**NA-HW #08**

2015004693\_양상헌

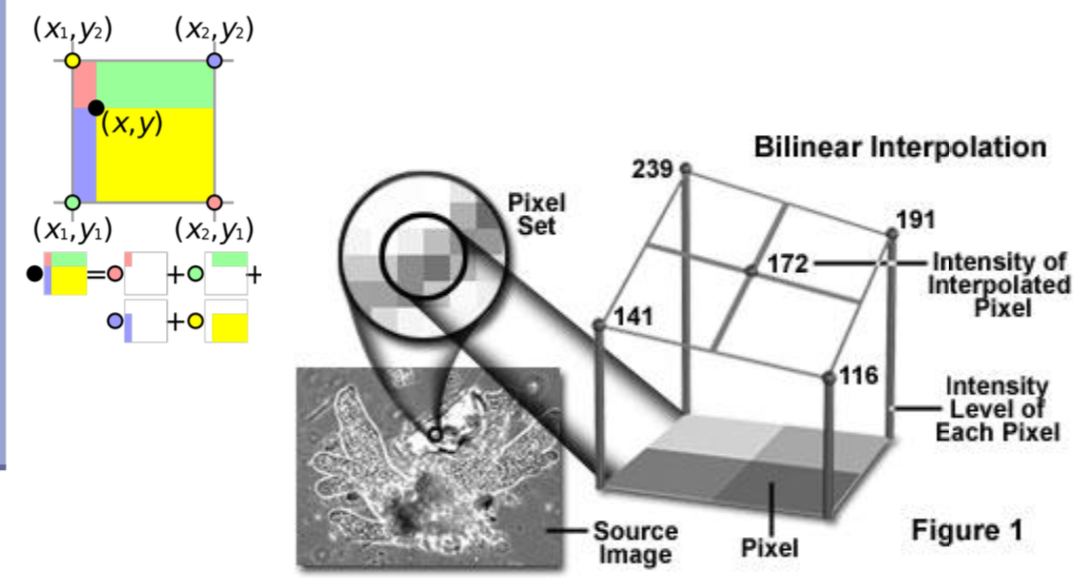
**Goal:**

* **Resizing image by using Bilinear Interpolation Method in OPENCV**

첨부된 Skeleton Code인 “forInterpolation.cpp” 파일을 편집하여 (“BilinearInterpolation()” 함수를 생성) Source Image를 읽어와 그 Image의 원래 크기와는 다른 임의의 원하는 사이즈로 Image를 편집하는(Resize) 프로그램을 만든다. 이때 Image를 Resize하는 Method로 Bilinear Interpolation Method를 사용한다. 또한 Image Processing 을 위해 C++ “Opencv” 라이브러리를 사용한다.

**Background Knowledge:**

**Bilinear Interpolation Method**는 새로운 픽셀위치에서의 주변 4개의 픽셀 값들의 weighted sum을 구해 그 값을 새로운 픽셀의 값으로 하는 방식이다.



***출저: NA\_Intepolation.pdf 강의자료***

위의 사진이 Bilinear Interpolation을 가장 이해하기 쉽게 설명한 사진이다. Destination 이미지에서 새롭게 값을 표현해 주어야할 위치를 (x,y)라고 할 때, Source 이미지에서의 (x,y)의 주변 4개 위치의 픽셀 값과 각각의 위치로부터의 가로, 세로 거리를 이용하여 각각 주변 픽셀의 값을 거리에 비례하게(거리가 가까우면 높은 weight, 거리가 멀면 낮은 weight) weighted Sum을 구해 (x,y)의 값으로 해준다. 이 과정을 Destination 이미지의 모든 픽셀 마다 반복 해주어 이미지 전체의 픽셀 값을 구해준다.

**Process:**

(Skeleton Code인 **‘forInterpolation.cpp’** 파일을 편집하여 구현하였다.)

