컴퓨터비전 HW01 Report

이름: 권도현

학번: 2023065350

학과: 컴퓨터소프트웨어학부

1. Projective Image Transformation

Code Analysis

```
5   def image_difference(image1, image2, name):
6       d = np.sum((image1 - image2) ** 2)
7       diff = d / float(512 * 512)
8       print(f"{name}\'s difference: {diff}")
9
10  def printImage(image, name):
11      plt.imshow(image)
12      plt.title(name)
13      plt.show()
```

- **image_difference**: Rendering에서 MSE를 사용하는 것을 보고, 이미지 간 비교에서도 MSE를 사용하겠다는 판단에서 단순하게 픽셀별로 비교할 수 있는 함수를 구현하였다.
  - 구체적인 함수는 아니어서 소수점 아래의 숫자를 조정하기 위해서만 이용하였다.
- **printImage:** 반복되는 부분을 코드를 간략화 하기 위해 정의하였다.

```
def rotation(I, a, name): #I: image, a: angle
    M = np.array(
        [[np.cos(a), -np.sin(a), 1],
        [np.sin(a), np.cos(a), 1]], dtype=np.float32)

    rotation_image = cv2.warpAffine(I, M, (0, 0))

    printImage(rotation_image, name)

    return rotation_image
```

- opencv library의 warpAffine 함수를 통해 Last low = [0, 0, 1] 인 Matrix와의 행렬곱을 구현할 수 있었다.
- Rotation angle만 입력받으면 나머진 고정되므로, angle만 입력 받을 수 있도록 하였다.

```python
def similarity(I, s, a, x, y, name):
    M = np.array(
        [[s*np.cos(a), -s*np.sin(a), x],
         [s*np.sin(a), s*np.cos(a), y]], dtype=np.float32)

    similarity_image = cv2.warpAffine(I, M, (0,0))

    printImage(similarity_image, name)

    return similarity_image
```

- Rotation matrix에서 Image scale, translation (x, y)를 추가하였다.

```python
def affine(I, scale, shear, name):
    M = np.array(
        [[scale, shear, 0],
         [shear, scale, 0]], dtype=np.float32)

    affine_image = cv2.warpAffine(I, M, (0, 0))

    printImage(affine_image, name)

    return affine_image
```

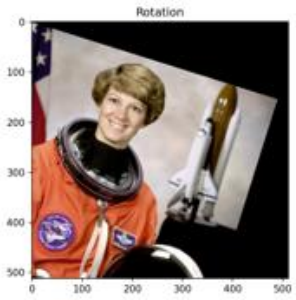
- 원칙적으로는 translation과 rotation도 구현되어야 하지만, 우리가 만들어야 되는 사진을 보았을 때, translation과 rotation은 필요 없다고 생각해서 scale과 shear만을 parameter로 구현하였다.

```python
def projective(I, M, name):
    projective_image = cv2.warpPerspective(I, M, (0, 0))

    printImage(projective_image, name)

    return projective_image
```

- Projective는 last row가 [0, 0, 1] 이 아니므로 opencv 라이브러리의 warpPerspective 함수를 사용하였다.
- Projective Matrix의 8개 element를 모두 parameter로 받기에는 많기도 하고, 각 element의 의미를 표현하기도 애매하여 외부에서 정의한 행렬은 parameter로 받을 수 있도록 하였다.

Test Image Results & Matrix Parameters

1. **Rotation**

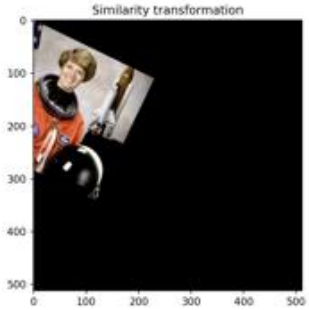



```
r_test = cv2.imread("rotation_test.jpg")
rotation_test = cv2.resize(r_test, (512, 512))
image_difference(rotation(image, 0.305, "Rotation"), rotation_test, "Rotation transformation")
```

```
Rotation transformation's difference: 173.6497802734375
```

- 정답 케이스를 봤을 때, 시계방향으로 회전이라 angle을 양수로 잡았다.
- Cos, sin 함수 안에 들어가는 값을 radian (1 radian = 약 57도)이고, 눈대중으로 보았을 때 회전 각도가 28도보다 작아 보였다. 처음에 시도해본 값은 0.4이고 각도가 정답 케이스보다 커서 점점 줄여가다가 0.3 부근에서 가장 비슷한 것을 확인할 수 있었다.
- 3.05는 0.01 단위로 바꾸며, image_difference 함수의 결과값이 가장 작은 것을 선택한 것이다.

2. Similarity

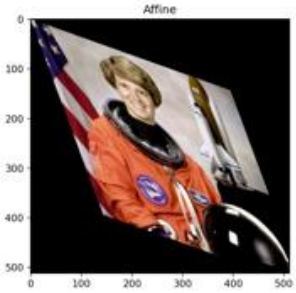



```
s_test = cv2.imread("similarity_test.jpg")
similarity_test = cv2.resize(s_test, (512, 512))
image_difference(similarity(image, 0.5, 0.45, -1, -1, "Similarity transformation"), similarity_test,
```

```
Similarity's difference: 47.8091926574707
```

- Rotation에서 사용한 angle보다는 커 보이고, Rotation의 angle에 가장 처음에 시도해본 0.4일 때의 각도와 비슷해 보여서 0.4부터 시도했다. 0.4보다는 커야 될 것 같아 0.01단위씩 증가시켜 0.45를 찾을 수 있었다.
- 각도를 먼저 찾고 보니, 정답 케이스와 비슷해지기 위해서는 이미지의 크기가 작아져야 된다고 생각했다. 가장 간단하게 절반으로 줄여보았다.
- 마지막으로 이미지에 미세한 조정이 필요했다. 1만큼 이동하면 얼마나 이동하는지 확인해보기 위해서 tx = -1을 넣어봤다. y축만 조절하면 해결될 것 같아 y축에도 값을 넣어보다 ty = -1을 찾게 되었다.

