

**과목명 | 디지털영상처리**

**실습 | 8**

**담당교수 | 정진우**

**학과 | 컴퓨터공학과**

**학년 | 3**

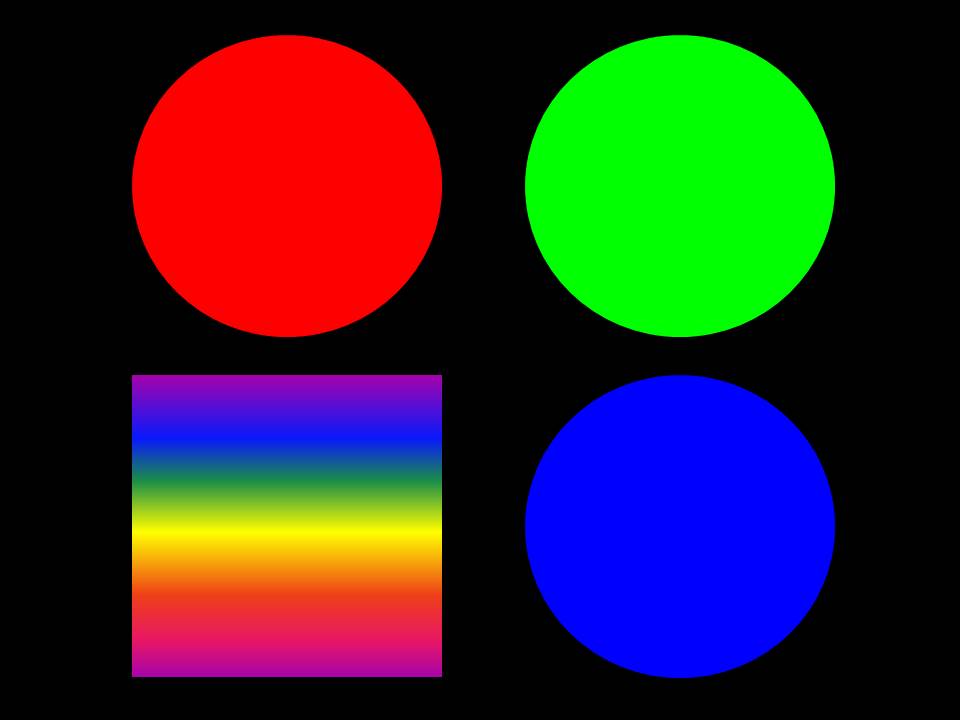
**학번 | 2010112406**

**제출일자 | 2016. 5. 12**

**이름 | 박도령**

**제출일 | 2016. 3. 31**

**실습1 에 사용한 이미지)**

****

**stuff\_color.jpg**

**실습 1)**

♋ **동작 순서 및 원리**

**1. Rotate the image by 45 degrees then enlarge the image 1.5 times. - A Image**

**이미지를 원 상태로 보존하면서 회전시키기 위해서는 많은 변수들이 필요하다. 설정한 변수들을 이미지를 회전시키는 Opencv 함수 warpAffine의 매개 변수로 사용할 수 있다. 해당 실습의 대부분은 이 이 함수를 사용하여 이미지 회전을 구현한다.**