Skeleton Code내부에 이미 image를 읽어와 cv::Mat type의 자료로 저장을 하는 부분이 이미 구현 되어있고, 이미지 창을 띄우는 부분까지 모두 구현되어 있어서, 읽어온 source image를 Bilinear Interpolation Method를 이용하여 원하는 크기의 이미지로 변환시키는 부분만 추가해주면 되었다. 이 부분은 “BilinearInterpolation()”함수로 선언 되어있고, 함수의 내부는 구현이 되지 않은 상태여서 이부분만 구현을 해주면 되었다.

“BilinearInterpolation()”함수를 구현하는데 있어서 두가지 방법을 사용하였다. 한가지 방법은 이미 Opencv 라이브러리에 존재하는 함수인 “cv::resize()” 함수를 이용하여 간단하게 하는 방법이고, 또 다른 방법은 이 resize()와 비슷한 함수를 Bilinear Interpolation의 공식을 사용하여 직접 구현하는 방법이다.

첫번째 방법으로 구현하는 부분은 단순히 “cv::resize()”함수를 한줄 추가해주는 것으로 구현이 완료 되었다. 이때 Interpolation 방법으로 “cv::INTER\_LINEAR”을 인자로 넘겨주어 Bilinear Interpolation을 사용하게끔 한다. 두번째 방법의 구현은 Bilinear Interpolation의 수식을 다 대입하여 Destination 이미지의 모든 픽셀 마다 각각의 픽셀 값을 새롭게 구하는 방식으로 구현하였는데, 이때 주변 픽셀 4개와의 거리와 비례하는 비율로 4개 픽셀 값의 Weighted Sum을 구하여 값을 구하는 방식이다.

또한 “main()” 함수 안에서 source 이미지의 size를 출력해주는 부분을 추가하고 resize하고 싶은 이미지의 크기 (height \* width)를 각각 입력 받아 입력 받은 숫자의 크기로 Source 이미지의 크기를 변형시킨 다음 위에서 언급된 두가지 방법의 함수를 사용한 결과를 각각 출력해 주는 부분을 추가하였다.

