DIP Final Project

——The recurrence of Color Balance and Underwater Image Enhance

Members：

1652690	苏昭帆
1652791	温庭杰
1551534	孙允鑫

Ⅰ. Introduction

Different from common images, underwater images suffer from poor visibility resulting from the attenuation of the propagated light, mainly due to absorption and scattering effects. The absorption substantially reduces the light energy, while the scattering causes changes in the light propagation direction. They result in foggy appearance and contrast degradation, making distant objects misty. Practically, in common sea water images, the objects at a distance of more than 10 meters are almost unperceivable, and the colors are faded because their composing wavelengths are cut according to the water depth.

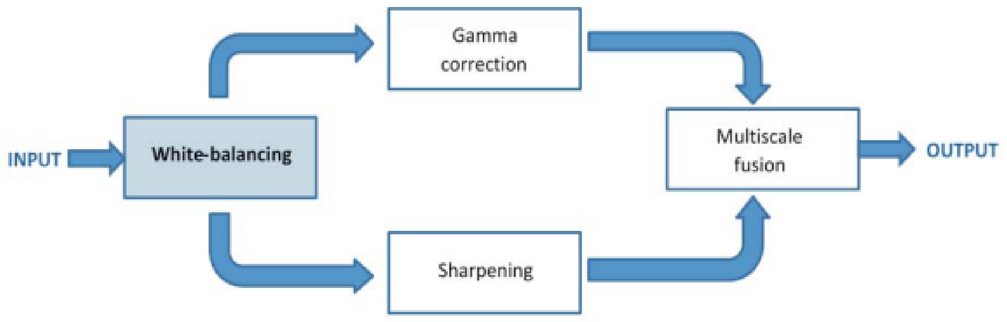
Traditional enhancing techniques such as gamma correction, histogram equalization appear to be strongly limited for restoring and enhancing the visibility of such degraded images since the deterioration of underwater scenes results from the combination of multiplicative. The problem has been tackled by tailored acquisition strategies using multiple images, specialized hardware or polarization filters. Despite of their valuable achievements, these strategies suffer from a number of issues that reduce their practical applicability. So the authors of the paper named Color Balance and Underwater Image Enhance introduce a novel approach to remove the haze in underwater images based on a single image captured with a conventional camera. Their approach builds on the fusion of multiple inputs, but derives the two inputs to combine by correcting the contrast and by sharpening a white-balanced version of a single native input image. (The white balancing stage aims at removing the color cast induced by underwater light scattering, so as to produce a natural appearance of the sub-sea images. The multi-scale implementation of the fusion process results in an artifact-free blending.)

And our group have learned the approach introduced in the paper, and implemented the recurrence of the algorithm. So in this report, we will talk about the principles of the algorithm, our code and the problems we met in our learning process.

Ⅱ.The specific implementation

A: The overview of the method

At first we will use the following figure to explain how the authors design their algorithm.



According to the method, we need two images derived from a white-balanced version of the single input, and then merge the two images based on a standard multi-scale fusion algorithm.

After this, we will get the output, the resulting image processed by the authors' method.

The authors' image enhancement approach adopts a two step strategy, combining white balancing and image fusion, to improve underwater images without resorting to the explicit inversion of the optical model. In our approach, white balancing aims at compensating for the color cast caused by the selective absorption of colors with depth, while image fusion is considered to enhance the edges and details of the scene, to mitigate the loss of contrast resulting from back-scattering.

B: Red channel compensating and white-balancing using Gray-world

Before starting the process, we should know that sub-sea water absorbs gradually different wavelengths of light. Red, which corresponds to the longest wavelength, is the first to be absorbed (10-15 ft), followed by orange (20-25 ft), and yellow (35-45 ft). Pictures taken at 5 ft depth will have a noticeable loss of red. Furthermore, the refractive index of water makes judging distances difficult. As a result, underwater objects can appear 25% larger than they really are.

And there are several classes of underwater dehazing techniques. An important class corresponds to the methods using specialized hardware. But the complex acquisition systems are very expensive, and power consuming. A second class consists in polarization-based methods. While being effective in recovering distant regions, the polarization techniques are not applicable to video acquisition, and are therefore of limited help when dealing with dynamic scenes. A third class of approaches employs multiple images or a rough approximation of the scene model. However these methods are impractical for common users.