**1) 회전시키려는 각도 : 45**

**//Rotatation angle**

**double angle = 45;**

**2) 회전의 기준이 되는 좌표(중심점)**

**//Gets rotatation point located on center of image A**

**Point2f img\_A\_center{ img\_A.cols / 2.0f, img\_A.rows / 2.0f };**

**3) 회전을 위한 2X3 Affine 변환 행렬(회전의 기준이 되는 좌표를 중심점을 적용)**

**//Gets rotation matrix with angle and point**

**Mat rotation\_matrix = getRotationMatrix2D(img\_A\_center, angle, 1.0);**

**4) 회전된 이미지를 담을 수 있는 Bounding Rect**

**//Rectangle to be background of Rotated Image**

**Rect bbox = RotatedRect(img\_A\_center, img\_A.size(), angle).boundingRect();**

**5) Affine 변환 행렬에 대한 연산**

**//Adjusts new option to rotation matrix**

**rotation\_matrix.at<double>(0, 2) += bbox.width / 2.0 - img\_A\_center.x;**

**rotation\_matrix.at<double>(1, 2) += bbox.height / 2.0 - img\_A\_center.y;**

**6) 이미지에 warpAffine 함수 적용(해당 옵션으로 이미지 회전)**

**//Rotates Image A by 45 degree with rotation matrix**

**warpAffine(img\_A, img\_A, rotation\_matrix, bbox.size());**

**이미지를 1.5배 스케일링 하기 위해서는 Opencv 함수 resize를 사용한다. INTER\_CUBIC 옵션을 적용하여 Cubic Interpolation을 적용하였다.**

**//Scaling 1.5 times**

**resize(img\_A, img\_A, Size(img\_A.cols \* 1.5, img\_A.rows \*1.5), 0, 0, INTER\_CUBIC);**

**2. Enlarge the image 1.5times then rotate the image by 45 degrees - B Image**

**위의 1의 과정의 순서를 바꾸어 적용한다. 먼저 원본 이미지를 1.5배 스케일링 한 뒤 45도로 회전 시킨다.**

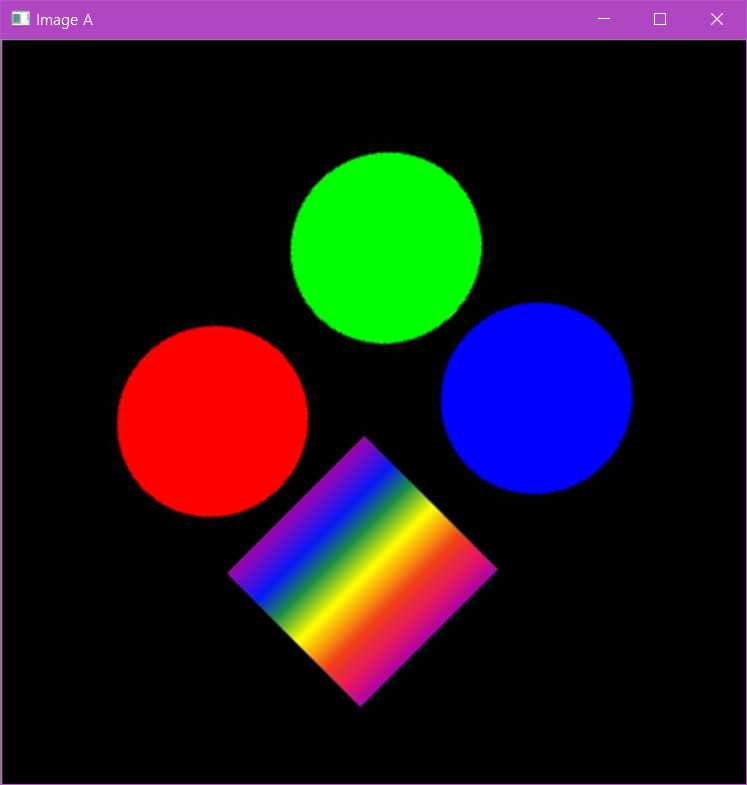
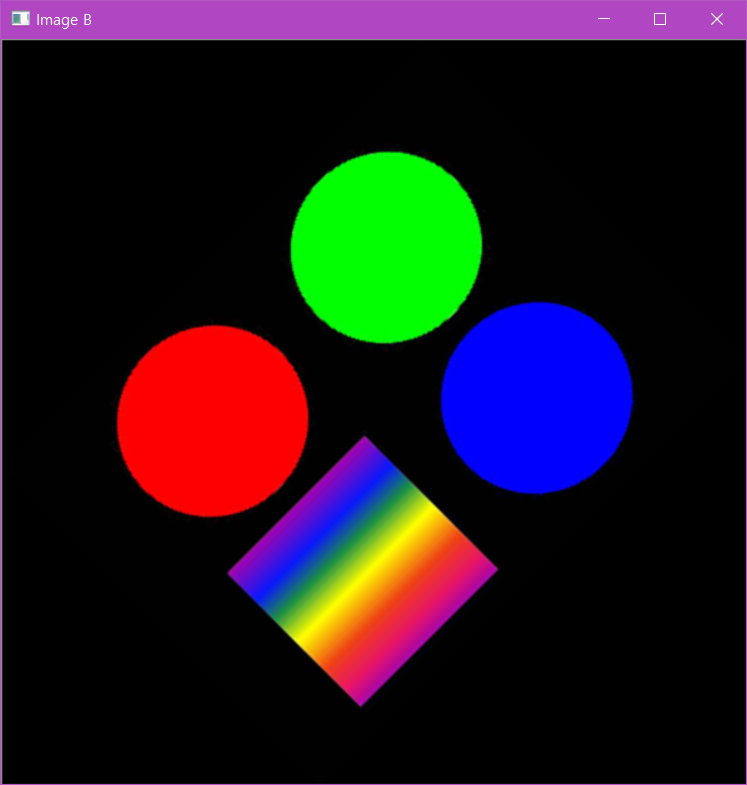
**3. Subtract B image from A image and check the result.(If two images are exactly same, the result would be black image.)**

