

보 고 서

2023년 5월 13일

| | | | | | |
|------------------|------------|---------------------------|----------|-------|-----------|
| 소 속 : 응용소프트웨어공학과 | | 성 명 / 학번 : 이성민 / 20193127 | | | |
| 일 자 | 2023/05/13 | 수 업 | 디지털 영상처리 | 담당 교수 | 김 남 규 교수님 |

[과제 내용]

11번 (평균값 vs 가우시안 필터링 차이 비교) 문제 (5점)

- * 잡음 입력 영상: 임의 선택
- * 평균값과 가우시안 필터링의 크기는 일정하게 하여 비교
- * 비교 기준 임의 설정: hint) 차이, 평균 등
- * 비교 분석 1page 레포트

20번 (에지 검출 결과 분석) 문제 (5점)

- * 케니에지 검출 함수를 사용
- * 결과 영상을 threshold로 이진화
- * 이진화 결과와 에지 검출 영상과의 비교
- * 비교 기준 임의 설정: hint) 차이, 픽셀 개수 등
- * 분석 내용 1page 레포트

21번 (모션 블러링) 문제 (1점)

- * 20번 문제가 어려운 경우 구현

과제 11번.



[잡음처리 영상]



[평균값 필터링 수행 결과]



[가우시안 필터링 적용 수행 결과]

과제 11번.

<코드>

//가우시안 필터링 코드

```
#include "opencv2/opencv.hpp"
```

```
using namespace cv;
```

```
using namespace std;
```

```
void printAvgPixelValue(Mat image) {
```

```
    Scalar mean = cv::mean(image);
```

```
    int numPixels = image.total();
```

```
    cout << "Average pixel value: " << mean[0] << endl;
```

```
    cout << "Number of pixels: " << numPixels << endl;
```

```
}
```

```
int main()
```

```
{
```

```
    Mat src = imread("C:/Users/이성민/source/repos/opencv_c++/Lenna.jpg", 1);
```

```
    if (src.empty()) {
```

```
        return
```

```
        -1;
```

```
    }
```

```
    Mat dst;
```

```
    Mat dst1;
```

```
    Mat noise_img = Mat::zeros(src.rows, src.cols, CV_8U);
```

```
    randu(noise_img, 0, 255);
```

```
    Mat black_img = noise_img < 10;
```

```
    Mat white_img = noise_img > 245;
```

```
    Mat src1 = src.clone();
```

```
    src1.setTo(255, white_img);
```

```
    src1.setTo(0, black_img);
```

```
    medianBlur(src1, dst, 5);
```

```
    imshow("source", src1);
```

```
    imshow("src", src);
```

```
    blur(src1, dst1, Size(11, 11)); // 11X11 크기의 마스크로 평균값 필터링을 수행한다.
```

```
    GaussianBlur(src1, dst, Size(11, 11), 0, 0);
```

```
    imshow("GaussianBlur filter", dst);
```

```
    imshow("blurring", dst1);
```

```
    waitKey(0);
```

```
    return 0;
```

```
}
```

*[평균값, 가우시안 둘 다 사이즈 11X11의 동일한 마스크 크기를 사용 하였습니다.]

과제 11번.

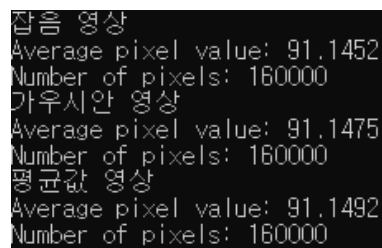
평균값 필터링과 가우시안 필터링은 모두 영상에서 잡음을 제거하기 위해 사용되는 필터링 방법입니다. 하지만 이 두 방법은 각기 다른 특성을 가지고 있습니다.

