영상 신호처리 이론 및 실습 정리

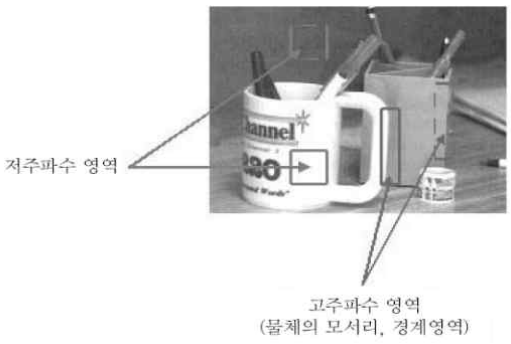
1. 영상 영역처리

영상신호 영역처리는 출력화소를 만들기 위해서 입력 화소 뿐만 아니라 입력 화소 주위에 있는 화소들을 사용하여 컨볼루션을 통해 이루어짐.

- 공간 주파수(spatial frequency)

이미지는 크게 두 가지 영역으로 표현할 수 있는데, 공간 영역과 주파수 영역이 있다. 공간 영역은 픽셀 값으로 이미지를 표현한다. 반면, 주파수 영역에서는 화소 값이 직접 표현되지 않고 변동계수(coeffcient)로 표현된다. 대표적인 변환 방법에는 이산 코사인 변환(DCT), 이산 푸리에 변환(DFT)가 사용된다.

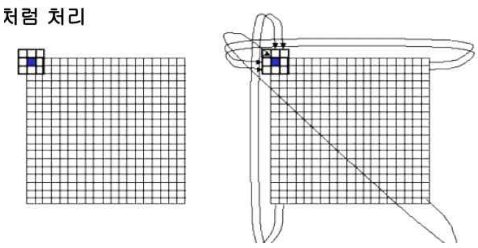
주파수는 이벤트가 주기적으로 재발생하는 빈도인데, 이미지에서도 주파수가 존재한다. 즉, 공간 영역 상 화소 밝기의 변화율을 나타내는 것이 공간 주파수이다. 공간 주파수에서도 밝기가 얼마나 빨리 변화하는가에 따라 고주파, 저주파 영역으로 분류할 수 있는데, 고주파 영역은 객체의 모서리 부분, 저주파 영역은 배경 부분이나 객체의 내부이다. 고주파, 저주파 성분을 알면 블러링과 에지 영상을 만들 수 있다.




- 컨볼루션 처리

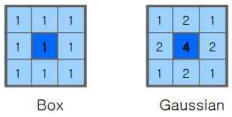
행렬로 표현되는 가중치를 컨볼루션 마스크, 컨볼루션 커널이라 한다. 가중 행렬의 크기는 항상 그 중심을 결정할 수 있도록 odd로 사용한다.

이 때, 컨볼루션 창의 중심이 (0,0)이 되는 영상 영역 밖의 처리를 할 때는 zero-padding, Warp-around convolution을 사용한다. Zero-padding은 컨볼루션 창의 비어있는 셀은 0으로 처리하는 것이고, Warp-around는 영상의 모서리가 서로 연결되어 있는 것처럼 처리하는 것이다.

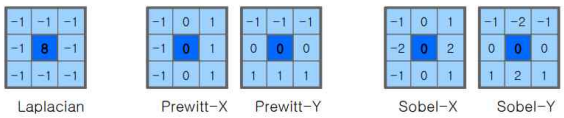



- 일반적인 마스크

평활화 (Smoothing; Low-Pass Filtering)




밝기값 차이의 강조 (Edge Detection; High-Pass Filtering)




- 저역통과필터(LPF)

공간 저역통과 필터는 영상의 저주파 성분을 그대로 통과, 고주파 성분은 감쇄되는 효과를 가진 필터이다. 주변의 8개의 이웃 화소의 밝기값과 자기 자신의 밝기값의 평균 연산하여 사용한다. 고주파 성분이 많은 경우, 급격한 밝기 변화가 완만한 변화로 감쇄되는 현상을 가질 수 있다. 이를 blurring 처리라고 한다.