**2의 과정에서 얻은 이미지 B에 1의 과정에서 얻은 이미지 A를 뺀다. 만약 두 이미지가 같다면 검정색 화면이 나올 것이다.**

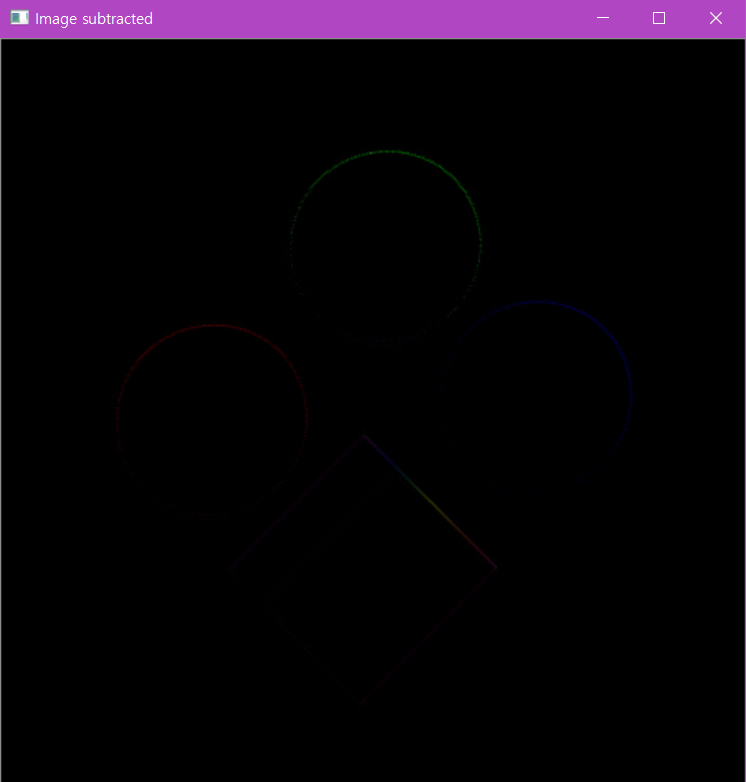
**//Subtract Image B from Image A**

**img\_subtracted = img\_B - img\_A;**

**♋ 결과 이미지 (Image\_A.jpg, Image\_B.jpg)**

**♋ 결과 이미지 (Image\_subtracted.jpg)**



**♋ 설명**

**두 이미지는 겉으로 보기에는 동일해 보이지만, 두 이미지를 뺀 결과를 보았을 때 완벽하게는 일치하지 않는 것을 확인할 수 있다. 그 이유는 내부적인 Interpolation 과정에서 추정되는 픽셀의 위치가 순서에 따라 약간씩 달라지기 때문일 것이다. 회전 과정에서 1차적으로 Interpolation이 일어나고, 1.5배 Scaling 과정에서 2차적으로 Interpolation이 일어나게 되는데 순서가 바뀔 경우 영향을 받는 픽셀도 달라질 수 있을 것이다.**

