**※ 영상처리 11주차**

빛(가시광선) : 다양한 파장을 갖는 많은 광자의 조합

광자 : 전자장계의 가장 작은 당위의 양자

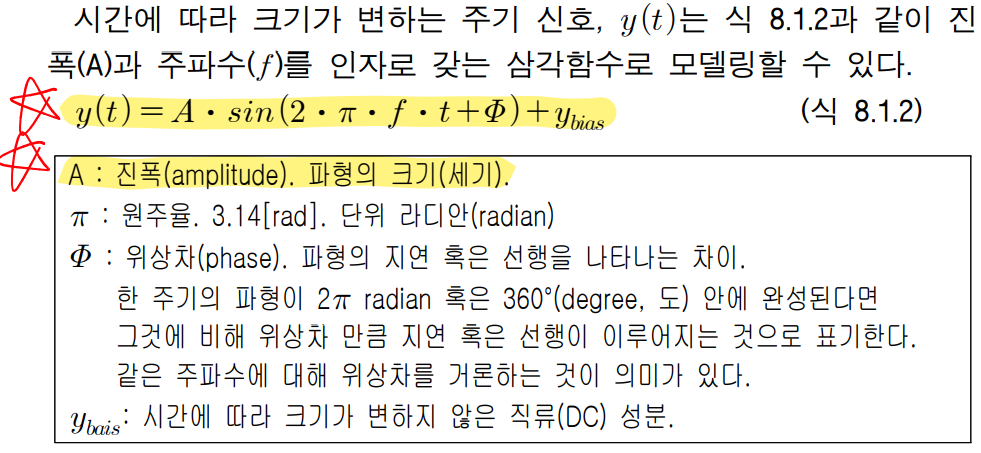
빛 : 파동처럼 주변을 휘젓고 다니지만, 입자처럼 부딪혀 에너지를 발산하게 만듦

-> 파동성과 입자성을 모두 가짐

주기 T : 하나의 파동(파형)이 완성되기까지의 시간[초]

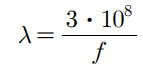
주파수 f : 1초당 만들어지는 파형의 수. 1초당 진동수[Hz].

f = 1/T T = 1/f => 주기와 주파수는 서로 역수 관계임



위상차 기호 : 파이라고 부름

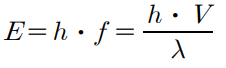
위상차에 대한 분석 : 같은 주파수를 갖는 신호에 대해 관찰하는 것이 의미가 있음

파장(Wavelength) - 파장 때문에 색이 차이가 나는 것

빛이 한 주기 동안 이동한 거리[m]를 나타냄

주기(T) : 시간 단위의 주기를 의미함

파장(λ) : 파형의 공간(거리) 관점의 주기를 의미함



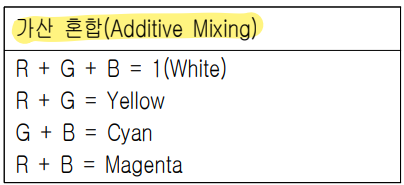
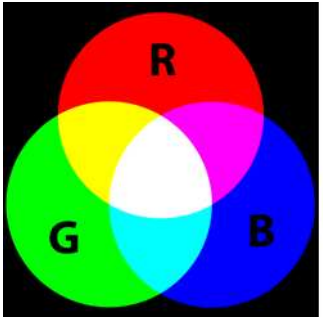
우측 식에서 얻을 수 있는 결론

주파수가 높거나 파장이 짧은 전자파 : 에너지가 큼

**RGB / CMYK 기반의 좌표계**

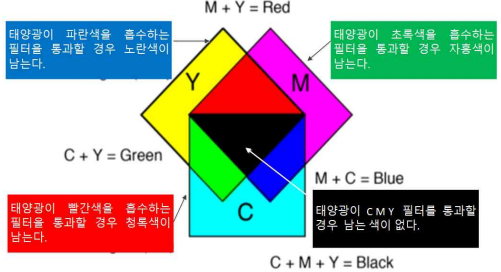
가산 혼합 : 각 구성 성분의 수치적 구성요소를 합산함으로써 색상을 만들어 내는 것