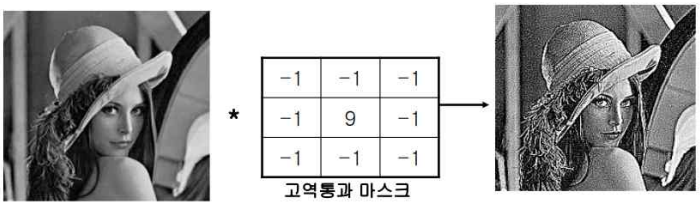

(a) 원영상


(b) 저역통과 필터링 결과

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

- 고역통과필터(HPF)

저주파 성분은 감쇄시키는 반면, 고주파 성분은 통과시키는 특징인 필터이다. 에지와 같은 고주파 성분이 있는 부분을 더욱 강조하는 샤프닝 효과가 있다. spot성 노이즈가 매우 부각되어 나타나고, 흐릿한 영상을 샤프닝 처리하는 데 사용한다. -




- 잡음 제거 알고리즘

y(t) = s(t) + n(t)

실제 신호 + 잡음 신호

영상에서 잡음의 종류는 광전자에 의한 노이즈, 임펄스성 노이즈(salt and pepper) = 가우시안 노이즈, 임펄스 노이즈, 구조적 노이즈 = 압축 과정에 의한 노이즈

- 잡음제거 알고리즘

(1) 평균 필터(Average or Mean Filter)

자기 자신의 화소와 주변 8개의 화소값들을 1/9로 나누어 계산.

```
void CRegionDlg::OnAVG(){
        int mask[3][3] = {{1,1,1,}, {1,1,1}, {1,1,1}}; // 3x3을 1/9를 곱해준다.
        int temp;
        for(int i=0;i<255;i++){
        for(int j=0;j<255;j++){
        temp = 0;
        for(int r=0;r<3;r++){ // 3x3 convolution mask
        for(int c=0;c<3;c++){
        temp +=
        (mask[r][c]*ORG[i+r-1][j+c-1]); //입력영상에 mask(1) 곱해준 후 더해준다.
         }
         RESULT[i][j] = (unsigned
        char)(temp/9); // 9로 나눈 값으로 화소 값 지정
         }
        }
        for(i=0;i<256;i++){
        RESULT[0][i]=ORG[0][i];
        RESULT[i][0]=ORG[i][0];
        RESULT[255][i]=ORG[255][i];
        RESULT[i][255]=ORG[i][255];
        }
        DrawResultImage();
}
```

(2) 메디안 필터(Median Filter)

자기 자신을 포함한 주위의 화소값들에서 중간값에 해당하는 값을 선택하는 알고리즘

```
void CRegionDlg::OnMedian(){
        const int winsize = 3; // 행렬 사이즈 지정
        unsigned char* arr = new unsigned
        char[winsize * winsize];
        for(int y=0 + winsize/2; y <256-winsize/2;y++){
        for(int x=0 + winsize/2; x<256 - winsize/2;x++){
        int total = 0;
        for(int by = 0;by<winsize;by++){
        for(int bx=0;bx<winsize;bx++){
        arr[total] = ORG[y+(by-winsize/2)][x+(bx-winsize/2)];
        for(int i=total;i>0;i--){
```

```
            if(arr[i] < arr[i-1]){ // 중간값 찾기
            unsigned char temp = arr[i];
            arr[i] = arr[i-1];
            arr[i-1] = temp;
            }
            }
            total++;
            }
            }
            RESULT[y][x] = arr[winsize * winsize/2];
             }
             }
            delete []arr;
            DrawResultImage();
}
```

(3) 가우시안 스무딩 필터(Gaussian Smooting Filter)
가우시안 스무딩 – 공학분야에서 가장 보편적인 확률분포함수
가우시안 스무딩 필터란, 가우시안 분포를 영상처리에 적용한 것으로 가우시안 잡음을 포함한 영상의 잡음을 제거하기 위한 필터이다.