**실습 2)**

**[stuff\_color.jpg] - 실습 1과 동일**

♋ **동작 순서 및 원리**

**실습 2에서는 영상을 30도 회전하는 동작이 많이 쓰이기 때문에 사용자 정의 함수를 작성하여 사용하였다. 이 함수는 실습 1의 이미지 회전 과정과 동일하며, 매개 변수 Image를 n번 30도 회전시킨다.**

**//Rotates Image by 30 degree**

**void rotate\_by\_30(Mat& img, const int n) noexcept{**

**double angle = 30;**

**Point2f img\_center;**

**Mat rotation\_matrix;**

**Rect bbox;**

**for (int i = 0; i < n; ++i) {**

**//Rotation point located on center of Image**

**img\_center = Point2f{ img.cols / 2.0f, img.rows / 2.0f };**

**//Rotation matrix created with angle and point**

**rotation\_matrix = getRotationMatrix2D(img\_center, angle, 1.0);**

**//Rectangle to be background of Rotated Image**

**bbox = RotatedRect(img\_center, img.size(), angle).boundingRect();**

**//Adjusts new option to rotation matrix**

**rotation\_matrix.at<double>(0, 2) += bbox.width / 2.0 - img\_center.x;**

**rotation\_matrix.at<double>(1, 2) += bbox.height / 2.0 - img\_center.y;**

**//Rotates Image**

**warpAffine(img, img, rotation\_matrix, bbox.size());**

**}**

**}**

**1. Rotate the image by 30 degrees.(clockwise) / Repeat Rotating the result image again 11 more times.**

**구현한 사용자 정의 함수를 사용하여 이미지를 30도로 1번 회전시킨다.**

**//Rotates image by 30degree with 1 time**

**rotate\_by\_30(img\_manipulated, 1);**

**구현한 사용자 정의 함수를 사용하여 이미지를 30도로 11번 회전시킨다. 이로써 이미지에 총 360도 회전이 이루어 졌다.**

**//Rotates image by 30 degree with 11 times (total : 12 times)**

**rotate\_by\_30(img\_manipulated, 11);**

**2. Subtract A image from the original image and check the result.**

**(If two images are exactly same, the result would be black image)**

**원본 이미지에 360도 회전시킨 이미지를 뺄셈한 결과를 확인한다. 뺄셈하기에 앞서 360도 회전시킨 이미지는 회전할 때마다 이미지 크기를 바꾸는 과정이 발생하므로 원본 이미지보다 크기가 훨씬 더 크다. 따라서 원본 이미지 크기에 맞게 크기를 잘라주는 과정이 필요하다. 원본 이미지와 동일한 크기를 정 가운데에 잘라낸다.**