Now, let's focus on the white-balancing stage. White-balancing aims at improving the image aspect, primarily by removing the undesired color castings due to various illumination or medium attenuation properties. After several comparing some white balancing method, the authors finally decide to use the famous Gray-World algorithm which achieves good visual performance for reasonably distorted underwater scenes. However, it also performances bad with extremely deteriorated underwater scenes. Though the Gray-World can best remove the bluish tone, it also suffers from severe red artifacts due to a very small mean value for the red channel. So before the white-balancing, we need to compensate for the loss of the red channel. The compensated red channel I_{rc} at every pixel location *x* as follows:

$$I_{rc}(x) = I_r(x) + \alpha.(\bar{I}_g - \bar{I}_r).(1 - I_r(x)).I_g(x),$$

where I_r, I_g represent the red and green color channels of image I, each channel being in the interval [0, 1], after normalization by the upper limit of their dynamic range; while $\bar{I}_r$ and $\bar{I}_g$ denote the mean value of I_r and I_g. Simply, we make the $\alpha = 1$.

When blue is strongly attenuated and the compensation of the red channel appears to be insufficient, we should also compensate for the blue channel attenuation, and compute the compensated blue channel I_{bc} as:

$$I_{bc}(x) = I_b(x) + \alpha.(\bar{I}_g - \bar{I}_b).(1 - I_b(x)).I_g(x),$$

where I_b, I_g represent the blue and green color channels of image I, and α is also set to one.

And the code of the red channel compensating with normalization is as following:

```
function ret = aRedCompensate( im, w )
    r=im(:,:,1);
    g=im(:,:,2);
    b=im(:,:,3);
    r=im2double(r);
    g=im2double(g);
    b=im2double(b);
    [height,width,~]=size(im);
    padsize=[(w-1)/2,(w-1)/2];
    padr = padarray(r,padsize,'symmetric','both');
    padg = padarray(g,padsize,'symmetric','both');
    ret=im;
    for i=1:height
        for j=1:width
            slider = padr(i:i+w-1,j:j+w-1);
            slideg = padg(i:i+w-1,j:j+w-1);
            rm=mean(mean(slider));
            gm=mean(mean(slideg));
            tem=r(i,j)+1.2*(gm-rm)*(1-r(i,j))*g(i,j);
            tem=tem*255;
            tem=uint8(tem);
            ret(i, j, 1)=tem;
        end
    end
end
```

The code of the blue channel compensating:

```
function [ ret ] = blueCompensate( im )
    im=im2double(im);
    [M, N, ~]=size(im);
    r=im(:,:,1);
    g=im(:,:,2);
    b=im(:,:,3);
    meanB=mean(mean(b));
    meanG=mean(mean(g));
    for i=1:M
        for j=1:N
            b(i, j)=b(i, j)+(meanG-meanB)*(1-b(i, j))*g(i, j);
        end
    end
    ret(:,:,1)=r;
    ret(:,:,2)=g;
    ret(:,:,3)=b;
```

```
end
```

As noted in our code, we use an adaptive method to implement red compensate for the reason that the means of r channel and g channel are quite different in different local areas.

After the red (and optionally the blue) channel attenuation has been compensated, we now use the conventional Gray-World assumption to estimate and compensate the illuminant color cast.

The white-balancing method, Gray-World, which devides each channel by its mean value was advanced by G. Buchsbaum in 1980. And the following is the code of this algorithm:

```
function ret = grayWorld( im )
    r=im(:,:,1);
    g=im(:,:,2);
    b=im(:,:,3);

    avgR = mean(mean(r));
    avgG = mean(mean(g));
    avgB = mean(mean(b));

    avgRGB = [avgR avgG avgB];
    grayValue = (avgR + avgG + avgB)/3;
    scaleValue = grayValue./avgRGB;

    ret(:,:,1) = scaleValue(1) * r;
    ret(:,:,2) = scaleValue(2) * g;
    ret(:,:,3) = scaleValue(3) * b;
end
```

Now we get the image processed by white-balancing. However it is not sufficient to solve the dehazing problem since the edges and details of the scene have been affected by the scattering. So we next use an effective fusion based approach, relying on gamma correction and sharpening to deal with the hazy nature of the white balanced image according to the authors.

C: Gamma correction

In water deeper than 30 ft, white balancing suffers from noticeable effects since the absorbed colors are difficult to be recovered. Gamma correction aims at correcting the global contrast and is relevant since, in general, white balanced underwater images tend to appear too bright.