2. 영상의 기하학적 변환
선형 – 평행이동, 회전, 크기변환(scaling)
비선형 – 워핑(warping), 모핑(morphing)

- 기하학적 처리
: 임의의 기하학적 변환에 의해 화소들의 배치를 변경하는 처리
영상 내의 화소들을 움직이거나 화소들을 생성 (보간법)

- 기하학적 변형 후 영상 만드는 방법
: 순방향 사상 – 원래의 이미지에서 목적 이미지로 구현
x,y 사상함수에 의해 입력화소가 출력화소로 이동하는 것이며, 문제점으로는 홀과 오버랩이 있다. 홀은 정의되지 않은 화소드로 목적화소에 대응하는 원시화소가 존재하지 않는 것이고, 오버랩은 두 개의 입력 화소가 하나의 출력 화소에 대응하는 것이다.
역방향 사상 – 목적 이미지에서 원래 이미지로의 구현, 실제 구현에서 많이 사용.
목적 이미지의 각 위치의 화소 값을 결정할 때, 목적 이미지로 매핑되는 원 이미지상의 좌표를 계산하여 해당 밝기값 결정. 홀과 중첩의 문제를 제거할 수 있다.

- 보간법(Interpolation)
: 어떤 점(x, y)가 확대 축소되어 (X, Y)로 위치가 변하면 두 좌표 사이에는 X=ax, Y=by의 관계가 성립된다. a, b가 1보다 큰 값을 가지면 확대, 작은 값을 가지면 축소이다.
보간법은 공간이 발생한 화소 값을 주변 화소값을 사용하여 얻는 방법으로, 주변의 화소들을 분석하여 새로운 화소를 생성한다.

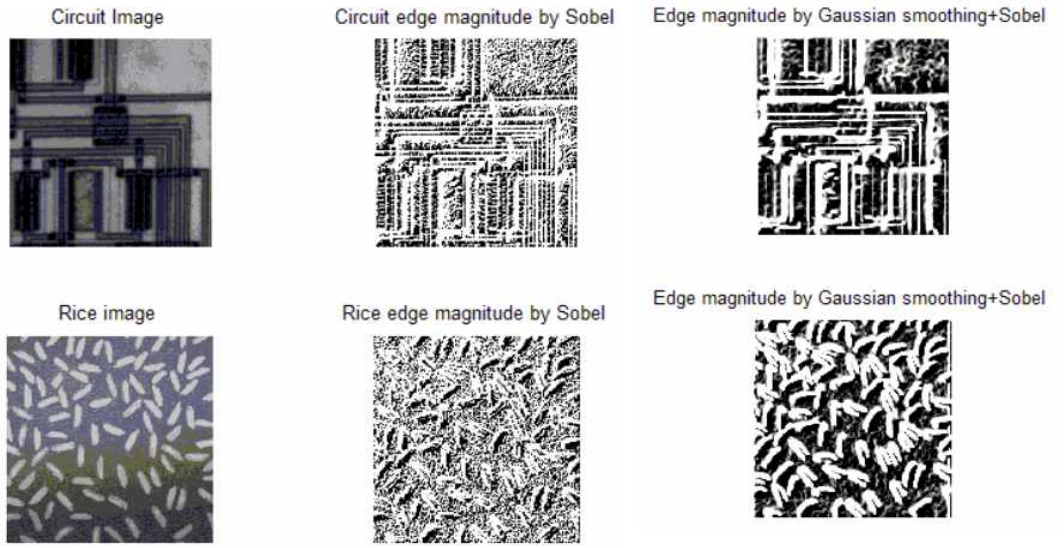
(1) 인접 화소 보간법
: 생성된 주소에 가장 가까운 원시 화소를 출력화소로 할당하는 것, 원시화소에 대해 계산된 분수 주소는 가장 가까운 유효한 화소 주소로 반올림한다. 장점은 처리 속도가 빠르다는 것이고, 단점은 입력 화소에 대응하는 출력 화소들의 수가 클수록 영상의 질이 저하된다는 점이 있다.