3. Affine

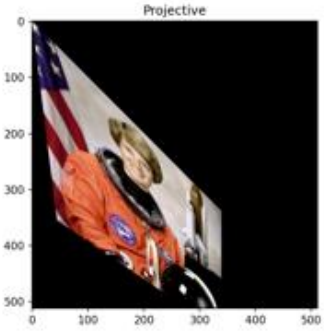



```
a_test = cv2.imread("affine_test.jpg")
affine_test = cv2.resize(a_test, (512, 512))
image_difference(affine(image, 0.77, 0.28, "Affine"), affine_test, "Affine transformation")
```

```
Affine transformation's difference: 133.7771339416504
```

- 앞서 이야기한 것처럼, rotation이나 traslation은 필요 없어 보였다. shearing먼저 시도해보았고, 이미지가 한쪽 방향으로 쏠리지 않도록 하려면 x, y에 대한 shearing value가 동일해야 된다고 생각했다. 0.3 부근에서 유사하게 기울어지는 것을 확인했다.
- Shearing을 하고 보니 이미지가 일부분 잘렸다. 내가 원하는 이미지는 잘린 부분이 있으면 안 되므로 Scale을 줄일 필요가 있다고 생각했다. 마찬가지로 여러 개 시도해보다가 0.8 부근에서 비슷한 것을 확인했다.
- 각각 값을 바꿔가며 image_diffence의 결과가 가장 작은 값으로 확정했다.

4. Projective




```
p_test = cv2.imread("projective_test.jpg")
projective_test = cv2.resize(p_test, (512, 512))
m = np.array(
        [[1, 0.1, 0],
         [1, 0.8, 0],
         [0.001, 0.0003, 1]], dtype=np.float32)
image_difference(projective(image, m, "Projective"), projective_test, "Projective transformation")
```

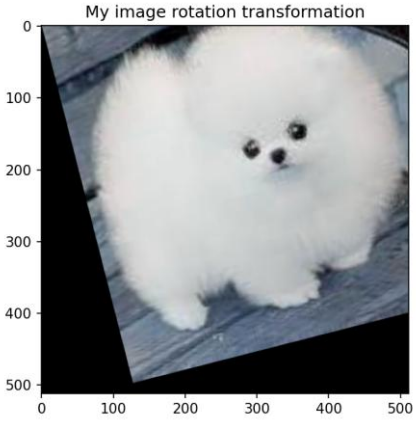
```
Projective transformation's difference: 103.17375183105469
```

- 우선 Traslation은 할 필요가 없다고 판단했다.
- Last row를 통해 이미지의 가장 윗 변이 y=x 선에 대응되어야 한다는 것에 집중했다.
  - 처음에는 큰 숫자를 넣었다가 원본 이미지의 x, y에 대응되는 x′, y′ 점의 분모가 너무 커져 거의 보이지도 않았다.
  - 이후, 굉장히 작은 값을 넣어 가장 유사한 값을 얻었다.
- 두 번째로, 우측 하단 쪽의 이미지가 잘리는 문제를 해결하기 위해 scale조정을 했다.
- 마지막으로, 우측을 바라보게 shearing되기 위해서 첫 번째 행을 조정했다.

- 큰 흐름은 위와 비슷한데, image_diffence 함수에 가장 많이 의존해서 하나씩 parameter를 찾아갔다.

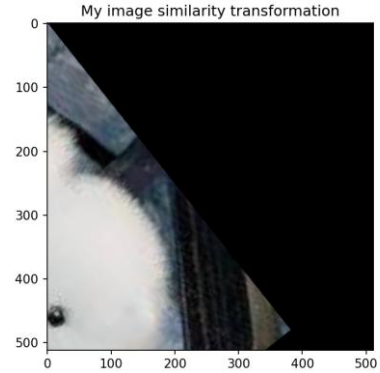
My image results

1. Rotation



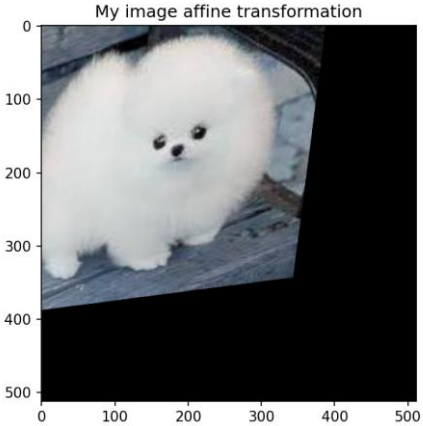
Angle에 음수(-0.25)를 사용하였다. Test image와 다르게 반시계 방향으로 회전하는 것을 확인할 수 있다.

2. Similarity



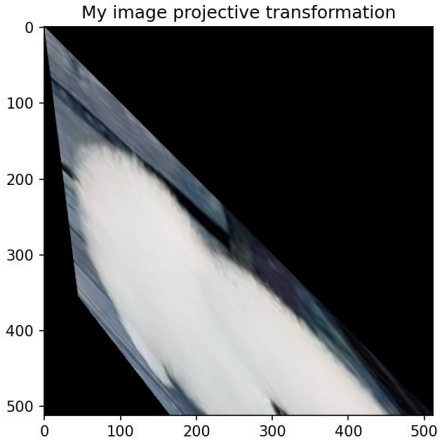
Similarity에서는 Rotation angle 값을 2배 늘리고, Scale도 원본 이미지의 1.2배로 설정했다.