RGB 좌표계에서 색상 소스가 광원일 경우 이루어짐

응용사례 : LCD 스크린

감산 RGB 좌표계 – CMY, CMYK

감산 혼합



백색 광원의 앞에 특정 파장영역을 흡수하는 필터를 통과시킨 후의 파장 분포에 의해 색상이 이루어지는 것을 가정함

=> 특정 파장만 통과시키고 나머지는 차단함

잉크 소비량을 줄이기위해 CMY 보다는 검은색(K)를 추가한 CMYK 좌표계가 더 선호됨

K를 만들기 위해서는 CMY를 합쳐야하기 때문

색조, 채도 기반의 좌표계

컬러영상을 RGB 평면으로만 해석하는 데는 한계가 있음

-> 색조(Hue), 채도(Saturation), 명도(Brightness) 총 3개의 공간으로 나누는 좌표계 사용

색조, 채도를 사용하는 컬러 좌표계

HSV : Hue, Saturation, Value / HSB : Hue, Saturation, Brightness

HSL : H S, Lightness

HSI : H S Intensity

=> V B L I == Brightness

색조(Hue)의 개념

평면에 투영하여 생긴 색상의 변화를 나타낸 평면

중요 색상들의 값을 degree(ﾟ) 로 표현한 것

**크로마(Chroma, C) : 무채색을 제거한 순수 색상 성분**

무채색 생분 = RGB에서 가장 적은 값. m

화소의 대표 색상 = RGB에서 가장 큰 값. M

C = 무채색 성분을 제거한 순수 색상성분 -> C = M 색상에 중간 값이 가미된 순수 색상 성분

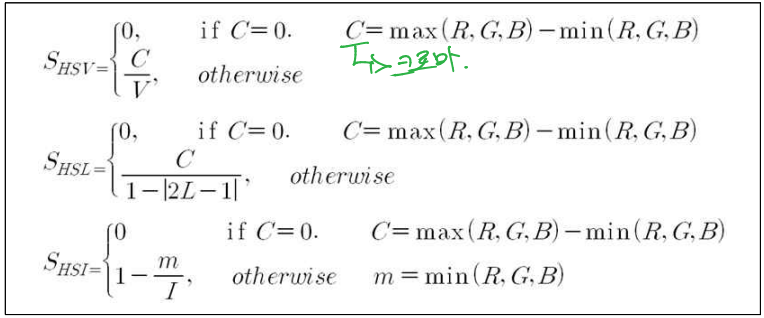
M = max(R,G,B) m = min(R,G,B) C = M – m

**채도(Saturation) :** **색상의 순수성을 의미함 -> 무채색(흑 혹은 백색)의 부재를 의미함**

무채색만 있으면 채도는 0. 무채색이 전혀 없으면 채도는 1

선명하지 않아도 채도가 1일 수 있음

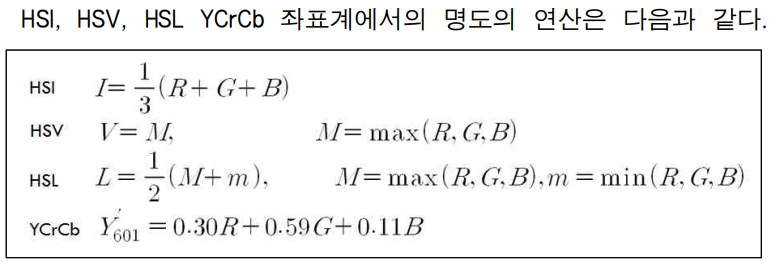
아래는 각각 HSV HSL HSI 좌표계일 경우의 채도 계산법



무채색 : RGB의 값들이 서로 같아야 만들어짐. -> RGB 세 값 중에서 최소값이 그 색상의 공통적으로 적용되는 무채색 성분이라고 할 수 있음

흰색이 만들어지려면 RGB의 공통 값이 있어야함 => 무채색은 RGB의 공통 값이 있어야함

명도의 연산



# 연산법은 함수로 구현이 다 되어있다고 언급하셨던 것으로 기억

**HSV 좌표계로의 변환**

opencv 변환 함수

dst = cv.cvtColor(src, code[, dstCn=0])

src: 입력영상. 8-bit/16-bit unsigned (CV\_16UC..), or 단정도 부동소수.

0 to 255 for CV\_8U images. full range를 사용 못할 수도 있음.

0 to 65535 for CV\_16U images

0 to 1 for CV\_32F images. -> 추천.

dst: 입력 영상과 같은 크기와 같은 깊이의 출력 영상.

code: 색상 변환 코드 (see ColorConversionCodes).

dstCn: 출력 영상의 채널 수. 0이면 소스와 변환코드에 따라 자동적으로 설정

RGB ↔ HSV 변환을 위한 color codes

1. cv::COLOR\_BGR2HSV 2. cv::COLOR\_RGB2HSV

3. cv::COLOR\_HSV2BGR 4. cv::COLOR\_HSV2RGB

위 함수 사용 시의 주의사항

1. 입력 RGB 데이터가 0~1의 부동소수이면 HSV 결과도 부동소수.

부동소수는 정규화된 입력값을 사용하기로 사정되어있음

HSV = cv.cvtColor(img/255, cv.COLOR\_BGR2HSV) # 부동소수 입력

S와 V는 모두 0~1의 범위. H 는 0~360ﾟ 값의 범위를 가짐

H를 회전하여 각도를 더한 것이 360ﾟ를 넘으면 OpenCV에서 자동으로 360ﾟ를 뺀 값으로 절환하는 기능이 작동함

1. 입력 RGB 데이터가 uint8인 경우 변환 과정을 거치면서 HSV 결과도 uint8로 됨
2. RGB2HSV, BGR2HSV 등 \_FULL 이 붙지 않은 변환의 경우 H는 0~179ﾟ로 스케일링됨

이때 H를 회전하여 각도를 더한 값이 180ﾟ를 넘으면 수동으로 180을 뺀 값을 사용해서 연산해야함

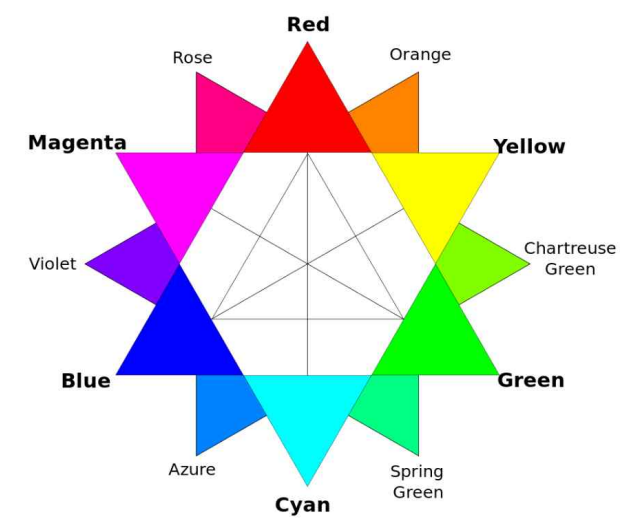
1. 아래와 같이 \_FULL이 붙은 변환에서는 H는 0~255의 범위를 가짐