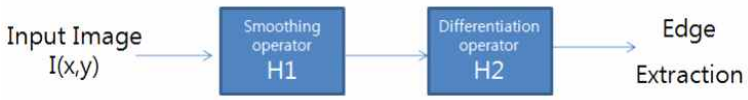
(2) 양선형 보간법
: 새로운 화소를 생성하기 위해 네 개의 가장 가까운 화소들에 가중치를 곱한 값들의 합을 사용. 각각의 가중치는 화소로부터 거리에 반비례. 화소당 3개의 일차 보간법으로 많은 계산량을 요구하지만 스무딩한 영상을 산출한다.

3. 에지 검출 알고리즘
- 에지 검출을 위한 미분 연산
영상의 밝기와 색에 급격한 변화가 있는 경우, 에지가 존재
1차 미분 : 좌표(x, y)의 경우 농담 분포를 나타낸 1차 미분값은 크기와 방향을 가진 벡터량으로 표현된다.

에지 검출 필터
- Prewitt 마스크
- Sobel 마스크

에지 검출에서 미분연산의 문제점
: 노이즈가 있는 1차원 스텝 신호를 미분하면 원본 영상보다 노이즈가 더 심해지는 문제점이 발생
이 문제점을 개선하기 위해 가우시안 스무딩 처리 후, 미분 연산을 수행한다.






(a) 에지검출을 위한 시스템 블록도

에지 검출을 위한 시스템 블록도의 문제점

: 일반적으로 영상에 대한 영역처리는 많은 계산량, 많은 시간을 필요로 하는데, 한 영상에서 두 번의 마스크 처리는 더 많은 시간, 계산량을 필요로 함. 이를 감소시킬 수 있는 방안으로 가우시안 스무딩을 미리 미분하여 그 함수로 연산을 하는 방안을 사용하는 것도 있음. (1차원 가우시안 함수를 미분, 2차원 가우시안 함수를 미분)

라플라시안 에지 검출기

에지 강도 결과에서는 다양한 임계값을 이용해 여러 에지 결과를 추출할 수 있다.

캐니에지 검출 알고리즘

영상처리에서 에지검출의 목적은 이후의 추가적인 영상처리를 위해 사용되는 여러 구조적인 성질을 그대로 유지하면서 많은 영상데이터의 양을 현저하게 줄이는 것이다. 캐니에지 검출기는 스무딩된 영상에서 가장 큰 에지강도에 따라서 에지를 추출하는 알고리즘이다.

캐니에지 검출 알고리즘 단계

(1) 노이즈 제거를 위해 영상을 스무딩 처리. 일반적으로 가우시안 필터링으로 스무딩.

(2) 에지 그레디언트를 계산한다. 에지 그레디언트는 에지 강도, 에지 방향을 포함 (sobel or Prewitt)

(3) 에지 그레디언트 화소 중 최대가 아닌 화소를 제거 (Non-maximum Suppression)

(4) 2-단 임계값 처리를 수행하여 임시 이진화

(5) 주변화소와의 연결성을 이용한 최종 에지화소 검출

- HPF 1

```
void CRegionDlg::Onhpf1(){ //HPF-1
int I,j,x,y,sum;
int hr[3][3] = {{-1,-1,-1,}, {-1,9,-1}, {-1,-1,-1}};
int hc[3][3] = {{-1,-1,-1,}, {-1,9,-1}, {-1,-1,-1}};
int temp;
for(int I=0;i<255;i++){
for(int j=0;j<255;j++){
sum = 0;
for(int y=0;y<3;y++){
for(int x=0;x<3;x++){
sum +=
ORG[i-1+y][j-1+x]*hr[y][x];
}
```

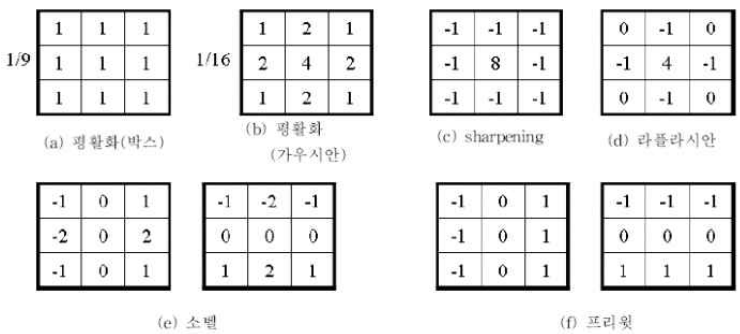
| | |
|---|---|
| if(sum>255) sum = 255; else if(sum<0) sum = 0;<br>for(y=0;y<3;y++)<br>for(x=0;x<3;x++)<br>sum += ORG[i-1+y][j-1+x]*hr[y][x];<br>if(sum>255) sum = 255; else if(sum<0) sum = 0;<br>RESULT[i][j] = sum;<br>DrawResultImage();<br>} | |