For the RGB image, we use the following code to complete the gamma correction:

```
function [ res ] = gammaCorrection( im,a,gamma )
    im = im2double(im);
    r=im(:,:,1);
    g=im(:,:,2);
    b=im(:,:,3);

    res_r = a *(r .^ gamma);
```

```
        res_g = a *(g .^ gamma);
        res_b = a *(b .^ gamma);

        res(:,:,1) = res_r;
        res(:,:,2) = res_g;
        res(:,:,3) = res_b;
    end
```

After the gamma correction, we can get the first input image of the next fusion process.

D: Sharpening

However, Gamma Correction increases the difference between darker and lighter regions at the cost of a loss of details in the under-/over-exposed regions. To compensate for this loss, we must extract details of the certain image by sharpening. As noted in $S = I + \beta(I - G * I)$,, where $G*I$ denotes the Gaussian filtered version of I, S is the sharp image and β is an constant, we can get details of a image by using an unsharp mask. However, it is hard to generate an appropriate β. Hence, we user another method of sharpening. We define the sharpened image S as follows

$$S = (I + N\{I - G*I\})/2,$$

where N is the histogram stretching operation. Using this equation, we can get our sharpen image without estimating an appropriate β. The following is the code of sharping.

```
    function [ ret ] = sharp( im )
        GaussKernel=fspecial('gaussian',5,1);
        im=double(im);
        im=im./255;
        imBlur=imfilter(im,GaussKernel);
        unSharpMask=im-imBlur;
        stretchIm=im2double(unSharpMask);
        ret=(im+stretchIm)/2;
    end
```

E: Laplacian Weight

It estimates the global contrast by computing the absolute value of a Laplacian filter applied on each input luminance channel. This straightforward indicator was used in different applications. However, it is not sufficient to recover the contrast, mainly because it cannot distinguish much between a ramp and flat regions.

```
function lapweight = lapweight(img)
grayimg = rgb2gray(img);
lapweight = double(abs(imfilter(grayimg, fspecial("laplacian"))));
end
```

F: Saliency Detection

This aims at emphasizing the salient object that lose their prominence in the underwater scene. This concept has been inspired by the biological concept of center-surround contrast. However, the saliency map tends to highlighted areas.

```
function sm = saliency_detection(img)
gfrgb = imfilter(img, fspecial('gaussian', 3, 3), 'symmetric', 'conv');
cform = makecform('srgb2lab', "AdaptedWhitePoint", whitepoint("d65"));
```

```
lab = applycform(gfrgb,cform);
l = double(lab(:,:,1)); lm = mean(mean(l));
a = double(lab(:,:,2)); am = mean(mean(a));
b = double(lab(:,:,3)); bm = mean(mean(b));
sm = (l-lm).^2 + (a-am).^2 + (b-bm).^2;
end
```

G: Saturation Weight

It enables the fusion algorithm to adapt to chromatic information by advantaging highly saturated regions. This weight map is simply computed as the deviation between each color channels and the luminance.

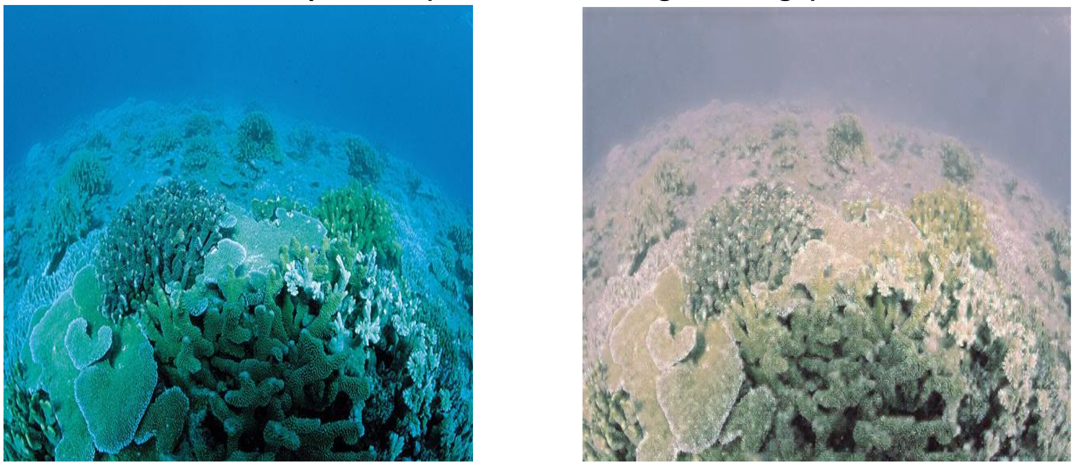
```
function wsat = wsat(img)
r = double(img(:, :, 1));
g = double(img(:, :, 2));
b = double(img(:, :, 3));
l = r * .3 + g * .59 + b * .11;
wsat = (r - l).^2 + (g - l).^2 + (b - l).^2;
wsat = sqrt(wsat ./ 3);
end
```