3. Affine



Test image와 scale은 그대로, shearing만 음수 (-1)로 바꿔봤다. 기존에는 좌하단에서 우상단 방향으로 기울어졌던 것과 달리, 좌상단에서 우하단 방향으로 기울어진 것을 확인할 수 있다.

4. Projective



Projective transformation의 가장 큰 특징인 Last row의 영향을 확인하기 위해, Last row를 제외한 다른 row는 유지하고, Last row만 바꾸어 봤다. 사용한 행렬은 아래와 같다.

```
pm = np.array(
        [[1, 0.1, 0],
         [1, 0.8, 0],
         [-0.001, 0.0003, 1]], dtype=np.float32)
```

X′의 분모에 들어가는 값이 작아져 X′ 값이 전체적으로 증가하여 기존 Projective Transformation image보다 X축 상단으로 왜곡된 것을 확인할 수 있다.

2. Color Space

Code Analysis

```
def printForOneChannel(image, name, option):
    plt.imshow(image, cmap = option)
    plt.title(name)
    plt.show()

def printForAllChannels(image, name):
    plt.imshow(image)
    plt.title(name)
    plt.show()
```

각각 1 channel 인 경우, 3 channel 인 경우에 결과 이미지를 출력할 수 있도록 하는 함

수이다.

```
class RGB:
    def __init__(self, r, g, b):
        self.R = r
        self.G = g
        self.B = b

    def make_rgb(self):
        return self.R, self.G, self.B

    def blue_incresing(self, value):
        self.B = np.add(self.B, value)
        self.B = np.clip(self.B, 0, 255).astype(np.uint8)
        return self.B

    def printRGB (self):
        printForOneChannel(self.R, "Red", "gray")
        printForOneChannel(self.G, "Green", "gray")
        printForOneChannel(self.B, "Blue", "gray")
```

Blue의 값을 변화시키는 함수를 구현했고, 위의 printForOneChannel 함수를 이용해 RGB 채널 이미지를 출력할 수 있도록 하였다.

```
class HSI():
    def __init__(self, r, g, b):
        self.R = r.astype(float)
        self.G = g.astype(float)
        self.B = b.astype(float)
        self.h = np.zeros((512, 512), dtype=float)
        self.s = np.zeros((512, 512), dtype=float)
        self.i = np.zeros((512, 512), dtype=float)
        self.newR = np.zeros((512, 512), dtype=float)
        self.newG = np.zeros((512, 512), dtype=float)
        self.newB = np.zeros((512, 512), dtype=float)

    def make_hsi(self):
        Rn = self.R / 255.0
        Gn = self.G / 255.0
        Bn = self.B / 255.0

        for i in range(512):
            for j in range(512):
                # Intensity
                self.i[i][j] = (Rn[i][j] + Gn[i][j] + Bn[i][j]) / 3

                # Saturation
                if (self.i[i][j] == 0): self.i[i][j] += 0.0001
                self.s[i][j] = 1 - (np.min([Rn[i][j], Gn[i][j], Bn[i][j]]) / self.i[i][j])

                # Hue
                temp = (2 * np.sqrt((Rn[i][j] - Gn[i][j]) ** 2 + (Rn[i][j] - Bn[i][j]) * (Gn[i][j] - Bn[i][j])))
                if (temp == 0): temp += 0.0001
                angle = (2 * Rn[i][j] - Gn[i][j] - Bn[i][j]) / temp
                angle = np.clip(angle, -1, 1)
                self.h[i][j] = np.arccos(angle) / (2 * np.pi)
                if Bn[i][j] > Gn[i][j]:
                    self.h[i][j] = 1 - self.h[i][j]

        self.h = np.clip(self.h, 0, 1)
        self.s = np.clip(self.s, 0, 1)
        self.i = np.clip(self.i, 0, 1)

        return self.h, self.s, self.i
```

강의 PPT에 주어진 공식을 이용해서 RGB to HIS Trasnform을 수행하였다.

Saturation 값을 구할 때, Intensity 값이 0 이거나, Hue 값을 구할 때, 분모가 0 인 경우를 대비하여 0.0001 (매우 작은 값)으로 세팅하였다.

시간이 오래 걸리지만, 각 pixel 별로 Divide-by-zero error가 발생하는 것을 예방하고, 계산을 더 정확하게 하기 위해 이중 반복문으로 pixel 별 계산을 하였다.

np.clip()을 통해 HSI 값은 0 ~ 1 사이 값을 가지도록 하였다.

```
class HSI():
    def hsi_to_rgb(self):
        for i in range(512):
            for j in range(512):
                H = self.h[i][j] * 2 * np.pi
                S = self.s[i][j]
                I = self.i[i][j]

                # 0 <= H < 2π/3
                if 0 <= H < 2 * np.pi/3:
                    self.newB[i][j] = I * (1 - S)
                    temp = np.cos(np.pi/3 - H)
                    if temp == 0: temp += 0.0001
                    self.newR[i][j] = I * (1 + (S * np.cos(H)) / temp)
                    self.newG[i][j] = 3 * I - (self.newR[i][j] + self.newB[i][j])

                # 2π/3 <= H < 4π/3
                elif 2 * np.pi/3 <= H < 4 * np.pi/3:
                    H = H - 2 * np.pi/3
                    self.newR[i][j] = I * (1 - S)
                    temp = np.cos(np.pi/3 - H)
                    if temp == 0: temp += 0.0001
                    self.newG[i][j] = I * (1 + (S * np.cos(H)) / temp)
                    self.newB[i][j] = 3 * I - (self.newR[i][j] + self.newG[i][j])

                # 4π/3 <= H < 2π
                elif 4 * np.pi/3 <= H < 2 * np.pi:
                    H = H - 4 * np.pi/3
                    self.newG[i][j] = I * (1 - S)
                    temp = np.cos(np.pi/3 - H)
                    if temp == 0: temp += 0.0001
                    self.newB[i][j] = I * (1 + (S * np.cos(H)) / temp)
                    self.newR[i][j] = 3 * I - (self.newG[i][j] + self.newB[i][j])

        self.newR = (np.clip(self.newR, 0, 1) * 255).astype(np.uint8)
        self.newG = (np.clip(self.newG, 0, 1) * 255).astype(np.uint8)
        self.newB = (np.clip(self.newB, 0, 1) * 255).astype(np.uint8)

        return self.newR, self.newG, self.newB
```

https://www.youtube.com/watch?v=MobTlRgio3I 이 영상에 나오는 공식을 이용하여, HIS to RGB 를 계산하였다.