cv::COLOR\_BGR2HSV\_FULL = 66, cv::COLOR\_RGB2HSV\_FULL = 67, cv::COLOR\_BGR2HLS\_FULL = 68, cv::COLOR\_RGB2HLS\_FULL = 69, cv::COLOR\_HSV2BGR\_FULL = 70, cv::COLOR\_HSV2RGB\_FULL = 71, cv::COLOR\_HLS2BGR\_FULL = 72, cv::COLOR\_HLS2RGB\_FULL = 73

H를 회전하여 각도를 더한 값이 255를 넘어도 uint8 변수의 특성상 상위 비트 데이터는 사라지고 하위 8비트만 남기 때문에. 특별한 처리가 필요 없는 장점이 있음

**HSV 좌표계를 이용한 영상의 색조 변화**

**Hue에만 변화를 주어 영상의 색상 변화를 함**

RGB Color Wheel

# 코드 관측 결과

Shift Value = int((hueshift/360) \* max) # max : 180 이거나 255 => uint8일 경우만 해당

역방향의 경우 Hue Shift : 360 – x 만큼 하면 원하는 결과를 얻음

정방향의 경우 Hue Shift : x 만큼 하면 원하는 결과를 얻음

부동소수형에 대해서는 Hue의 자동 Clipping이 진행됨

uint8 타입에 대해서는 아래와 같은 수동 절환 작업이 필요함(FULL의 경우 필요없음)

180 이상 되는 값에 대해서는 180을 감산함. 360이 넘게 회전하면 0부터 다시 시작한다고 보면 됨

총 3개의 실습

uint8 0~179 테스트, 0~360(FULL) 테스트, 부동소수 테스트

**HSV 좌표계를 이용한 영상의 색상(채도)/명도 강조**

색상(채도) S(Saturation)를 강조하면 색상이 진해지는 효과를 얻음

V(Value)를 강조하면 명도가 밝아지는 효과를 얻음

**HSV HSI HSL 좌표계에서의 채도의 표현**

명도(휘도)가 중간(0.5)에 있을 때 가장 강한 색채를 띰

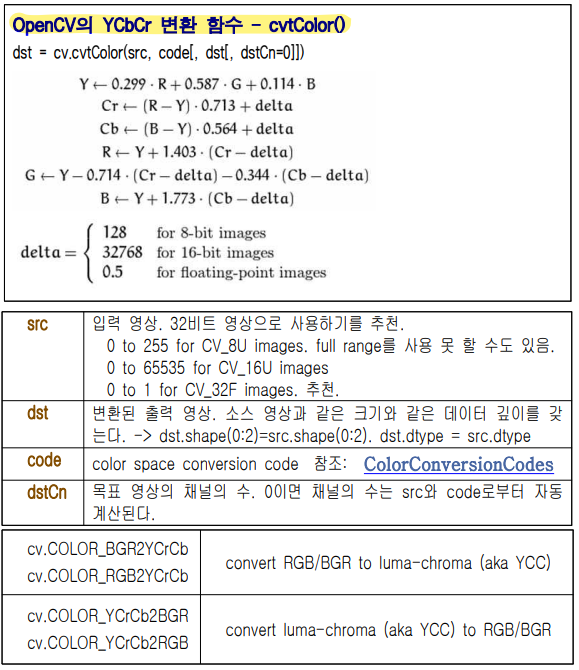
HSV 좌표계의 경우 명도가 1일 때 가장 강한 색상을 가짐

**색차 기반의 좌표계 – YCbCr**

Cb = R – 휘도 Cr = B – 휘도

색차 : 색상 정보에서 휘도를 뺀 성분 Cr, Cb를 말함

OpenCV 의 cvtColor 함수를 통해 변환이 가능함



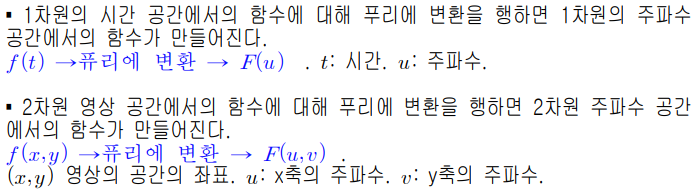
주의점

연산 과정에서 충분한 길이의 비트를 사용하지 못해 컬러 해상도가 떨어지는 단점이 존재함

11주차 완료

**푸리에 변환(Fourier Transform)** – 연산식은 거의 다 스킵하셨음.

시간에 따라 변화하는 함수를 주파수에 따라 변화하는 함수로 변환하는 작업을 수행함



**장점**

공간 영역에서의 합성곱은 주파수 영역에서는 단순 곱셈으로 처리가 가능하여 빠른 속도로 필터링한 결과를 얻을 수 있음

영상에 푸리에 변환 적용 시 영상에 들어 있는 주파수 성분 분석이 가능함

변환된 값에 특정 성분을 제거하여 역변환을 하면 잡음이 제거된 신호로 복구가 가능함

공간 영역의 필터 커널에 푸리에 변환 적용 시 그 필터의 주파수 통과 특성 분식이 가능함

**푸리에 급수(Fourier Series)**

모든 주기함수는 다양한 주파수와 크기를 가진 sin 과 cos 함수의 합으로 표시할 수 있다.