**//Rectangle area to crop**

**Rect crop{(img\_manipulated.cols - img.cols)/2, (img\_manipulated.rows - img.rows)/2 , img.cols, img.rows };**

**//Crop area with Rectangle(Same size with Original Image)**

**img\_360\_1time = img\_manipulated(crop);**

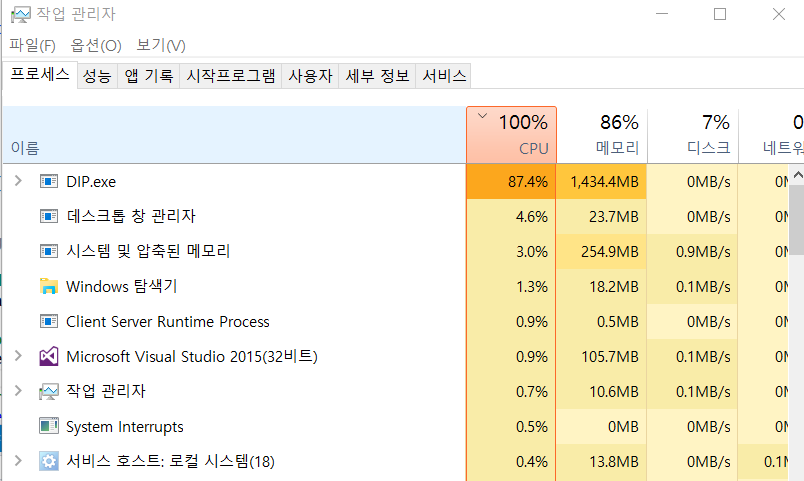
**원본 이미지에 회전 이미지를 뺄셈한다. 만약 두 이미지가 같다면 결과는 검정색 화면이 될 것이다.**

**//Gets Substracted Image**

**img\_subtracted = img - img\_360\_1time;**

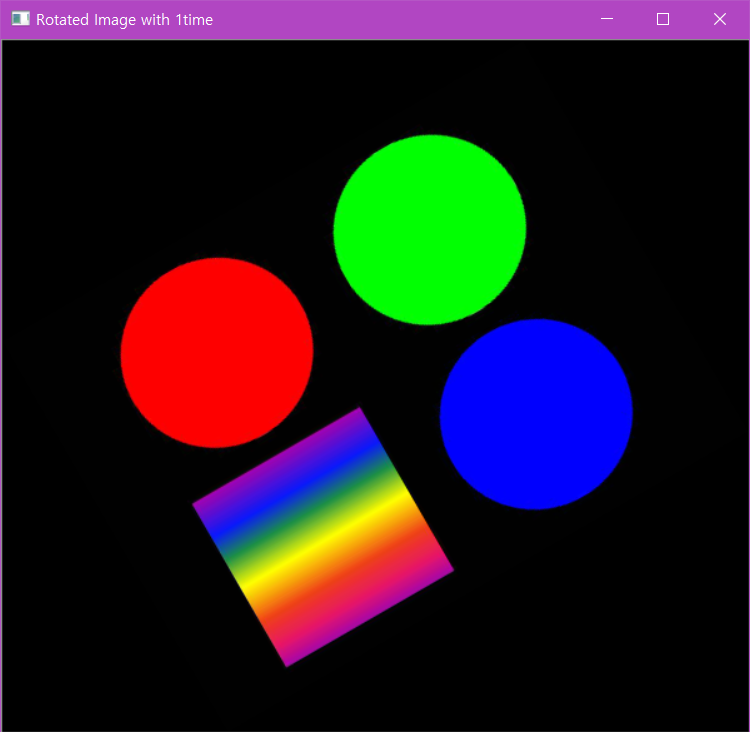
**3. Repeat the first step (1) of Prob.2 more than five times and then repeat (2) again.**

**이 과정을 수행해보려고 하였으나 컴퓨터에 극한의 오버헤드를 발생시키고 지나치게 메모리 공간이 사용된 데에도 불구하고 시간이 너무 오래 걸리므로 더 이상 진행할 수 없었다. 이미지를 30도씩 총 60번 이상 회전시켜야 하기 때문이다. 따라서 코드를 작성은 해보았으나 결과를 확인해 볼 수 없었다.**