make_hsi()와 마찬가지로 numpy 내장 함수를 이용하는 대신 이중 반복문을 사용하였다.

```
def s_increasing(self, value):
    self.s = np.add(self.s, value)
    self.s = np.clip(self.s, 0, 1)
    return 0;
```

마지막으로 Saturation 값 조정을 위해 s_increasing() 함수를 구현하였다.

```python
class YCbCr():
    def __init__(self, r, g, b):
        self.R = r.astype(float)
        self.G = g.astype(float)
        self.B = b.astype(float)

    def make_ycbcr(self):
        self.y = (77*self.R + 150*self.G + 29*self.B) / 256.0
        self.cb = ((-43*self.R - 84*self.G + 127*self.B) / 256.0) + 128
        self.cr = ((127*self.R - 106*self.G - 21*self.B) / 256.0) + 128

        self.y = np.clip(self.y, 0, 255).astype(np.uint8)
        self.cb = np.clip(self.cb, 0, 255).astype(np.uint8)
        self.cr = np.clip(self.cr, 0, 255).astype(np.uint8)

    def printYCbCr (self):
        print("Y min:", np.min(self.y), ", Y max:", np.max(self.y))
        print("Cb min:", np.min(self.cb), ", Cb max:", np.max(self.cb))
        print("Cr min:", np.min(self.cr), ", Cr max:", np.max(self.cr), end = "\n\n")

        printForOneChannel(self.y, "Luminance", "gray")
        printForOneChannel(self.cb, "Chroma Blue", "gray")
        printForOneChannel(self.cr, "Chroma Red", "gray")
```

RGB to YCbCr Transformation을 강의 PPT 기반으로 구현하였다.

```python
image = skimage.data.astronaut()
# rgb
rgb = RGB(image[:,:,0], image[:,:,1], image[:,:,2])
R, G, B = rgb.make_rgb()
rgb.printRGB()

# RGB to YCbCr
ycbcr = YCbCr(R, G, B)
ycbcr.make_ycbcr()
ycbcr.printYCbCr()

# RGB to HSI
hsi = HSI(R, G, B)
h, s, i = hsi.make_hsi()

h = (np.clip(h, 0, 1) * 255).astype(np.uint8)
s = (np.clip(s, 0, 1) * 255).astype(np.uint8)
i = (np.clip(i, 0, 1) * 255).astype(np.uint8)

print("Hue min:", np.min(h), ", Hue max:", np.max(h))
print("Saturation min:", np.min(s), ", Saturation max:", np.max(s))
print("Intensity min:", np.min(i), ", Intensity max:", np.max(i), end = "\n\n")

printForOneChannel(h, "Hue", "gray")
printForOneChannel(s, "Saturation", "gray")
printForOneChannel(i, "Intensity", "gray")

# Increasing Blue
newB = rgb.blue_incresing(30)
BlueIncresedImage = np.stack((R, G, newB), axis=2)
BlueIncresedImage = np.clip(BlueIncresedImage, 0, 255).astype(np.uint8)
printForAllChannels(BlueIncresedImage, "Blue Increased Image")

# Increasing Saturation
hsi.s_increasing(0.2)
newr, newg, newb = hsi.hsi_to_rgb()
SaturationIncresedImage = np.stack((newr, newg, newb), axis=2)
SaturationIncresedImage = np.clip(SaturationIncresedImage, 0, 255).astype(np.uint8)
printForAllChannels(SaturationIncresedImage, "Saturation Increased Image")
```

Astronaut data에 대해서 과제를 수행하는 코드이다.