주기가 T인 임의의 주기함수 f(t)를 기본주기가 f0 = 1/T인 고조파들의 합으로 표현할 수 있다는 것

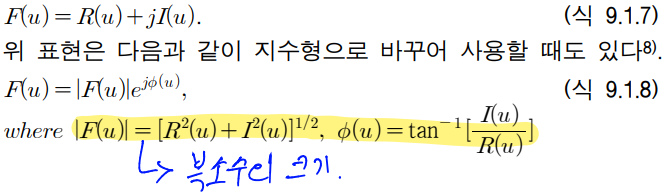
* 푸리에 변환은 주기가 무한대인 함수의 푸리에 계수를 구하는 과정이라고 볼 수 있음

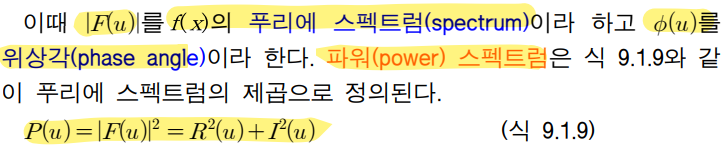
푸리에 정변환 : 푸리에 계수를 구하는 문제

푸리에 역변환 : 원함수를 구하는 문제

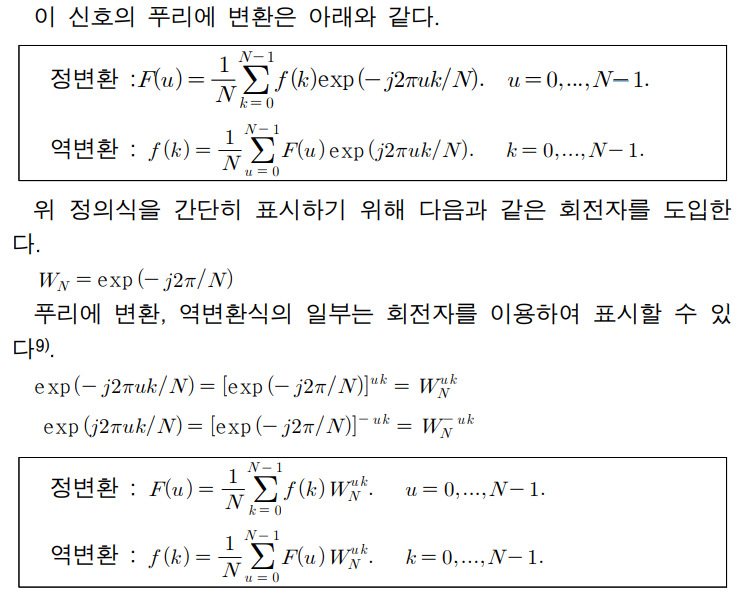
푸리에 변환의 결과는 복소수가 됨

연속 신호에 대한 푸리에 변환





**DFT(Discrete Fourier Transform) : 이산신호에 대한 푸리에 변환**



**FFT(Fast Fourier Trasnform) : 푸리에 변환의 고속화**

기존의 계산 결과를 그대로 활용할 수 있으며, 따라서 계산량을 줄일 수 있다.

1차원 신호에 대한 적용 실험 -> 1차원 신호 : 영상이 아닌 음성 신호임

푸리에 변환은 상대적 크기가 중요하기 때문에 루트(root, 제곱근)을 씌우지 않아도 상관없다.

푸리에 변환 함수의 특징

실수 입력에 대한 변환 결과

실수부 : 우 대칭(y축 대칭) 허수부 : 기 대칭(원점 대칭)

푸리에 스펙트럼, 파워 스펙트럼 : 우 대칭 위상 : 기 대칭

코사인 함수의 푸리에 변환 값 : 실수부에서 큰 값의 계수가 관찰됨

사인 함수의 푸리에 변환 값 : 허수부에서 큰 값의 계수가 관찰됨

코사인 함수와 사인 함수의 푸리에 변환 값 스펙트럼은 똑같음

함수에 직류 성분이 들어있을 경우 그 성분이 주파수 0에 교류 성분의 2배의 크기로 나타남

ex) f1 = 50; y = sin(2\*pi\*f1\*t) + 1

2개의 주파수 성분을 갖는 sin 신호의 경우 해당 주파수에 큰 에너지가 있음을 보여줌

ex) f1 =50, f2 = 100, y = 2\* sin(2\*pi\*f1\*t) + sin(2\*pi\*f2\*t)

사인 함수와 코사인 함수의 주파수가 다를 경우 스펙트럼은 같으나, 실수부에는 코사인 성분이, 허수부에는 사인 성분이 나타남. 두 구성함수의 진폭의 비율이 주파수에도 반영됨

ex) f1=50; f2=100; y = 2\* sin(2\*pi\*f1\*t) + cos(2\*pi\*f2\*t);

**푸리에 변환의 단점**

시간에 따라 주파수가 변하는 신호에 대해서는 그 시간별 주파수의 변화를 관찰할 수 없음

**극복을 위한 방안**

Short Time Fourier Transform 사용 – 시간을 나누어 각 구간 별로 푸리에 변환을 행함

Wavelet Transform – 더 원천적인 해결법

헤르미트 대칭(Hermitian Symmetry)

