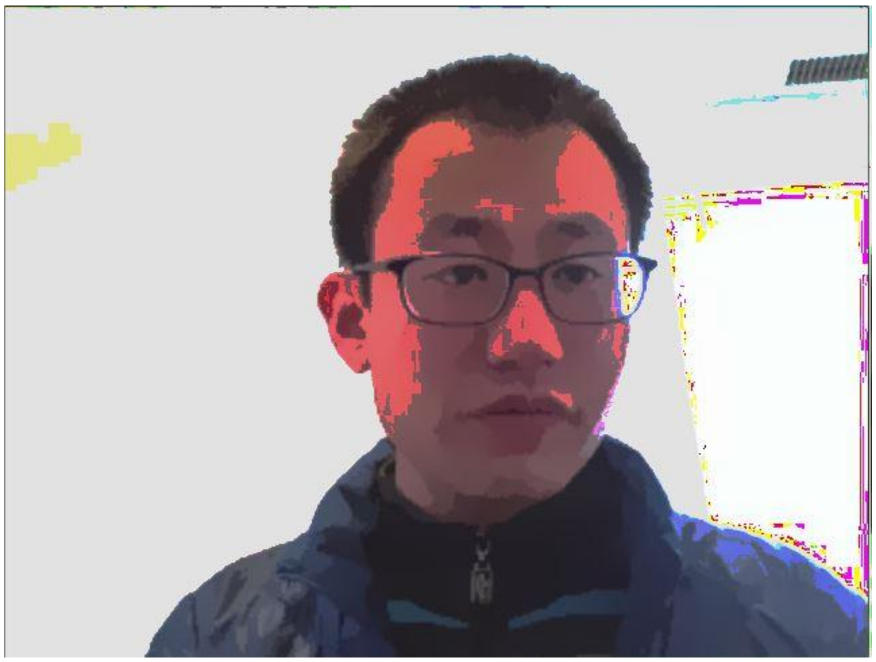
Step 2:
Base on each pixel's Red, Green, Blue channel's value, use the mathematics calculate the new RGB value. If the RGB channel value is less than 128, then set them to 0; If the RGB channel value is not less than 128, then set them to 255.

```
double ob = src.at<Vec3b>(y,x)[0];
double og = src.at<Vec3b>(y,x)[1];
double or = src.at<Vec3b>(y,x)[2];

double nb = ob < 128 ? 0 : 255;
double ng = og < 128 ? 0 : 255;
double nr = or < 128 ? 0 : 255;

src.at<Vec3b>(y,x)[0] = nb;
src.at<Vec3b>(y,x)[1] = ng;
src.at<Vec3b>(y,x)[2] = nr;
```



Step 3: Calculate the per pixel Error RGB channel value between the Step1 and Step 2 the algorithms below:

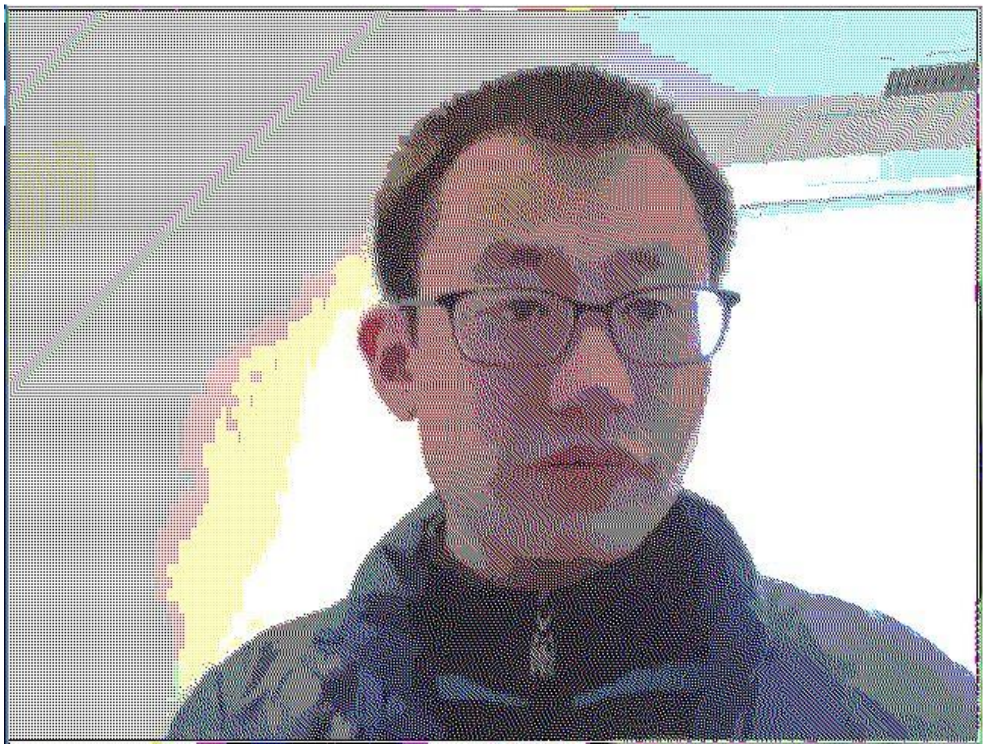
```
 errorB = ob - nb;
 errorG = og - ng;
 errorR = or - nr;

int tmp = -30;
if(errorB < tmp) errorB = tmp;
if(errorG < tmp) errorG = tmp;
if(errorR < tmp)  errorR = tmp;
```