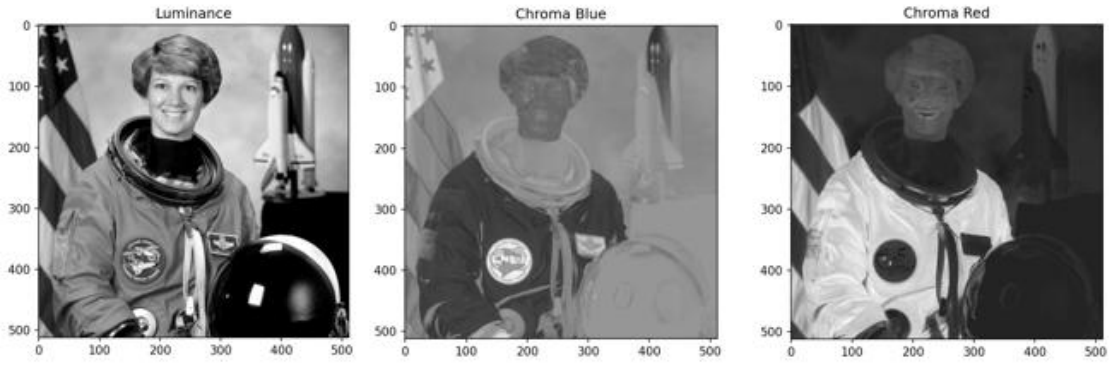
```python
img = cv2.imread("rainbow.jpg") # BGR 순서
myImage = cv2.resize(img, (512, 512))
printForAllChannels(myImage, "My image")

# my RGB
myRGB = RGB(myImage[:,:,2], myImage[:,:,1], myImage[:,:,0])
myRGB.printRGB()
myR, myG, myB = myRGB.make_rgb()

# my YCbCr
myYcbcr = YCbCr(myR, myG, myB)
myYcbcr.make_ycbcr()
myYcbcr.printYCbCr()

# my HSI
myHsi = HSI(myR, myG, myB)
myH, myS, myI = myHsi.make_hsi()

myH = (np.clip(myH, 0, 1) * 255).astype(np.uint8)
myS = (np.clip(myS, 0, 1) * 255).astype(np.uint8)
myI = (np.clip(myI, 0, 1) * 255).astype(np.uint8)

print("Hue min:", np.min(myH), ", Hue max:", np.max(myH))
print("Saturation min:", np.min(myS), ", Saturation max:", np.max(myS))
print("Intensity min:", np.min(myI), ", Intensity max:", np.max(myI), end = "\n\n")

printForOneChannel(myH, "Hue", "gray")
printForOneChannel(myS, "Saturation", "gray")
printForOneChannel(myI, "Intensity", "gray")

# my RGB image, blue decreasing
myNewB = myRGB.blue_incresing(-30)
myBlueDecresedImage = np.stack((myNewB, myG, myR), axis=2)
myBlueDecresedImage = np.clip(myBlueDecresedImage, 0, 255).astype(np.uint8)
printForAllChannels(myBlueDecresedImage, "My Blue Decreased Image")

# my HSI image, saturation decreasing
myHsi.s_increasing(-0.2)
myNewr, myNewg, myNewb = myHsi.hsi_to_rgb()
mySaturationDecresedImage = np.stack((myNewb, myNewg, myNewr), axis=2)
mySaturationDecresedImage = np.clip(mySaturationDecresedImage, 0, 255).astype(np.uint8)
printForAllChannels(mySaturationDecresedImage, "My Saturation Decreased Image")
```