진폭은 우 대칭의 특징을 같고 위상은 기 대칭의 특징을 갖는 것. 공액 대칭이라함

공액 복소수 : 임의의 복소수에 대해 실수부는 같고 허수부는 부호가 반대인 수

ex) a + jb에 대한 공액 복소수 = a - jb

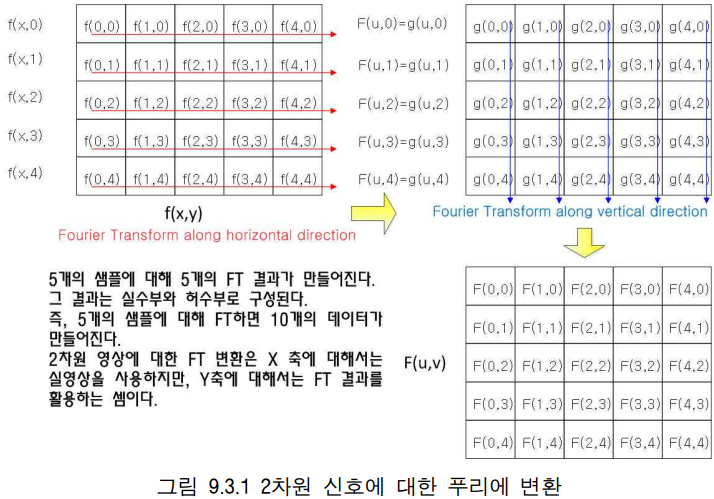
시간 영역 실수 함수의 푸리에 변환 : 복소수 함수가 됨 -> 반드시 공액 대칭을 이룸

=> 실수부는 같고 허수부의 부호가 반대인 것

헤르미트 대칭성을 갖는 함수의 푸리에 역변환 : 실수 함수

ㄴ 보통 허수부는 수직축에 도시하므로, 헤르미트 대칭의 경우 실수부를 고려하지 않으면 기 대칭과 같은 개념임

**2차원 신호, 영상에 대한 적용**



**2D FFT 변환 코드**

**FFT 정방향 변환 함수**

Y=fft2(double(img)); 변환 결과, Y=M\*N, 복소수

numpy: Y=np.fft.fft2(img), complex123형 MxN 복소수

OpenCV: Y=cv.dft(np.float32(img), flags = cv.DFT\_COMPLEX\_OUTPUT). MxNx2채널 복소수

**FFT 역방향 변환 함수**

img2 = ifft2(Y); 복원된 영상

numpy: img=np.fft.ifft2(Y). MxN 복소수

OpenCV: img=cv.idft(Y). MxNx2 복소수

**중심점에 저주파가 오도록 푸리에 데이터를 재배치**

Y1 = fft2shift(Y);

numpy: Y1 = np.fft.fftshift(Y),

원상복귀 시킬 때는 Y\_new = np.fft.ifftshift(Y1) OpenCV: fftshift() 유사 함수가 없는 듯

**복소 푸리에 값의 크기 변환 -> magnitude spectrum으로 변환**

Y2 = abs(Y1)=sqrt(real(Y1).^2 + image(Y1).^2) = sqrt(Y1 .\* conj(Y1))

numpy: np.real(Y1)과 np.imag(Y1) 함수로 실수부와 허수부를 추출. 직접 계산해도 되고

Y2=np.abs(Y1) 함수를 활용 가능.

OpenCV: 변환 결과 Y, Y1의 채널 0는 실수부, 채널 1은 허수로 구성된다.

Y2 = cv.magnitude(Y1[:, :, 0], Y1[:, :, 1])

스펙트럼의 화면 출력 : 크기의 차이가 커서 log scaling이 필요

Spectrum = log(1 + Y2);

스케일링된 결과의 화면 출력 J=imshow(Spectrum, []);

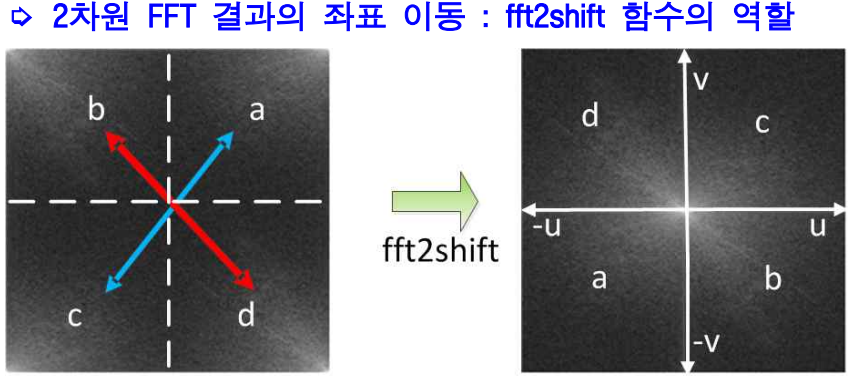
혹은 최대, 최소를 찾아 정규화여 출력할 수도 있다.