Step 4: using error diffusion, pushes (adds) the residual quantization error of a pixel onto its neighboring pixels and get final image

```
if(x < (width - 1))
{
        src.at<Vec3b>(y, x+1)[0] += 7*errorB/16;
        src.at<Vec3b>(y, x+1)[1] += 7*errorG/16;
        src.at<Vec3b>(y, x+1)[2] += 7*errorR/16;
}
if(x > 1 && y < (heigh - 1))
{
        src.at<Vec3b>(y+1, x-1)[0] += 3*errorB/16;
        src.at<Vec3b>(y+1, x-1)[1] += 3*errorG/16;
        src.at<Vec3b>(y+1, x-1)[2] += 3*errorR/16;
}
if(y < (heigh - 1))
{
        src.at<Vec3b>(y+1, x)[0] += 5*errorB/16;
        src.at<Vec3b>(y+1, x)[1] += 5*errorG/16;
        src.at<Vec3b>(y+1, x)[2] += 5*errorR/16;
}
if(x < (width - 1) && y < (heigh - 1))
{
        src.at<Vec3b>(y+1, x+1)[0] += errorB/16;
```

```
                    src.at<Vec3b>(y+1, x+1)[1] += errorG/16;
                    src.at<Vec3b>(y+1, x+1)[2] += errorR/16;
                }
```



3.

(a) No, both the correlation and SSD are sensitive to the intensity or the illumination offsets, not robust to changes in brightness. When a scene is imaged by different sensors, or under different illumination intensities, both the SSD and the C_{fg} can be large for windows representing the same area in the scene!

Consider correlation of template with an image of constant grey value:

$$\begin{array}{|c|c|c|}\hline a & b & c \\ \hline d & e & f \\ \hline g & h & i \\ \hline \end{array} \otimes \begin{array}{|c|c|c|}\hline v & v & v \\ \hline v & v & v \\ \hline v & v & v \\ \hline \end{array}$$

Result: v*(a+b+c+d+e+f+g+h+i)

Now consider correlation with a constant image that is twice as bright.

$$\begin{array}{|c|c|c|}\hline a & b & c \\ \hline d & e & f \\ \hline g & h & i \\ \hline \end{array} \otimes \begin{array}{|c|c|c|}\hline 2v & 2v & 2v \\ \hline 2v & 2v & 2v \\ \hline 2v & 2v & 2v \\ \hline \end{array}$$

Result: 2*v*(a+b+c+d+e+f+g+h+i)
> v*(a+b+c+d+e+f+g+h+i)

Larger score, regardless of what the template is!

(b). Yes, we will get the same answer with two methods. Rewritten the equation $C(d) = w \cdot w'$ $S(d) = |w - w'|^2$ into the zero mean and normalized form following. In the following equation, parameter u = d, parameter $\boldsymbol{I_0} = \boldsymbol{w}$, parameter $\boldsymbol{I_1} = \boldsymbol{w'}$

$$E_{\mathrm{NCC}}(\boldsymbol{u}) = \frac{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]\,[I_1(\boldsymbol{x}_i+\boldsymbol{u}) - \overline{I_1}]}{\sqrt{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]^2}\sqrt{\sum_i [I_1(\boldsymbol{x}_i+\boldsymbol{u}) - \overline{I_1}]^2}},$$

where

$$\overline{I_0} = \frac{1}{N}\sum_i I_0(\boldsymbol{x}_i) \quad \text{and}$$
$$\overline{I_1} = \frac{1}{N}\sum_i I_1(\boldsymbol{x}_i+\boldsymbol{u})$$