내 Image에 대해서 과제를 실행하는 코드

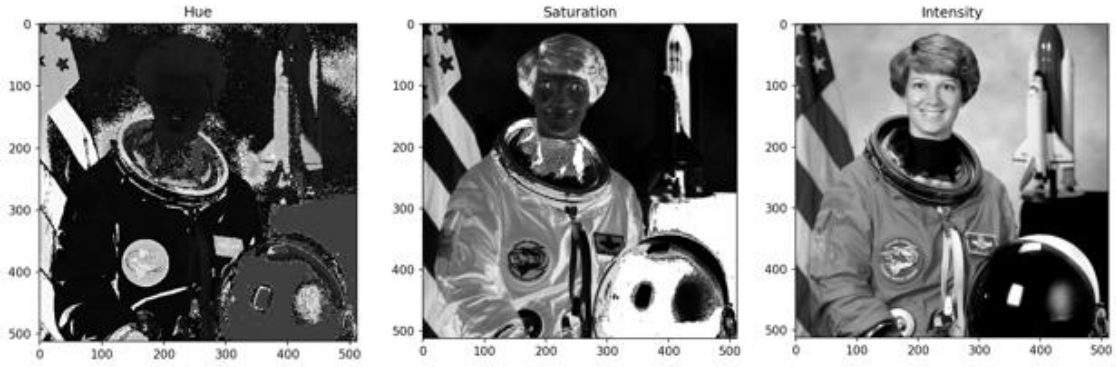
Test Image Results

1. RGB

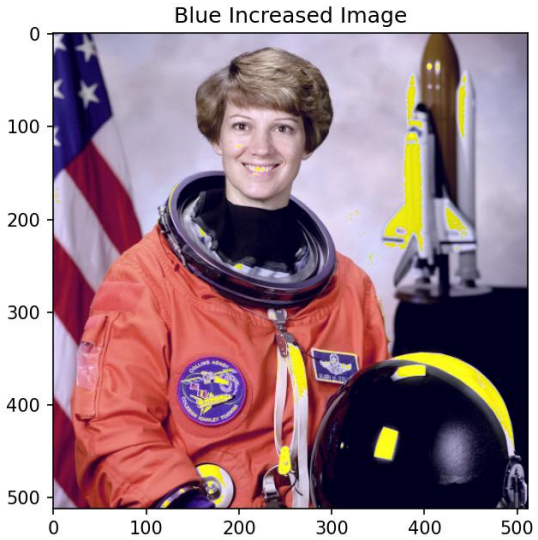
2. YCbCR



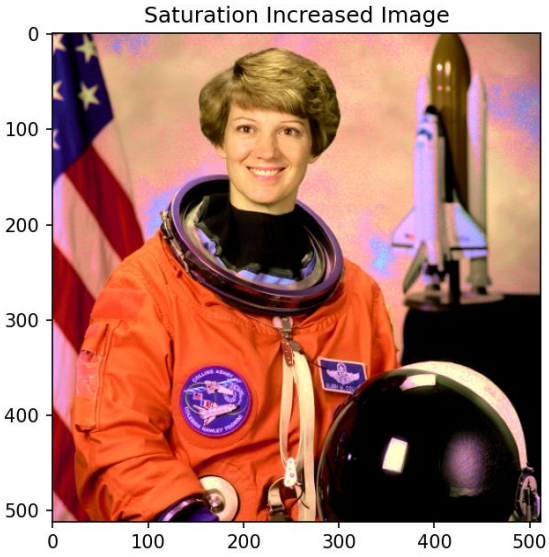
3. HIS



4. Increasing ‘Blue’ Value

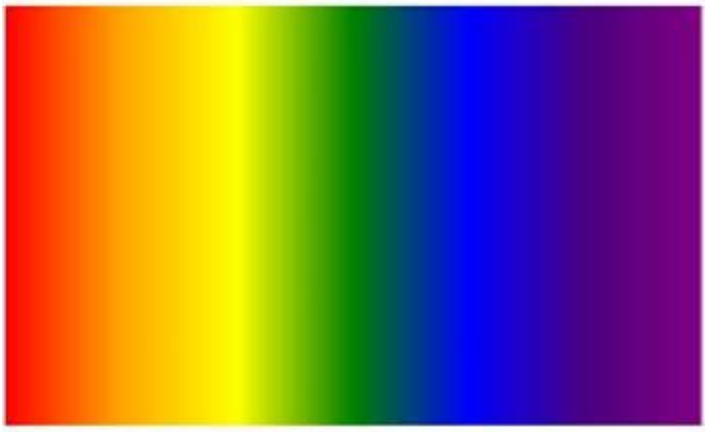


5. Increasing ‘Saturation’ Value



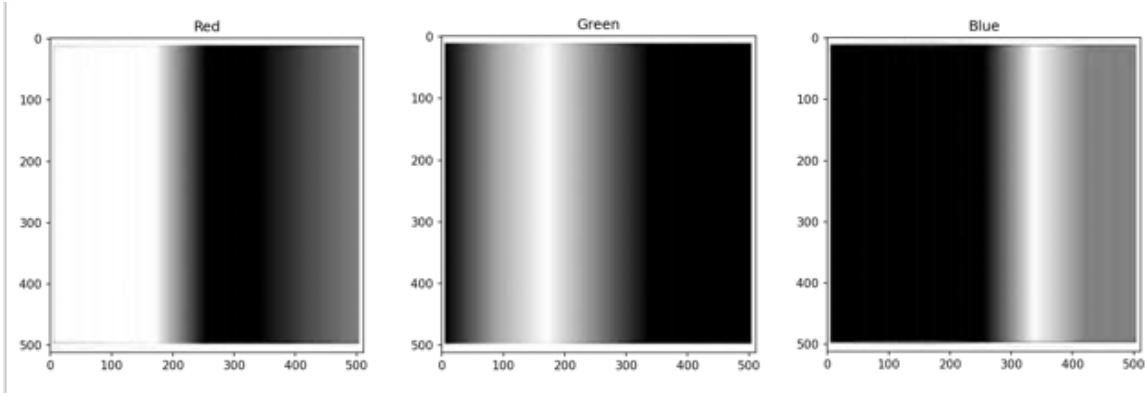
My image results

1. 원본 이미지

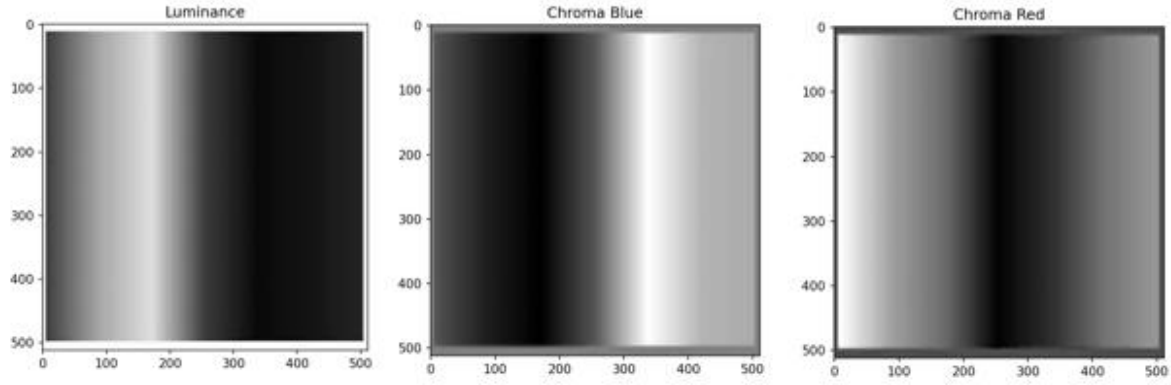


원래는 무지개 이미지였으나, cv2.imread() 상에서 b, g, r 순으로 RGB channel을 인식하여 plt.show()의 결과로 나오는 이미지 출력은 위 이미지와 달라진다. 하지만 아래의 분석은 이 이미지 기준으로 진행한다.