MaxS = max(Spectrum(:));

MinS = min(Spectrum(:));

Spectrum = uint8( Spectrum\*255 / (MaxS-MinS) );

figure, imshow(Spectrum);



fft2shift 함수 : 중심점에 주파수가 0인 점이 오도록 화살표와 같이 1사분면의 데이터를 대각선 방향으로 이동시킴

**u 와 v** : 각각 x, y 방향의 주파수 변수. 끝 방향으로 갈수록 주파수가 증가하며, 원점은 주파수가 0인 직류 성분

밝게 표시된 것 : 그 값이 큰 것을 의미함

어둡게 표시된 것 : 그 값이 작음을 의미함

1차원의 경우 스펙트럼은 y축에 대하여 우 대칭을 이루었으나, 2차원에서는 x 축에 대해서도 대칭을 이룸

y축 상의 푸리에 변환에 의해 스펙트럼이 우 대칭을 이루었기 때문

결과적으로 2차원 신호에 대한 푸리에 스펙트럼은 기대칭인 것처럼 x, y축에 대해 1회 대칭이 이루어짐

FT2\_1\_imageFT\_numpy.py

*# Fourier Transform(푸리에 변환)  
 . 시간 도메인(X축)에서 표현된 신호(일반적인 파형 도표)를 주파수 도메인으로 변환.  
 . 시간축이 제거되어 대상의 전체적인 특징을 파악할 수 있음.  
 . 이미지에 적용이 되어 중심이 저주파 영역, 주변이 고주파 영역을 나타냄.  
 . 푸리에 변환을 하여 저주파 또는 고주파를 제거하고 다시 역으로 이미지로 변환 함으로써  
 이미지가공을 할 수 있음.  
 (ex; 푸리에 변환 후 중심의 저주파를 제거하고 다시 Image로 전환 하면 이미지의 경계선만 남게 됨.  
 푸리에 변환 후 주변의 고주파를 제거하면 모아레 패턴(휴대폰으로 모니터를 찍었을 때 나타나는 현상)  
 을 제거할 수 있음.(모니터의 고주파를 제거함.)  
 )*

*# 푸리에 변환은 영상의 위치에 크게 영향 받지 않음*

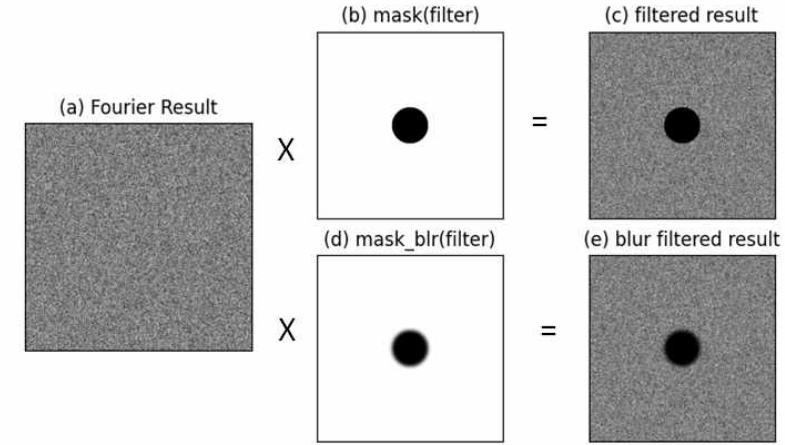
가로 방향으로 존재하는 주파수 성분은 u축에 나타남(세로축)

세로 방향으로 존재하는 주파수 성분은 v 축에 나타남(가로축)

영상에 존재하는 성분의 수직 방향으로 강한 주파수 성분이 나타남

**주파수 공간에서의 필터링**

푸리에 변환 결과에 마스크 영상을 곱해주는 동작으로 영상의 특정 주파수 성분을 제거할 수 있음



중앙의 작은 원만 0이고 나머지는 모두 1인 mask 영상 => (b)

(c)를 푸리에 역변환하면 특정성분 주파수가 제거된 원본 영상을 얻음

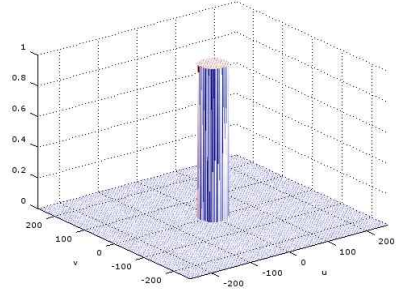
def get\_filter\_mask(mask\_size, d=20, option='circle', sigma=5):  
 rows, cols = mask\_size  
 mask = np.ones((rows, cols), dtype=float)  
  
 crow = rows // 2 # center of rows  
 ccol = cols // 2 # center of cols  
  
 # 1) 정사각형 마스킹: 중앙부 fft 값을 0에 가까운 값으로 만든다.  
 if option == 'square':  
 mask[crow-d:crow+d+1, ccol-d:ccol+d+1] = 0  
  
 # 2) 원형 마스킹: 중앙부를 중심으로 반경 d인 원의 영역 내부를 0에 가까운 값으로 만든다.  
 if option == 'circle':  
 for x in range(-d, d+1):  
 for y in range(-d, d+1):  
 if x\*\*2 + y \*\*2 < d\*\*2:  
 mask[crow+y, ccol+x] = 0.00000001  
  
 if sigma > 0:  
 mask\_blr = cv.GaussianBlur(mask, (21, 21), 5)  
  
 return mask\_blr, mask

주파수 공간에서 제거할 마스크를 얻는 함수

**주파수 공간상의 필터 정의**

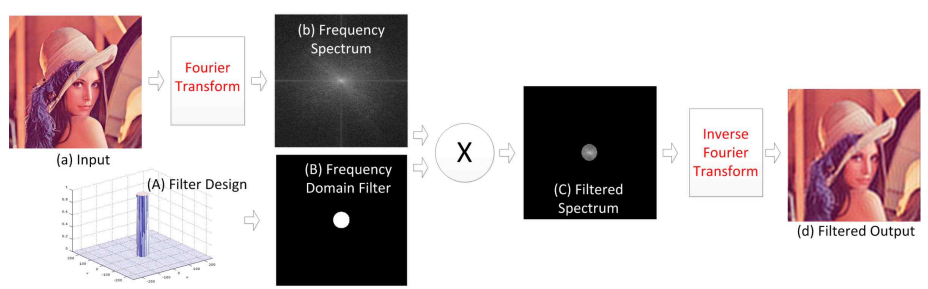
주파수 공간상의 필터 : 영상의 주파수 스펙트럼에 대해 적용할 통과할 주파수에 대한 이득(gain)을 결정하는 문제로 볼 수 있음