And it also turns out that E_{NSSD} is only the variant of the E_{NCC}, which is related to the bias–gain regression implicit in the matching score

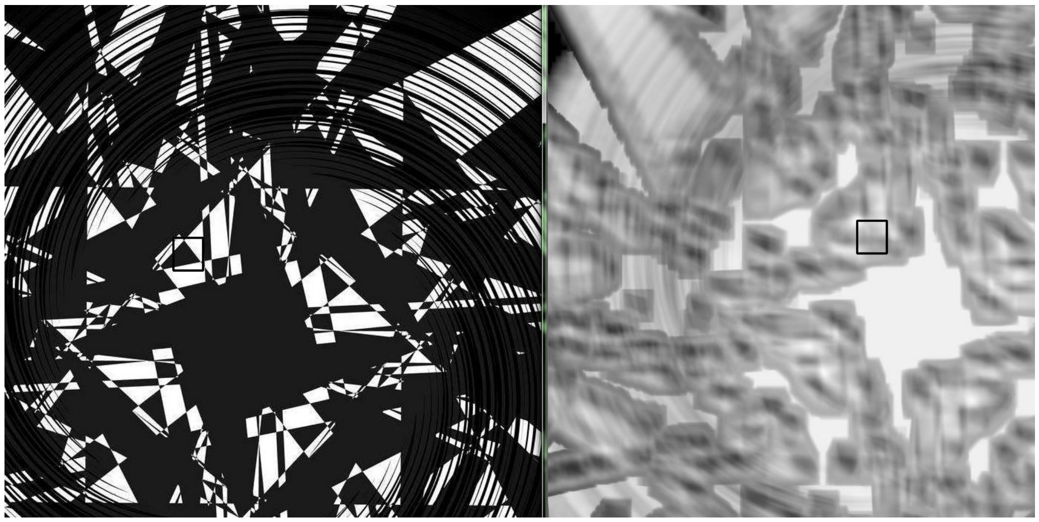
$$E_{\mathrm{BG}}(\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x}_i+\boldsymbol{u}) - (1+\alpha)I_0(\boldsymbol{x}_i) - \beta]^2 = \sum_i [\alpha I_0(\boldsymbol{x}_i) + \beta - e_i]^2.$$

So that, we could combine the E_{NCC} equation and the E_{BG} equation together to derive the E_{NSSD}:

$$E_{\mathrm{NSSD}}(\boldsymbol{u}) = \frac{1}{2}\frac{\sum_i \left[[I_0(\boldsymbol{x}_i) - \overline{I_0}] - [I_1(\boldsymbol{x}_i+\boldsymbol{u}) - \overline{I_1}]\right]^2}{\sqrt{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]^2 + [I_1(\boldsymbol{x}_i+\boldsymbol{u}) - \overline{I_1}]^2}}$$

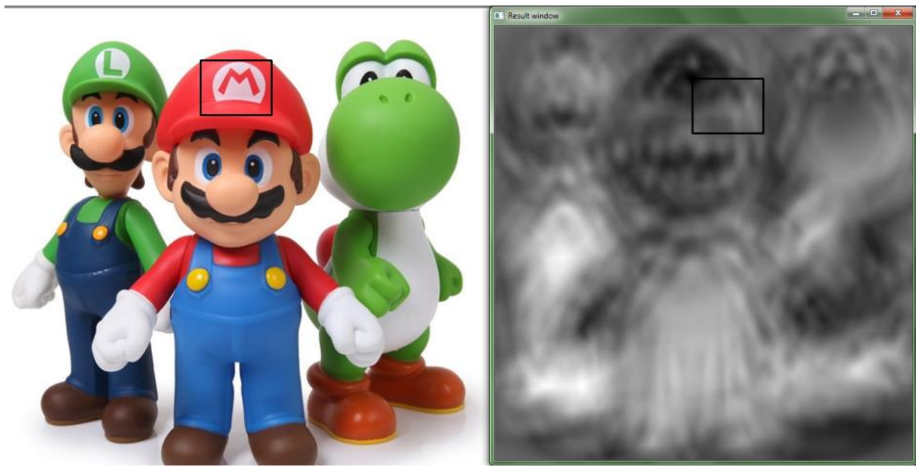
Beyond that, recently proposed by Criminisi, Shotton, Blake et al. (2007). In their experiments, they find that E_{NSSD} produces comparable results to E_{NCC}.