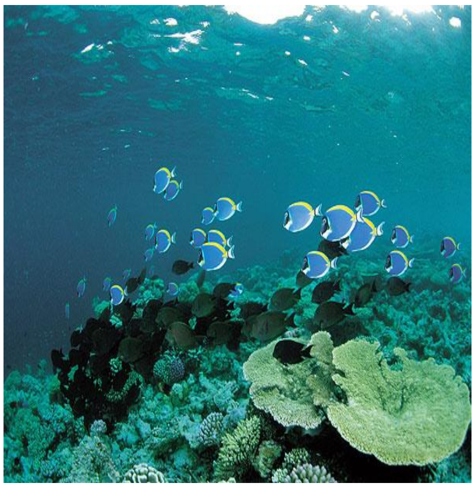
H: Multi-Scale Fusion

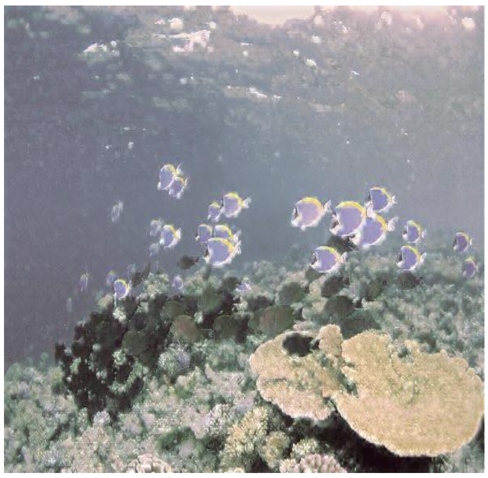
The multi-scale decomposition is based on Laplacian pyramid. The pyramid representation decomposes an image into a sum of bandpass images. By independently employing a fusion process at every scale level, the potential artifacts due to the sharp transitions of the weight maps are minimized. Multi-scale fusion is motivated by the human visual system, which is very sensitive to sharp transitions appearing in smooth image patterns, while being less sensitive to variations occurring on edges and textures.

```
function out = gaussian_pyramid(img, level)
h = 1/16* [1, 4, 6, 4, 1];
filt = h'*h;
out = 1:1:level;
out{1} = imfilter(img, filt, 'replicate', 'conv');
temp_img = img;
for i = 2 : level
    temp_img = temp_img(1 : 2 : end, 1 : 2 : end);
    out{i} = imfilter(temp_img, filt, 'replicate', 'conv');
end

function out = laplacian_pyramid(img, level)
h = 1/16* [1, 4, 6, 4, 1];
%filt = h'*h;
out{1} = img;
temp_img = img;
for i = 2 : level
    temp_img = temp_img(1 : 2 : end, 1 : 2 : end);
    %out{i} = imfilter(temp_img, filt, 'replicate', 'conv');
    out{i} = temp_img;
end
% calculate the DoG
for i = 1 : level - 1
    [m, n] = size(out{i});
    out{i} = out{i} - imresize(out{i+1}, [m, n]);
end

function out = pyramid_reconstruct(pyramid)
level = length(pyramid);
for i = level : -1 : 2
    %temp_pyramid = pyramid{i};
    [m, n] = size(pyramid{i - 1});
    %out = pyramid{i - 1} + imresize(temp_pyramid, [m, n]);
    pyramid{i - 1} = pyramid{i - 1} + imresize(pyramid{i}, [m, n]);
end
out = pyramid{1};
```

Ⅲ. The result of the experience(The left is the original image)







Ⅳ. The problems we met and the solutions

a、At first the image processed by the Gray-World wasn't the way it should be.

First, we forgot to normalize each channel of the image. After normalizing, we found that the effect was still not very good. We thought that the reason might be the image we chose. it might be after some processing, not the real underwater image. So we selected other images, tried for several times and get the result appropriate visually.