*< hw08.cpp >, (편집된 부분은 빨간색으로 표시)*

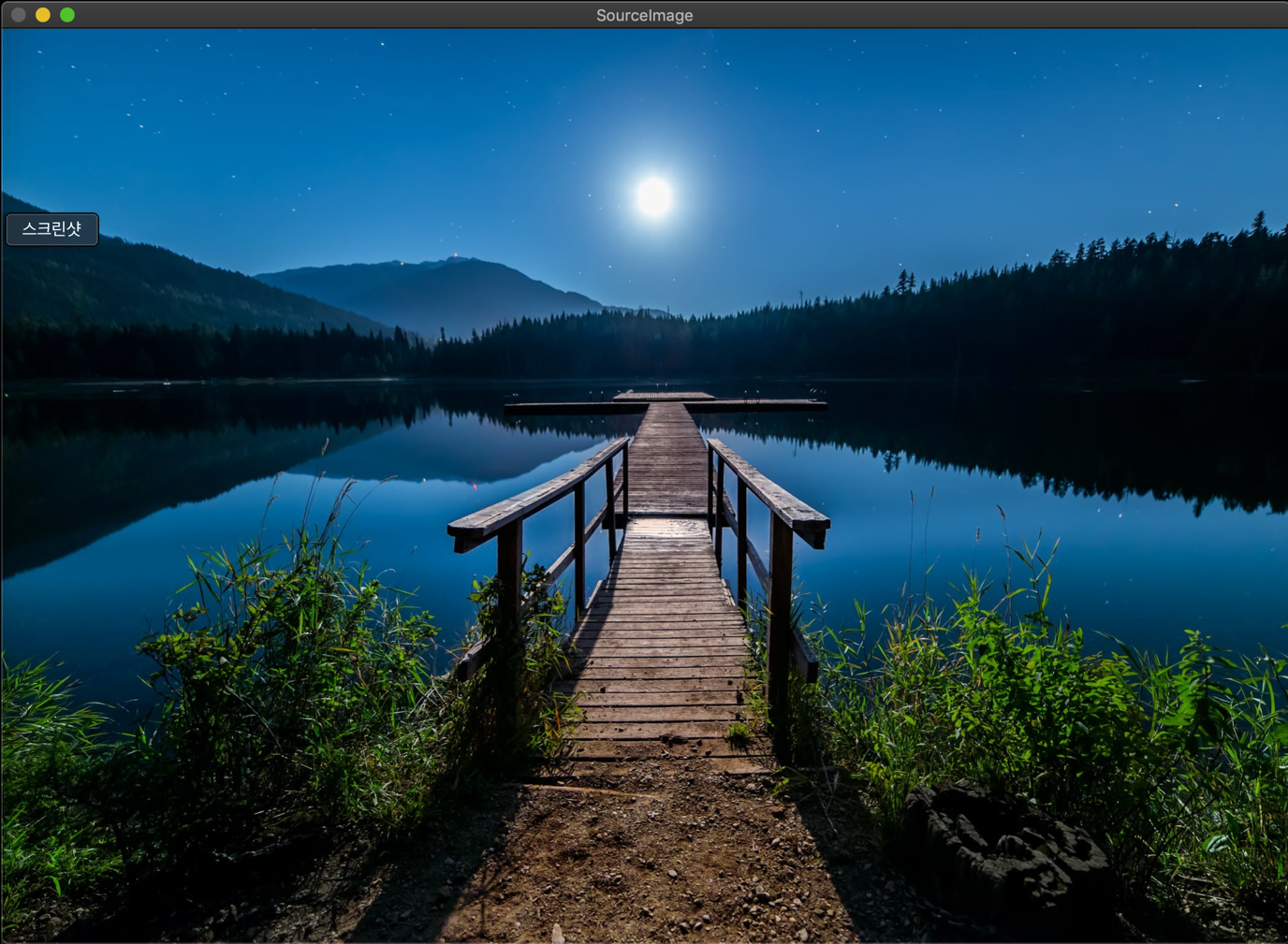
|  |
| --- |
| **//file I/O and resize with interpolation**  **#include <opencv2/opencv.hpp>**  **#include <iostream>**  **using namespace std;**  **using namespace cv;**  **//the place where edition is needed.**  **void BilinearInterpolation(cv::Mat &src, cv::Mat &dst){**  **// bilinear interpolation**  **// using built-in-function in OPENCV;**  **cv::resize(src, dst, cv::Size( dst.cols, dst.rows ), 0, 0, cv::INTER\_LINEAR);**  **}**  **// implementing Bilinear interpolation NOT using built-in OPENCV function**  **void BilinearInterpolationMyown(cv::Mat &src, cv::Mat &dst){**  **// bilinear Interpolation**  **// implementing MyOwn Resize function**    **//-------------IndexError-------------------//**  **//double rate\_of\_Height = dst.rows/src.rows;**  **//double rate\_of\_Width = dst.cols/src.cols;**    **//-----------Solved Index Error Problem---//**  **double rate\_of\_Height = (double)dst.rows/(double)src.rows;**  **double rate\_of\_Width = (double)dst.cols/(double)src.cols;**    **for(int y = 0;y < dst.rows;y++){**  **for(int x = 0; x< dst.cols;x++){**    **//src image's index of Mat src[px][py] (in order to get pixel value)**  **int px = (int)(x / rate\_of\_Width);**  **int py = (int)(y / rate\_of\_Height);**    **// calculating a, (1-a), b, (1-b)**  **double fx0 = (double)x/(double)rate\_of\_Width - (double)px;**  **double fx1 = 1 - fx0;**  **double fy0 = (double)y/(double)rate\_of\_Height - (double)py;**  **double fy1 = 1 - fy0;**    **double area00 = fx1\*fy1; //(1-a)\*(1-b)**  **double area01 = fx0\*fy1; //a\*(1-b)**  **double area10 = fx1\*fy0; //(1-a)\*b**  **double area11 = fx0\*fy0; //a\*b**    **if(src.channels() == 1){//when src image is gray image**  **unsigned char P00 = src.at<unsigned char>(py,px);**  **unsigned char P01 = src.at<unsigned char>(py,px+1);**  **unsigned char P10 = src.at<unsigned char>(py+1,px);**  **unsigned char P11 = src.at<unsigned char>(py+1,px+1);**  **dst.at<unsigned char>(y,x)= area00\*P00+area01\*P01+area10\*P10+area11\*P11;**  **}**    **else{ //case of RGB image**  **Vec3b P00 = src.at<cv::Vec3b>(py,px);**  **Vec3b P01 = src.at<Vec3b>(py,px+1);**  **Vec3b P10 = src.at<Vec3b>(py+1,px);**  **Vec3b P11 = src.at<Vec3b>(py+1,px+1);**  **dst.at<Vec3b>(y,x)= area00\*P00+area01\*P01+area10\*P10+area11\*P11;**  **}**  **}**  **}**  **}**  **cv::Mat bi\_dst; // for output of built-in function**  **cv::Mat bi\_dst2; // for output of Self-Implemented MyOwn function**  **int main()**  **{**  **cv::Mat src;**  **src = cv::imread("example.jpeg", 1);**  **if (src.empty())**  **{**  **std::cout << "Cannot find an image" << std::endl;**  **return -1;**  **}**  **cv::imshow("SourceImage", src);**  **cv::waitKey(0);**  **//¿ÃπÃ¡ˆ∏¶ 2πË ≈∞ø¸¿ª ∂ß¿« Bilinear interpolation**  **int height = src.rows;**  **int width = src.cols;**    **//------------- edited --------------//**  **printf("\nOriginal Image size : %d x %d \n", height , width );**    **int newHeight, newWidth;**  **std::cout << "\nDesired scale of height : "<< endl;**  **std::cin >> newHeight;**  **std::cout << "\nDesired scale of width : " << endl;**  **std::cin >> newWidth;**  **printf("\ndesired size of picture: %d x %d\n",newHeight, newWidth);**    **bi\_dst = cv::Mat(newHeight, newWidth, src.type(), cv::Scalar(0));**  **bi\_dst2 = cv::Mat(newHeight, newWidth, src.type(), cv::Scalar(0));**    **//bilinearInterpolation**  **BilinearInterpolation(src, bi\_dst);**  **BilinearInterpolationMyown(src, bi\_dst2);**  **cv::imshow("BIImage\_of\_built-in-function", bi\_dst);**  **cv::imshow("BIImage\_of\_Myown\_function", bi\_dst2);**  **cv::waitKey(0);**  **return 0;**  **}** |