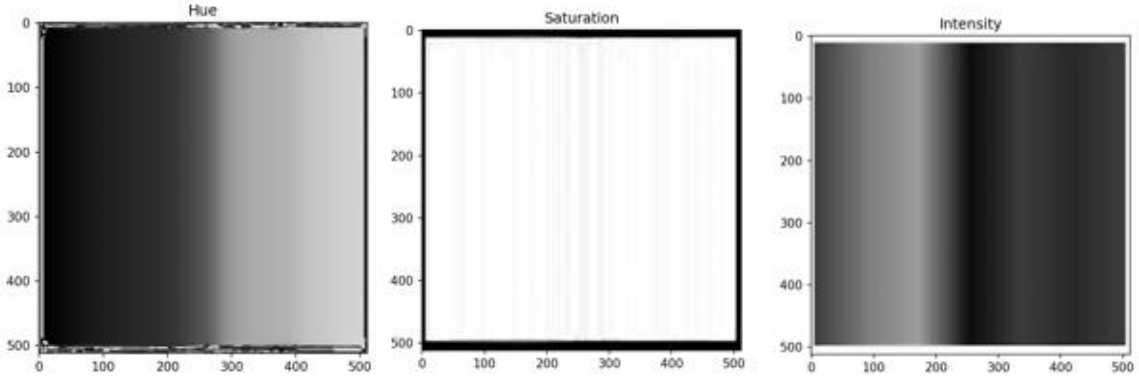
2. RGB



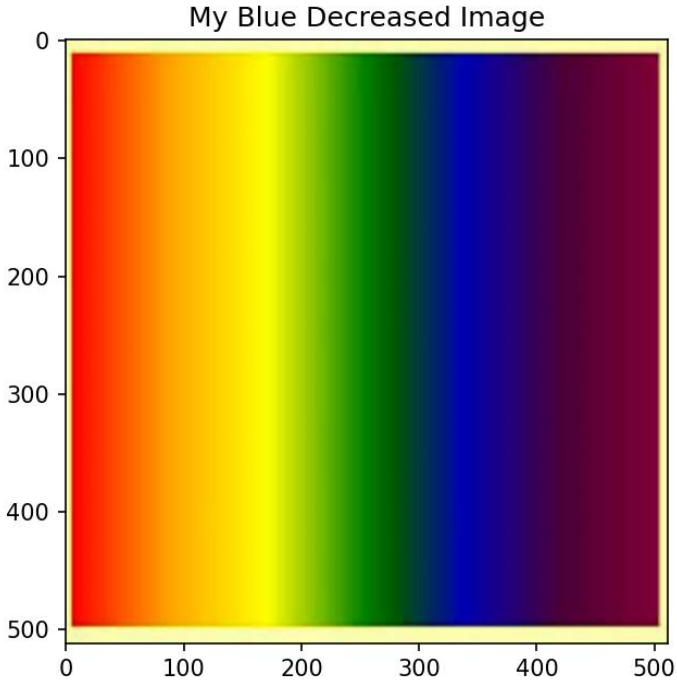
3. YCbCr



4. HSI

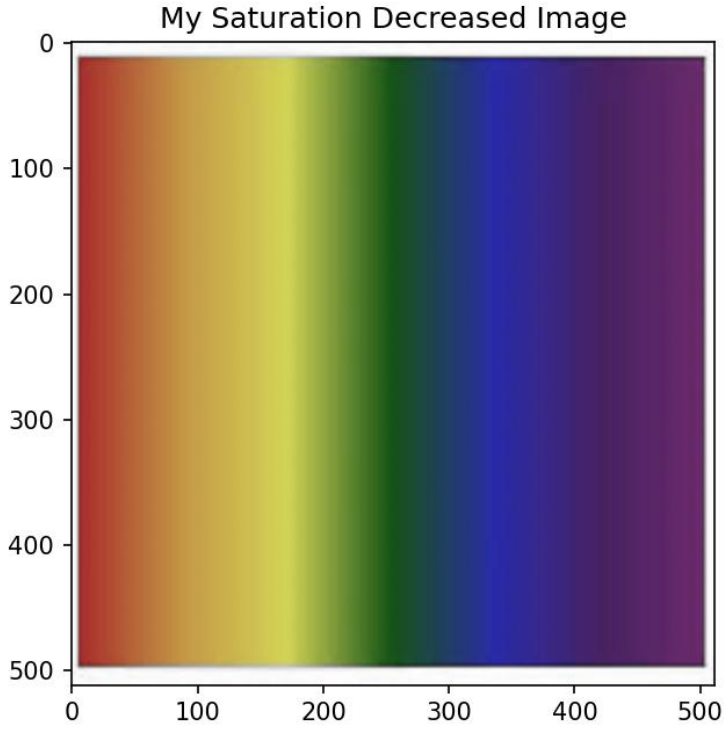


5. Decreasing ‘blue’ value



- 전체적으로 파란색이 감소되었다. 원래 흰색이었던 부분도 노란색(파란색의 보색)으로 채워진 것을 확인할 수 있다.

6. Decreasing 'Saturation' value

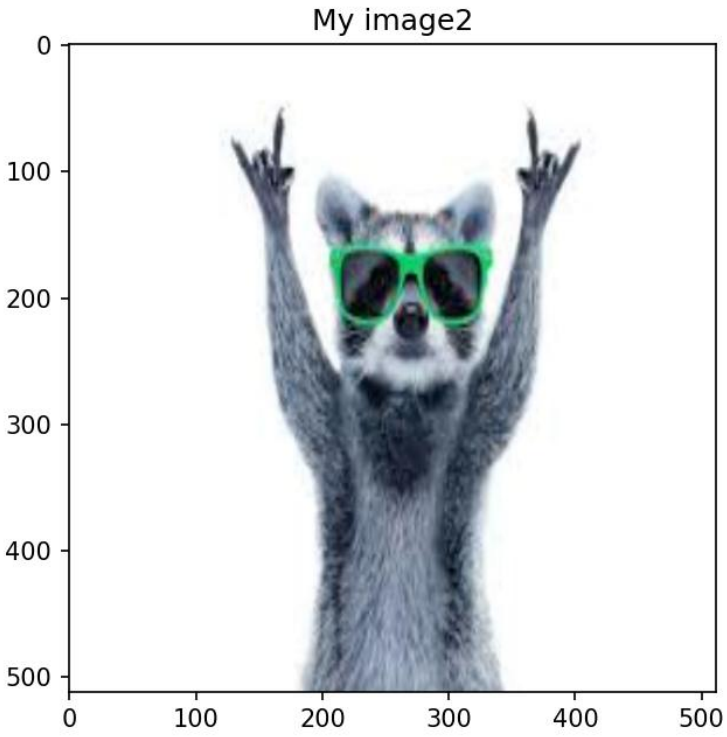


- 전체적으로 색의 선명도가 낮아졌다.