ㄴ 주파수의 통과 여부를 결정함



중앙의 기둥 : 중앙 부분의 DC(직류성분)을 포함한 저주파 성분의 통과 이득

즉 1의 값으로 해당 주파수 성분은 통과. 나머지 평면부분의 주파수들은 0의 이득으로 통과시킴



b x B => C C 역변환 => d

**실제 프로그래밍 시의 고려사항**

1) 그림 (b)의 스펙트럼은 f=np.fft.fft2(img) 연산에 의해 구한 푸리에 변환 결과를 가시성을 높이기 위해 log scaling하고, 중심 영역에 저주파가 오도록 배치해 보인 것을 사용한 것이다. 그러나, 실제로 필터링할 때는 log 스케일링 한 것을 사용하면 안된다.

2) (C)의 filtered Spectrum은 중앙에 직류성분(DC)이 위치하도록 np.fft.fftshift() 함수로 이동시킨 것이다. 따라서 np.fft.ifftshift() 함수를 통해 원상 복귀시킨 후 역변환해야 한다.

FT3\_frequency\_domain\_filtering\_1\_numpy.py – 저주파 성분을 제거하고 고주파 성분을 통과시킴

fftshift() : 주파수가 0인 부분(DC 성분)을 중앙(원점)에 위치하게 하여 분석을 용이하게 하기 위함

ifftshift() : 영상으로 역변환 시 사용되는 부분으로, 대각 대칭 이동된 데이터를 본래 위치로 복귀 시킬 때 사용

마스크 처리된 푸리에 변환 결과를 다시 자신의 위치로 재배치하는 것

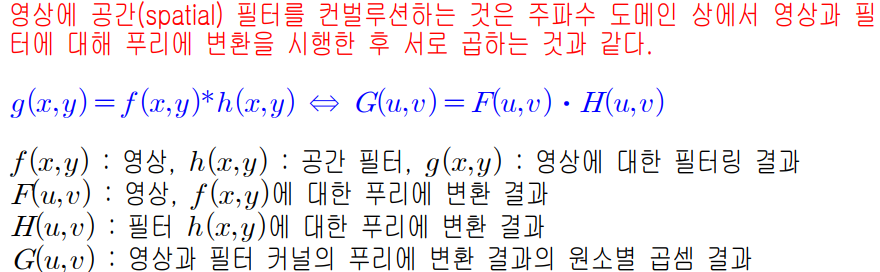
고주파가 제거된 영상에 물결무늬 같은 ringing effect가 관촬되는 이유

주파수의 통과 특성을 정의하는 마스크 정의 시, 1과 0으로만 값을 구성하였기 때문에 지나치게 날카로운 cut off 동작을 요구하게 됨. 따라서 새로운 alias 잡음이 유기되었음

블러링된 마스크 사용 시 ringing effect 가 대폭 감소된 필터링 결과를 관측 가능함.

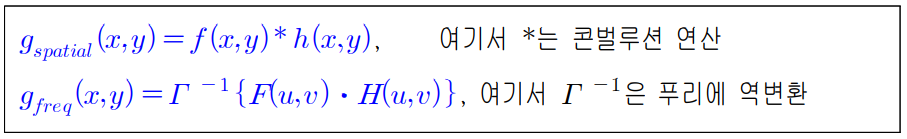
**공간 필터의 주파수 응답**

쌍대성 원칙(duality principle) : 푸리에 변환의 특성 중 제일 중요함



\* 은 컨벌루션 연산을 의미하며, 등가성의 의미로 ⬄ 기호가 사용되었음

-> 공간 상에서 컨벌루션 연산에 의한 필터링 처리 결과는 영상과 커널 각각에 대해 푸리에 변환한 결과를 곱해서 역변환한 것과 같다는 것.



공간상의 필터링 : 코릴레이션으로 수행됨

1. 필터 커널의 계수가 원점 대칭일 경우 콘벌루션과 연산 결과가 동일함

ex. 가우시안, 라플라시안, 평균 커널의 경우, 코릴레이션과 콘벌루션 결과가 서로 같음

위 세 커널은 대칭되는 계수의 값이 같기 때문

즉, 

2. 대칭 관계의 계수와 크기가 같고, 그 부호가 서로 반대인 경우 코릴레이션과 컨벌루션 결과는

서로 부호가 반대임

ex. 소벨 필터의 경우 원점을 중심으로 대칭되는 계수의 크기가 같고 부호가 반대임

즉, 

**영상처리에 사용되는 커널은 모두 원점 대칭이거나 부호만 바뀐 있는 경우가 있음**

**따라서 코릴레이션과 콘벌루션을 구별하지 않고 사용하는 경우가 많음**

공간 차원(spartial domain)의 필터를 주파수 평면에서 필터로 바꾸면 필터링 작업을 푸리에 변환을 이용해

수행 가능함

-> 공간 필터링을 푸리에 변환을 통해 주파수 평면 상에서 실시할 수 있다는 것

주어진 공간 필터를 통해 주파수 도메인에서 필터링을 행하는 방법

1) 소벨 필터(h)를 푸리에 변환한다. H=fft2(h);