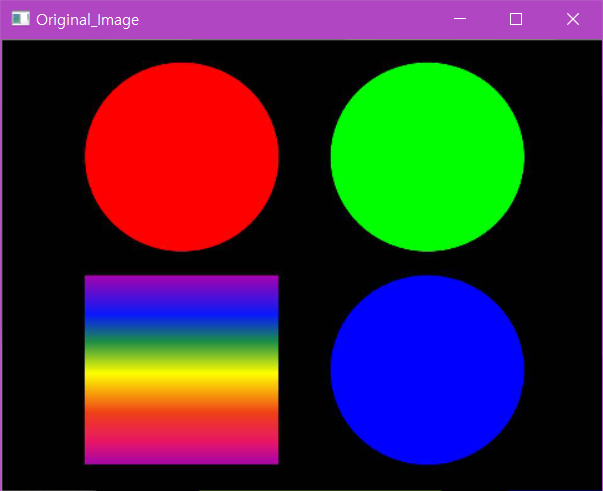


**따라서 프로그램을 5번 돌려서 30도씩 12번 회전을 총 5번 실행한 결과 이미지와 원본 이미지와 해당 결과 이미지를 뺀 결과 이미지를 얻을 수 있었다.**

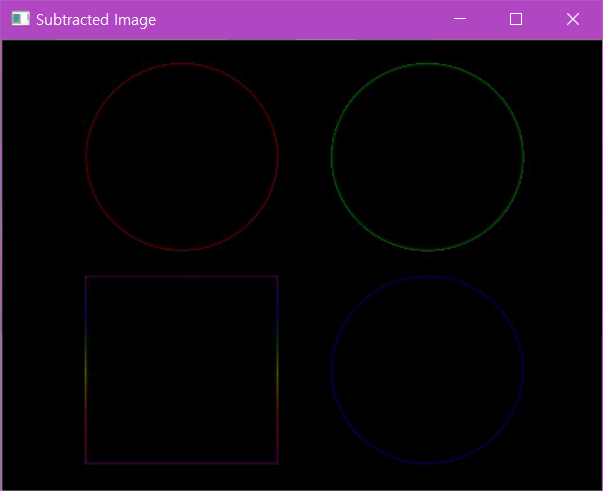
**♋ 결과 이미지(Rotated\_1time.jpg)**



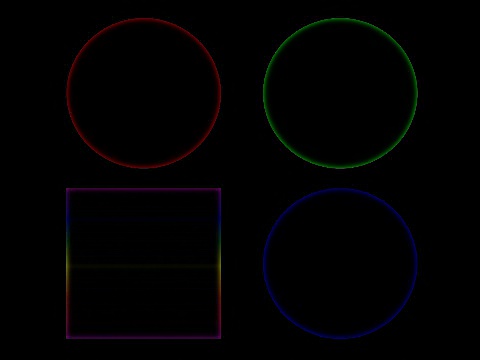
**♋ 결과 이미지( [원본 이미지], Rotated\_12times.jpg )**

**♋ 결과 이미지(Image\_substracted \_from\_rotated.jpg)**



**♋ 결과 이미지(Rotated\_360\_5times.jpg, Rotated\_360\_5times\_subtracted.jpg)**

** **

**♋ 설명**

**두 이미지 역시 겉으로 보기에는 동일해 보이지만 두 이미지를 뺄셈할 경우 완벽하게 동일하지 않음을 확인할 수 있다. 이미지를 회전할 때마다 빈 공간이 발생하기 때문에 빈 공간을 채우기 위한 보간(Interpolation) 과정이 필요하다. 따라서 위의 실습 같은 경우 빈 공간이 채워지는 과정에서 겉 부분에 해당되는 일부 픽셀이 회전과정에서 색깔이 있는 픽셀로 추정되어 채워졌음을 확인할 수 있다. 따라서 원본 이미지와 원본 이미지를 360도 돌린 결과 이미지는 완벽하게 동일하지 않다는 결론을 내릴 수 있다. 해당 과정을 5번 실행하는 과정에서는 이미지가 조금 더 팽창하다가 어느 순간부터 일정해 지게 되는 현상을 발견할 수 있었다. 하지만 결론적으로 1번 돌린 결과보다 5번 돌린 결과가 이미지 바깥쪽의 경계 부분에 더 많은 색깔 픽셀이 채워졌다는 것을 확인할 수 있었다.**