(c). Template:

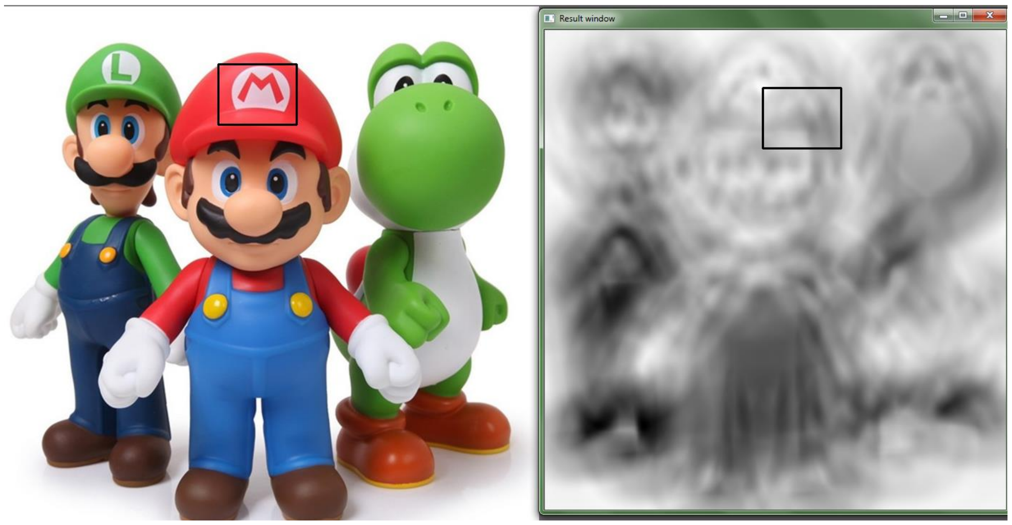


4. a. template

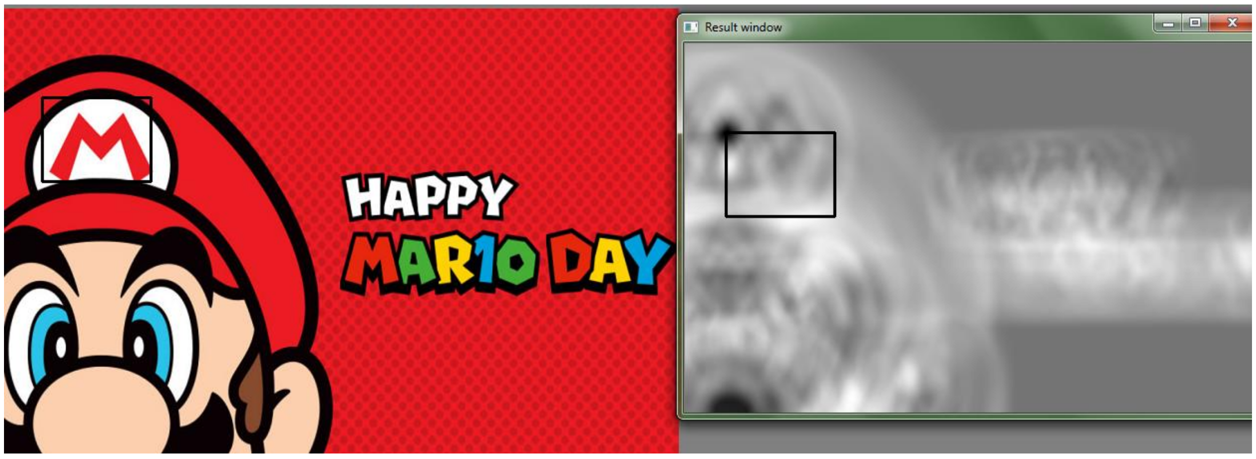
SSD:



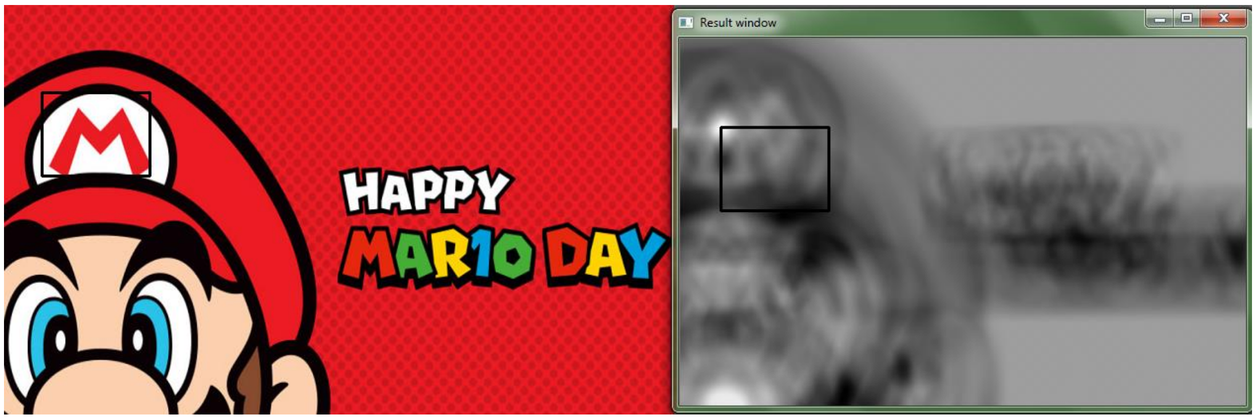
Normalized Correlation:



SSD:



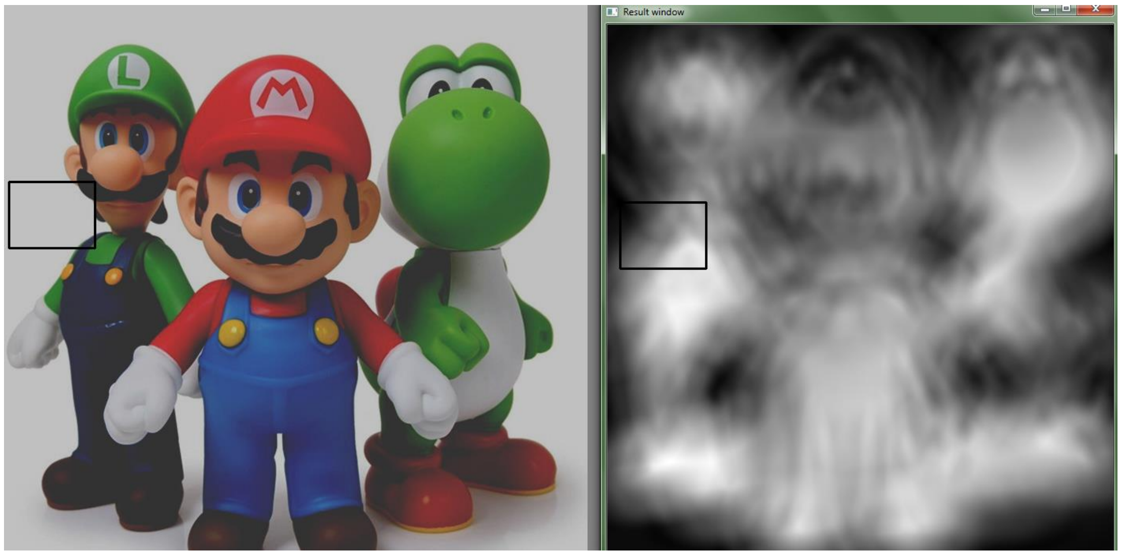
Normalized Correlation:



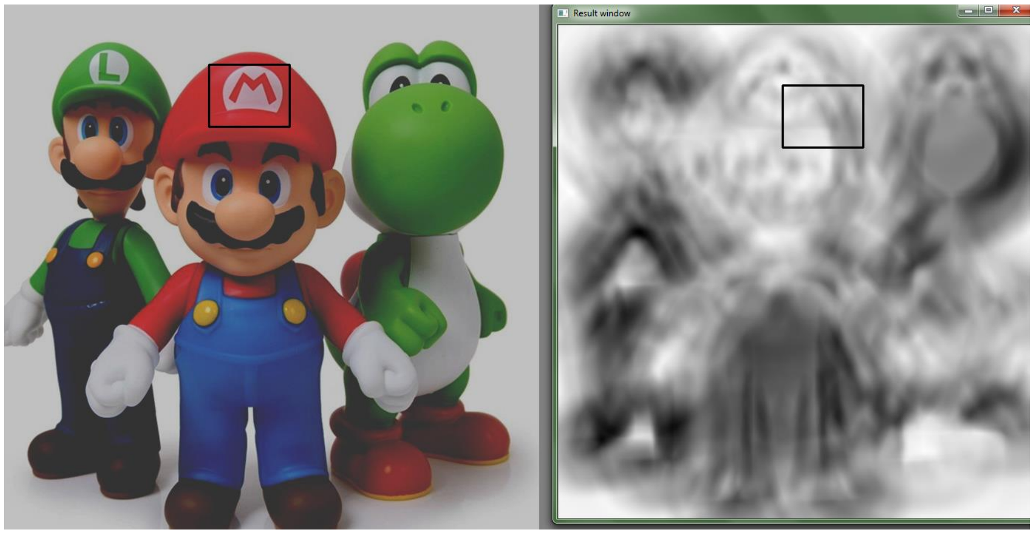
b.
Result: In terms of change of overall intensity and contrast, Normalized Correlation is more robust than SSD. In terms of rotation, both Normalized Correlation and SSD are robust.
In terms of scale, SSD is more robust than Normalized Correlation.

Decrease overall Intensity:

SSD: (failed match)

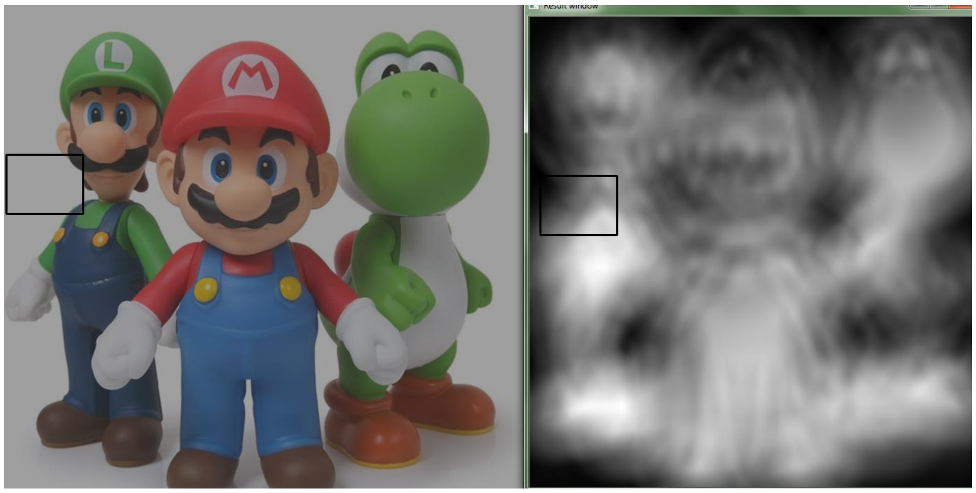


Normalized Correlation:(Successful match)

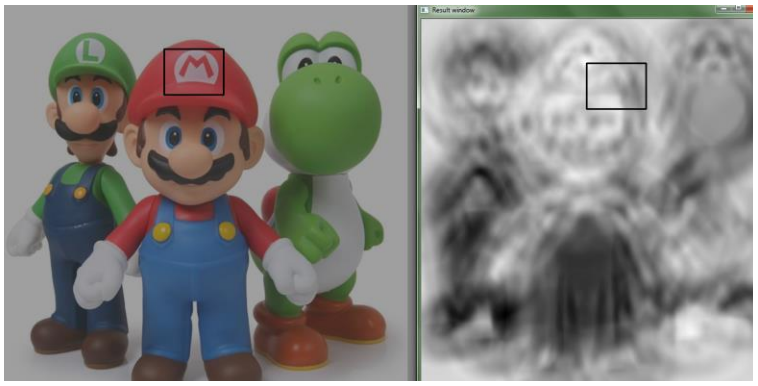


Contrast:

SSD: (failed match)