2) 영상을 푸리에 변환한다. F = fft2(f);

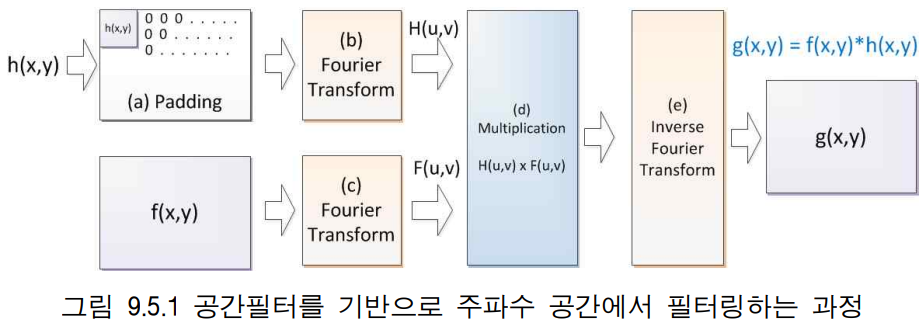
3) 두 변환 결과를 곱한다. ↔ 이것은 공간차원에서 콘벌루션한 것과 같다. G = H · F;

4) 곱한 결과를 푸리에 역변환하여 필터링 결과를 구한다. g=ifft2(G);

\* 주의점

▪ 필터와 영상의 크기가 같아야 곱할 수 있으므로 필터 h에 패딩이 필요하다.

▪ 실제 공간 차원에서는 코릴레이션으로 필터링을 수행하므로 콘벌루션과 코릴레 이션의 결과가 같은 공간 필터에 대해서만 이러한 연산이 유효하다.



(a) 공간 차원의 필터에 영상과 같은 크기로 패딩(padding)을 실시함

패딩하는 값으로는 0을 사용. 주파수 통과 성분 결정에 영향 X

(b) (c) 같은 크기의 어레이 데이터를 각각 푸리에 변환함

(d) 각 변환 값을 원소 단위로 곱셈 연산함

(e) 역변환 실시. 허수부는 버리고 실수부를 취해 필터링 된 결과를 얻음

Zero Padding

필터 커널은 영상에 비해 크기가 작기 때문에, 그 크기를 맞춘 다음, 늘어난 나머지 부분은 0을 채워 넣는 것

공간 필터의 주파수 응답

주파수 응답 : 원래 시스템의 입력되는 신호의 주파수의 변화에 따른 출력 주파수의 변화를 측정하는 수단으로 활용되는 지표

영상처리에 사용되는 코릴레이션 필터라면 공간 필터의 주파수 응답 특성은 해당 커널이 어떤 주파수대의 신호를 통과 시키고, 차단하는지를 알아볼 수 있음

-> 주파수 응답은 주파수 변화에 따른 게인(gain) 특성이라고 말할 수 있음

공간산의 코릴레이션 연산이 주파수 평면상에서는 단순한 곱셈연산으로 대치됨

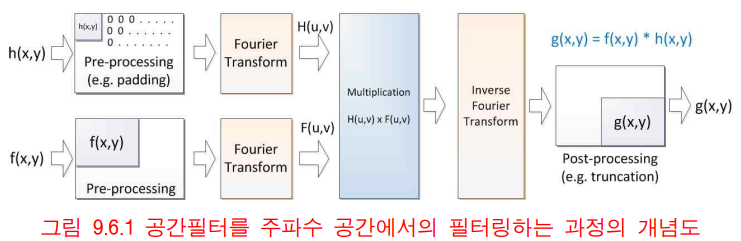
따라서 공간 필터에 대해서 푸리에 변환을 시행할 경우 그 필터의 주파수 응답을 알아볼 수 있음

= 강조된 성분은 큰 값. 감쇄된 성분은 작은 값을 갖게 될 것임.

공간 필터의 주파수 응답 사례 : PDF 50 ~ 53 정독

공간필터의 주파수 공간상에서의 필터링

종합적 고찰



필터 h(x, y) 와 영상 f(x, y)에 대해 푸리에 변환한 값을 서로 곱한 후 이를 역변환하여 필터링을 수행함

필터 h의 크기 : 영상의 크기와 같도록 zero padding 을 해야함

필터링 연산에 사용되는 공간 필터는 코릴레이션 커널이므로 콘볼루션 커널로 변환해야함

만약 코릴레이션 커널이 원점 대칭일 경우 콘볼루션 커널과 같음

원점 대칭 관계에 있는 커널의 부호만 다르고 크기가 같다면, 연산 결과의 부호만 반대로 바꾸면 됨

padding & truncation

주파수 공간 상에서 정의된 필터 함수와 영상의 푸리에 변환과의 곱셈으로 공간 차원에서의 코릴레이션에 의한 필터링을 수행 시 출력되는 결과는 패딩 영상을 포함함.

따라서 이를 제거하는 truncation 작업이 필요함

12주차 푸리에 변환 정리 완료