**Results & Discussion:**

*/\*\*실행환경: (IDE: “Xcode”, Macbook Pro 2015 Retina)\*\*/*

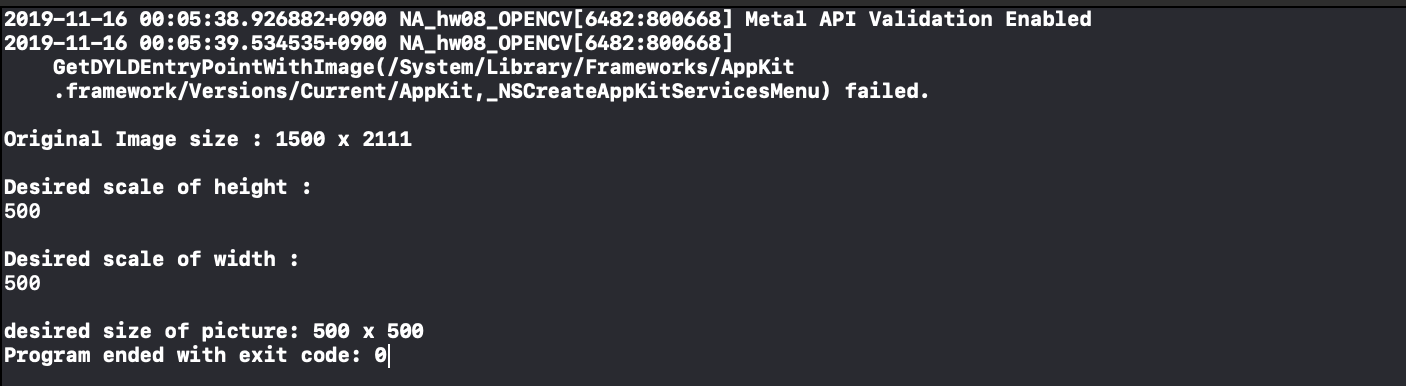
**-실행(1) 결과: (image: example.jpeg)**

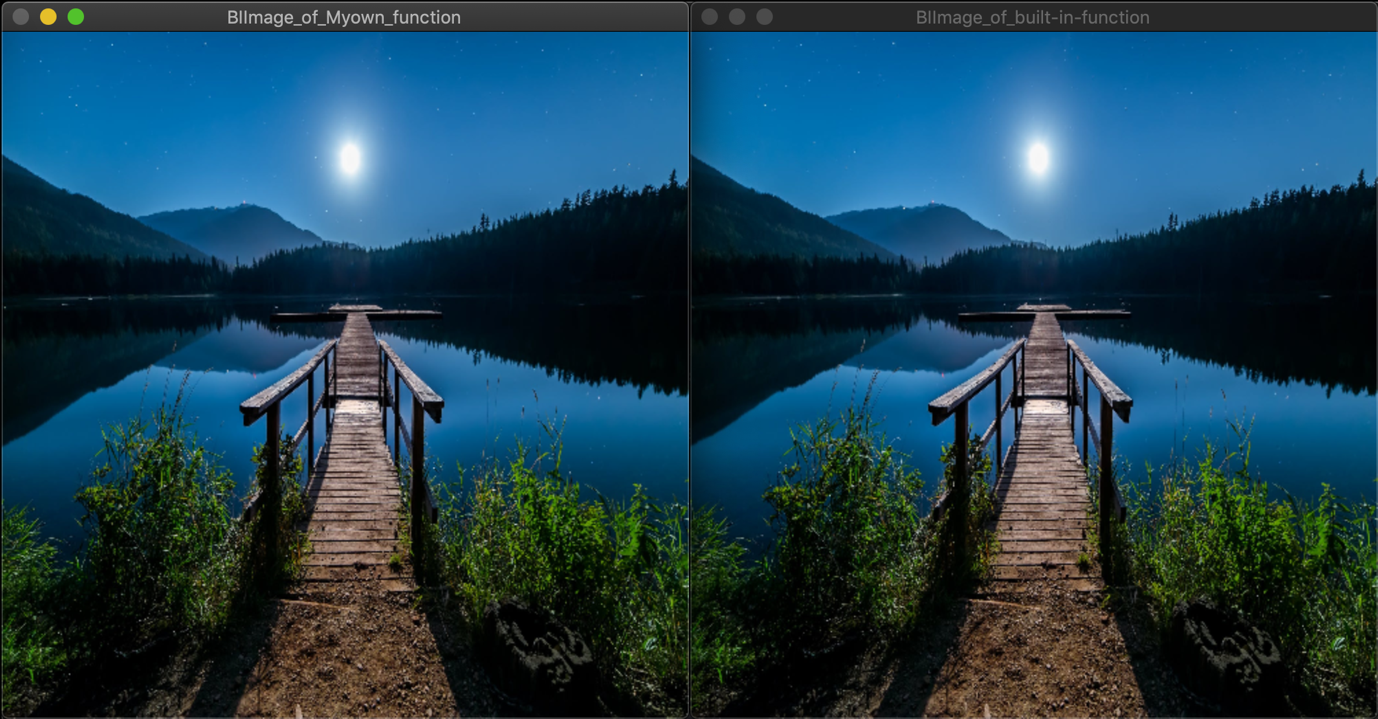
1. **Source Image출력:**

****

(원본 이미지의 크기가 1500x2111로 너무 커서 실제로 노트북 모니터 상에서는 원래의 크기보다 조금 작은 사이즈로 출력된다.)