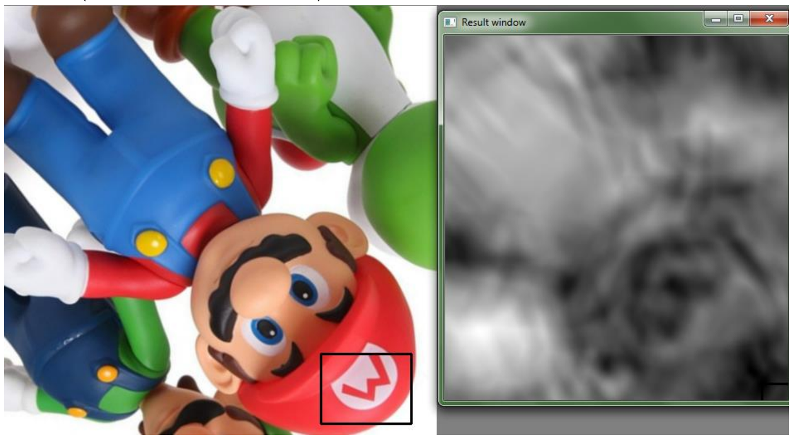


Normalized Correlation:(Successful match)

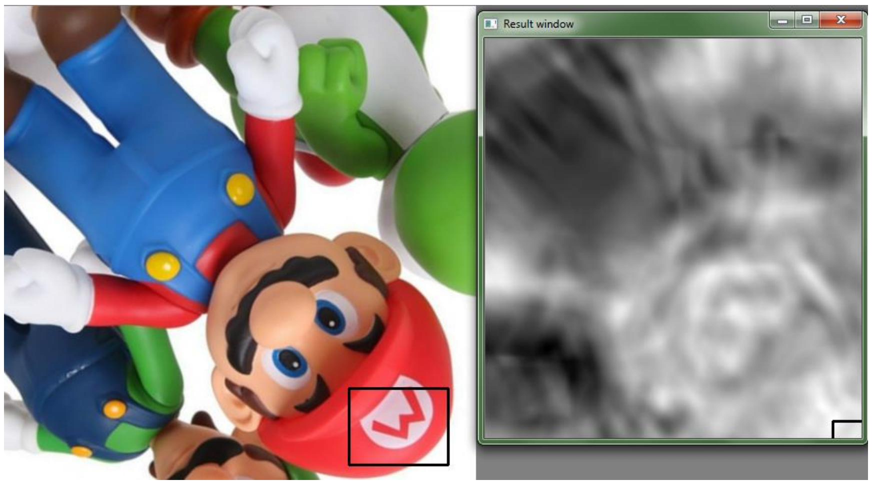


Rotate:

SSD: (Successful match)

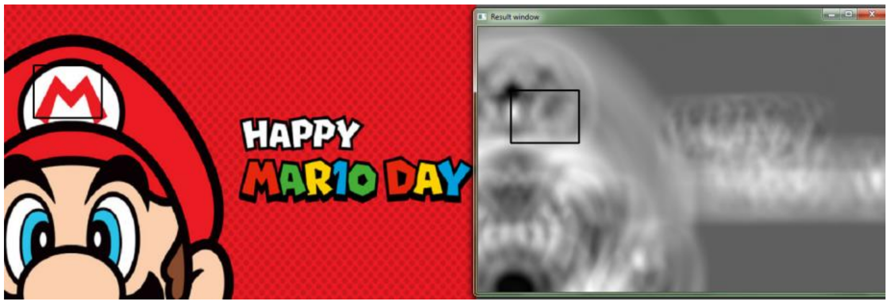


Normalized Correlation:(Successful match)

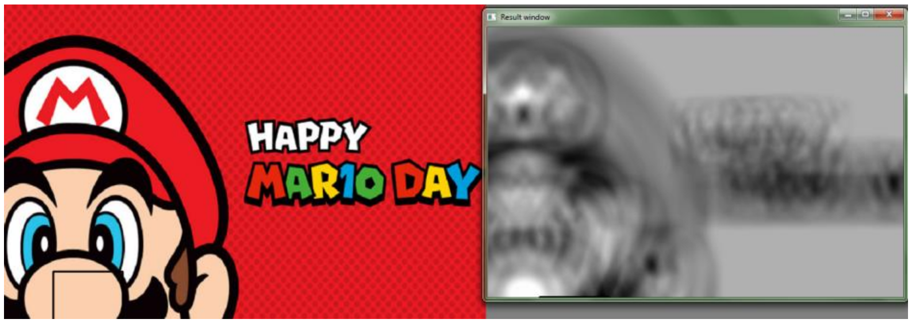


Scale x1.12

SSD: (Successful match)



Normalized Correlation:(Failed match)


HAPPY
MAR10 DAY
Result window