4. 주파수 분석 알고리즘
영상 함수 f(x,y) : 공간좌표(x,y)에서 f의 값은 그 점에서 영상의 밝기 혹은 빛의 세기
좌표계의 원점이 영상의 왼쪽 위에 있음

표본화와 양자화 과정을 거쳐 수치 데이터, 또는 디지털 영상으로 바뀌며 이 과정에 사용되는 장치가 아날로그 디지털 변환기(ADC), 특히 디지털 영상으로 바꾸는 데 사용하는 ADC를 프레임 그래버라고 함

일반적으로 영상의 공간영역 해석은 컨볼루션 연산을 통해 수행된다. 출력 화소의 값은 이웃 화소들의 가중치의 합으로 계산, 가중치 행렬은 2차원 임펄스 응답으로 생각할 수 있다. 가중치 행렬은 마스크라는 용어를 사용한다.
여기서 마스크란, 영상 안에서 일정 부분에 위치시키기 위한 일종의 행렬 모양의 구조체이다. 주로 3x3, 5x5 등 정방 행렬을 사용한다. 마스크를 이용한 컨볼루션은 원 영상에 마스크를 겹쳐서 왼쪽부터 오른쪽으로 위에서 아래 방향으로 진행하면서 계산된 값을 (0,0) 자리에 다시 할당함으로써 출력 영상을 얻음.
컨볼루션 마스크의 각 원소의 합은 결과 영상, 혹은 출력 영상의 전체 밝기에 영향을 미치기 때문에 많은 컨볼루션 마스크의 원소의 합을 1이 되도록 가중치를 정하고 있다. 이 경우 출력 영상은 입력 영상과 같은 평균 밝기를 갖게 됨.
윤곽선 검출에 사용되는 컨볼루션 마스크는 음의 계수를 포함하며, 모든 원소의 합이 0이 되는 경우가 있다. 이 경우 음의 출력 화소값이 나올 수 있다. 보통 상수값을 출력 화소값에 더하고, 더해도 음이 되는 경우에는 출력 화소값을 0으로 놓음.

마스크 종류
: 영상을 평활화하거나 노이즈를 제거하거나, 윤곽 정보를 추출하는 등의 마스크가 사용된다.



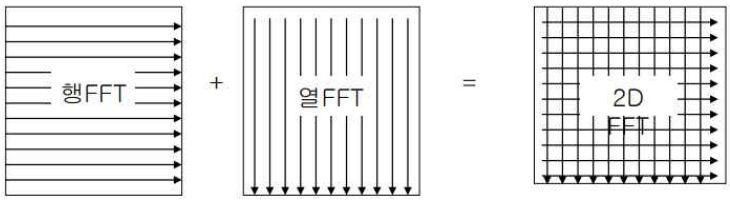
```
void CDSPTestDoc::PrewittEdge()
{
   // 엣지 검출 프로그램
   int row, column,i,j;
   int centerValue1=0;
   int centerValue2=0;
   int sum=0;
   int mask1[3][3]={-1,0,1,  -1,0,1, -1,0,1};
   int mask2[3][3]={1,1,1,  0,0,0, -1,-1,-1};

   for(row=0;row<256;row++){
      for(column=0;column<256;column++){
         for(i=0;i<3;i++){
            for(j=0;j<3;j++){
               centerValue1+=m_InImg[i+row][j+column]*mask1[i][j];
               centerValue2+=m_InImg[i+row][j+column]*mask2[i][j];
            }
         }
         sum=abs(centerValue1)+abs(centerValue2);
         if(sum>255) sum=255;
         m_OutImg[row+1][column+1]=sum;
         centerValue1=0;
         centerValue2=0;
      }
   }
}
```

그림 9-17 PrewittEdge() 함수

- 영상 필터링을 위한 FFT

: 2차원 푸리에 변환

**FFT의 분해 성질**




주파수 영역에서 필터링 단계

(1) 영상의 푸리에 변환을 구한다.