1. **원하는 size (height \* width)를 각각 입력 후 결과 이미지 출력:**

****

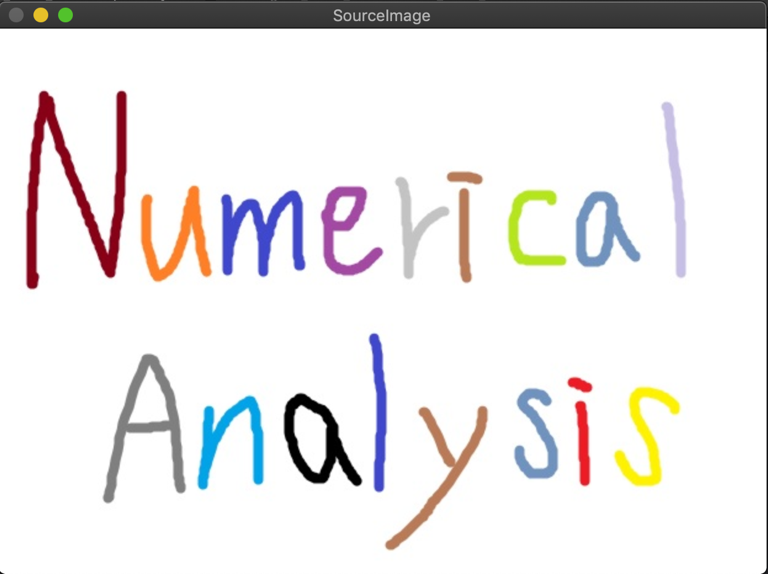
****

resize된 이미지들은 500x500 크기에 맞게 잘 출력된다. 두개의 결과 이미지 중 오른쪽에 있는 이미지가 opencv의 “resize()”함수를 이용한 결과 이미지이고, 왼쪽에 있는 이미지가 직접 수식을 대입하여 구현한 “BilinearInterpolationMyOwn()”함수를 이용한 결과 이미지이다.

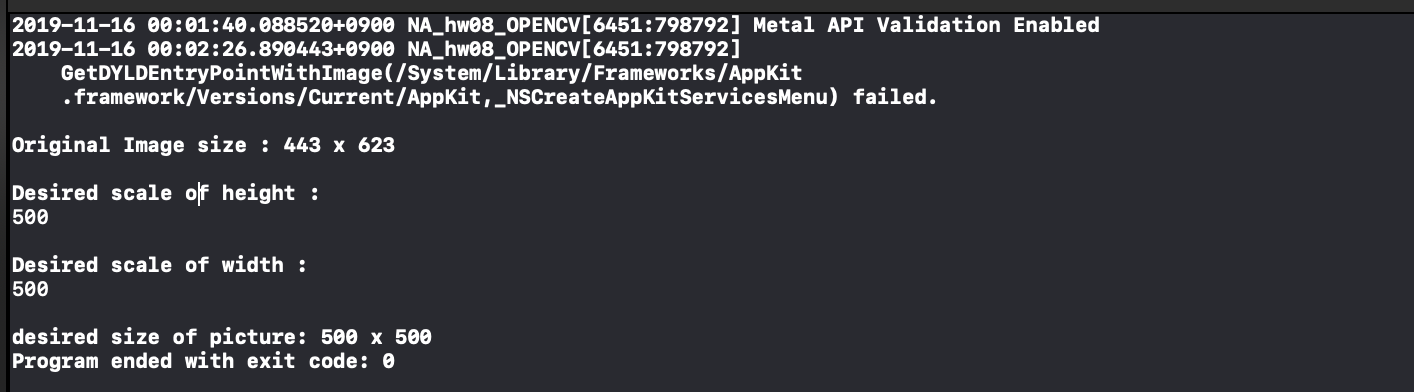
두 이미지가 육안으로 확인 했을 때 거의 99% 똑같은 사진으로 보이기 때문에 직접 구현된 “BilinearInterpolationMyOwn()”함수가 오류없이 잘 구현 되었다고 볼 수 있다.