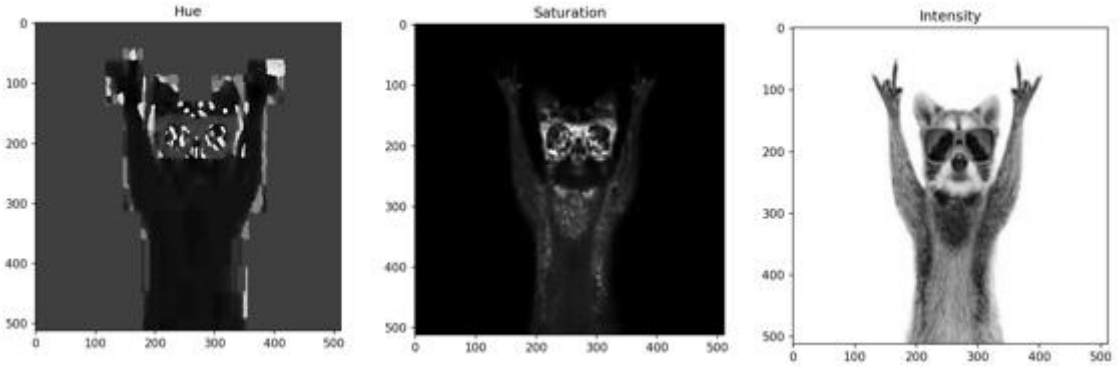
Additional test case for HSI

- 무지개 이미지를 사용해서 위의 HIS의 결과가 애매한 것 같아 다른 이미지를 사용하여 추가 테스트를 진행했다.

1. 원본 이미지



2. HSI



Comparing RGB, YCbCr, HSI

RGB channel: 각 channel이 R, G, B를 나타내는 값을 0 ~ 255까지의 수로 나타낸다. 각 채널이 맡은 색에 가까울수록 큰 값을 갖고, 밝게 나타난다.

- 테스트 이미지에 대해서, 우주인의 옷(주황색)과 얼굴 (살구색)이 R채널에서 비교적 밝게 나타난 이유이다.

- 내 무지개 이미지에선 각 R, G, B 채널 마다 어둡고 밝은 영역이 달라지는 이유이다.

YCbCr channel

- **Y channel**: 인간의 눈이 Green에 가장 민감한 것을 이용하여 G channel에 가중치를 크게 준 밝기 정보
  - 테스트 이미지에서는 초록색이 없어 흑백 이미지와 비슷하게 나타났다.
  - 내 이미지에서는 초록색 – 빨간색 부분이 밝게, 파랑 – 보라 부분이 어둡게 나왔다.
- **Cb channel**: 파란색과의 색깔 차이
  - 테스트 이미지에서는 성조기의 별 부분이 밝게 나왔다.
  - 내 무지개 이미지에서는 파란색이 밝게, 빨간색이 어둡게 나왔다.
- **Cr channel**: 빨간색과의 색깔 차이
  - 테스트 이미지에서는 우주인의 옷이 비교적 밝게 나왔다.
  - 내 무지개 이미지에서는 빨간색 부분이 밝게 나왔다.

HSI channel

- **H channel:** 빨간색 – 보라색을 원형으로 표현한 것
  - 테스트 이미지에서는 색이 구분되는 곳마다 밝기의 차이가 있는 것을 확인할 수 있다.
  - 내 무지개 이미지에서는 파란색을 기준으로 하여 원형으로 만들면 이어지는 원을 확인할 수 있다.
  - 내 너구리 이미지에서는 색깔이 다른 손톱, 선글라스 부분의 밝기만 다른 것을 확인할 수 있다.
- **S channel**: 채도, 색의 선명도
  - 테스트 이미지에서도 배경인 흰색은 검정색으로 나머진 밝게 나온 것을 확인할 수 있다.
  - 내 무지개 이미지에서는 무채색이 아니라 모두 원색이기 때문에 전체가 밝게 나왔다.
  - 내 너구리 이미지에서는 흰색, 검은색은 어둡게, 유일한 초록색인 선글라스 부분만 비교적 밝게 나온 것을 확인할 수 있다.
- **I channel**: 명도, 전체적인 밝기
  - 테스트 이미지와 내 이미지 모두 흑백 화면과 비슷하게 나왔다.

결론

RGB는 이미지의 색상 정보를을 직관적으로 표시하여 각 색이 얼만큼 있는지 밝기로 확인할 수 있다.

반면 YCbCr은 Y를 통해 전체적인 밝기 정보를 얻을 수 있고 Cb,Cr 을 통해 파란색, 빨간색을 기준으로 색상 정보를 확인할 수 있다. 밝기 정보와 색상 정보가 독립되어 표시된다는 장점이 있다.

마지막으로 HSI는 Hue를 통해서 색상 정보, Saturation을 통해 무채색 / 원색의 분포 정도, Intensity로 밝기 정보를 얻을 수 있다. 세 정보를 독립적으로 확인할 수 있다는 장점이 있다.

보완할 점

- 첫 번째 과제에서 내가 테스트 이미지에 대해서 생성한 이미지와 테스트 이미지를 비교하는 image_difference() 에서 더 구체적으로 구현하지 못 하고 단순 pixel별 비교만 한 것이 아쉬웠다.
- 테스트 케이스를 과제 명세서에서 스크린샷 해서 사용해서 스크린샷의 불균형, 해상도 저하, PNG -> JPG 변환 과정에서의 해상도 저하 등의 문제로 오차를 정확히 측정하지 못한 부분이 아쉬웠다.

- 두 번째 과제에서 Saturation increased image에서 계산 상의 오차, np.clip()에서 범위를 넘어가면 자르기, float와 int 변환 간의 오류 등으로 인해 완벽하게 구현되지 않고, noise가 발생한 부분이 아쉬웠다.