(2) 마스크 함수의 푸리에 변환을 구한다.

(3) 두 변환의 원소끼리 곱한다.

(4) 곱의 결과를 역푸리에 변환 수행한다.

- 버터워스 필터 필터링

영상에서의 고주파는 값의 변화율이 큰 부분으로 주로 윤곽선이며, 저주파는 값의 변화가 적은 영상의 부분을 의미 원 영상에서 FFT 처리 후, IFFT를 다시 거쳐 필터링된 영상을 얻는다.

```
void CDSPTestDoc::TwoD_FFT()
{
 int index=0,i=0,j=0,k=0;
COMPLEX col_data[256];
COMPLEX row_data[256];
for(i=0; i<256; i++){
col_data[i].re=0;
```

```
col_data[i].im=0;
row_data[i].re=0;
row_data[i].im=0;
for(j=0; j<256; j++){
complex_data[k+j].re=m_InImg[i][j];
complex_data[k+j].im=0;
}
k+=256;
}
k=0;
for(int y=0; y<256; y++){
index=y*256;
for(int x=0; x<256; x++)
{
row_data[x].re = complex_data[index].re;
row_data[x].im = complex_data[index++].im;
}
fft(row_data,8,256,1);
index=y*256;
for(x=0; x<256; x++)
{
complex_data[index].re = row_data[x].re;
complex_data[index++].im = row_data[x].im;
}
}
for(int x=0; x<256; x++){
index=x;
for(int y=0; y<256; y++)
{
col_data[y].re = complex_data[index].re;
col_data[y].im = complex_data[index].im;
index += 256;
}
fft(col_data,8,256,1);
index=x;
for(y=0; y<256; y++)
{
complex_data[index].re = col_data[y].re;
complex_data[index].im = col_data[y].im;
index += 256;
```

```
}
}
//result
for(i=0; i<256; i++){
for(j=0; j<256; j++){
m_OutImg[i][j]=20*log(1+abs(complex_data[k+j].re));
if(m_OutImg[i][j]>=170)m_OutImg[i][j]=255;
else m_OutImg[i][j]=0;
}
k+=256;
}
fftview();
}
```

- fft() 함수

**scramble(numpoints,f);**

N=8인 경우 입력의 순서가 0,4,2,6,1,5,3,7과 같이 순서가 바뀌어야 하는데, 이를 스크램블링 동작이라 한다

```
void CTestDoc::scramble(int numpoints, COMPLEX *f)
{
int i,j,m; //loop variable double temp; //temporary storage during swapping
j=0;
for(i=0;i<numpoints;i++) //numpoint배열의 element 수 { if(i>j)
{ //swap real
temp=f[j].re;
f[j].re=f[i].re;
f[i].re=(float)temp;
//swap imagenary
temp=f[j].im;
f[j].im=f[i].im;
f[i].im=(float)temp; }
m=numpoints>>1;
while(( j >= m ) & ( m >= 2 ))
{
j -=m;
m = m >> 1; }
j +=m; } }
```

**butterflies(numpoints,logN,dir,f);**

5. USB CAM 영상처리

```
void CUsbCamDoc::m_negaDetect(BYTE *InImg, BYTE *OutImg, int width, int height){
int index,row,column,i,j;
BYTE m_GrayImg[240][320];
// 입력 영상을 칼라에서 Gray 영상으로 변환하기
for(row=0; row<height; row++){
index = row*width*3;
for(column=0; column<width; column++){
//명도에 대한 NTSC 표준 수식.
m_GrayImg[row][column] = (BYTE)((299*(InImg[index+column*3+2])
+587*(InImg[index+column*3+1])+114*(InImg[index+column*3]))/1000);
}
}
```