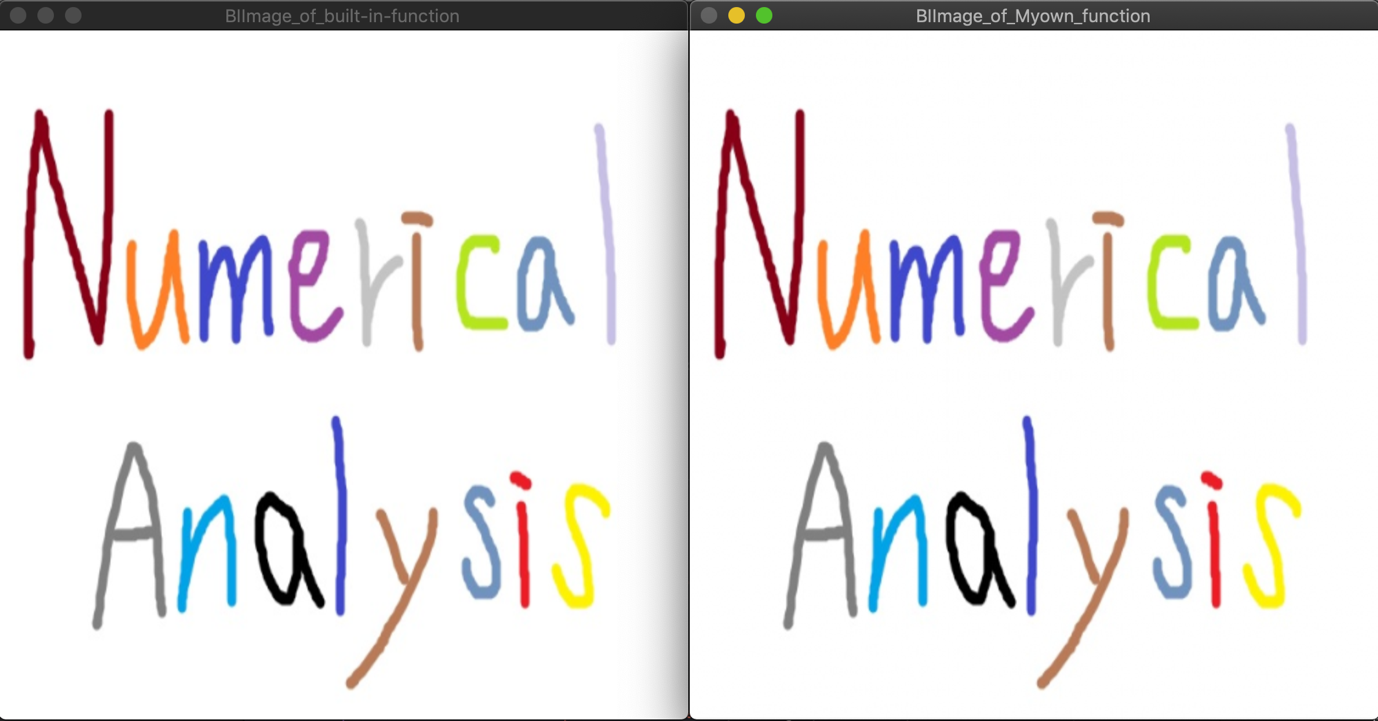
**-실행(2) 결과: (image: masking1.jpg)**

1. **Source Image출력:**

****

1. **원하는 size (height \* width)를 각각 입력 후 결과 이미지 출력:**

****

****

이번에도 역시 resize된 이미지들이 500x500 크기에 맞게 잘 출력되고 Source 이미지와 출력된 창에서 크기 비교를 해봤을 때 알맞은 크기 비율로 출력되는 것을 확인 할 수 있었다.

두개의 결과 이미지 중 왼쪽에 있는 이미지가 opencv의 “resize()”함수를 이용한 결과 이미지이고, 오른쪽에 있는 이미지가 직접 수식을 대입하여 구현한 “BilinearInterpolationMyOwn()”함수를 이용한 결과 이미지이다.

이번에도 두 이미지가 육안으로 확인 했을 때 거의 99% 똑같은 사진으로 보이기 때문에 직접 구현된 “BilinearInterpolationMyOwn()”함수가 오류없이 잘 구현 되었다고 볼 수 있다.