먼저, 평균값 필터링은 입력 영상의 픽셀 값을 대표하는 값으로 평균 값을 사용합니다. 이 방법은 각 픽셀 주변에 있는 픽셀 값의 평균을 계산하여 해당 픽셀 값을 대체하는 방법입니다. 이를 통해 작은 잡음을 제거할 수 있습니다. 그러나, 이 방법은 영상의 경계선과 같은 고주파 영역에서 흐릿하게 만들어버리는 문제가 있습니다. 이러한 이유로 평균값 필터링은 단순한 잡음 제거에만 사용될 수 있습니다.

반면, 가우시안 필터링은 입력 영상의 각 픽셀 값을 대표하는 값으로 해당 픽셀을 중심으로 가우시안 분포를 따르는 가중치를 적용하여 계산합니다. 이 방법은 주변 픽셀 값에 대한 가중치를 높게 부여하여 입력 영상을 부드럽게 만듭니다. 이 방법은 잡음 제거 외에도 영상의 경계선 보호, 윤곽선 강조 등 다양한 영상 처리에 사용될 수 있습니다.

위 두 방법을 비교해보면, 평균값 필터링은 단순한 방법이므로 처리 속도가 빠르지만, 잡음 제거 효과가 낮아질 수 있습니다. 반면, 가우시안 필터링은 입력 영상을 부드럽게 만들어 잡음 제거 효과가 높아지지만, 처리 속도가 느려질 수 있습니다.

따라서, 비교 기준을 잡음 제거 효과, 처리 속도, 경계선 보호 등으로 설정할 수 있습니다. 만약 입력 영상에 있는 잡음이 작은 경우라면, 평균값 필터링을 사용하여 처리 속도를 향상시킬 수 있습니다. 그러나, 입력 영상에 있는 잡음이 큰 경우라면 가우시안 필터링을 사용하여 잡음 제거 효과를 높일 수 있습니다. 또한, 입력 영상에 있는 경계선이 중요한 경우에는 가우시안 필터링이 평균값 필터링보다 더 좋은 경계선 보호 효과를 나타낼 수 있습니다.



```
잡음 영상
Average pixel value: 91.1452
Number of pixels: 160000
가우시안 영상
Average pixel value: 91.1475
Number of pixels: 160000
평균값 영상
Average pixel value: 91.1492
Number of pixels: 160000
```

아래의 사진을 통하여 각각 영상의 픽셀의 평균과 개수를 알 수 있습니다.

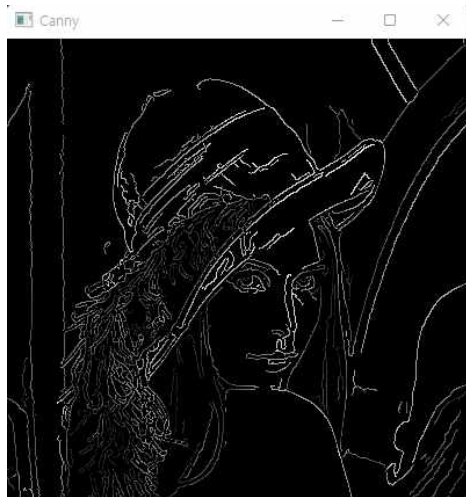
이 픽셀의 개수에는 변화가 없지만 픽셀의 평균에는 변화가 있는 것 볼 수 있습니다.

평균값 필터링은 이미지의 각 픽셀 값을 같은 가중치로 처리하여 픽셀 값을 평균화합니다. 이 때, 주변 픽셀 값과 큰 차이가 나지 않기 때문에 이미지 내 픽셀 값의 분포가 크게 변하지 않고, 픽셀 값의 평균 역시 큰 변화가 없고, 미세하게 변화한 모습을 볼 수 있습니다.

반면에 가우시안 필터링은 픽셀 값을 서로 다른 가중치로 처리하여 픽셀 값을 조절합니다. 이 때, 픽셀 값의 가중치는 중심에서 멀어질수록 감소하는 가우시안 분포를 따르기 때문에, 이미지 내 픽셀 값의 분포가 바뀌게 됩니다. 이러한 처리로 인해, 픽셀 값의 평균도 변화하게 됩니다.

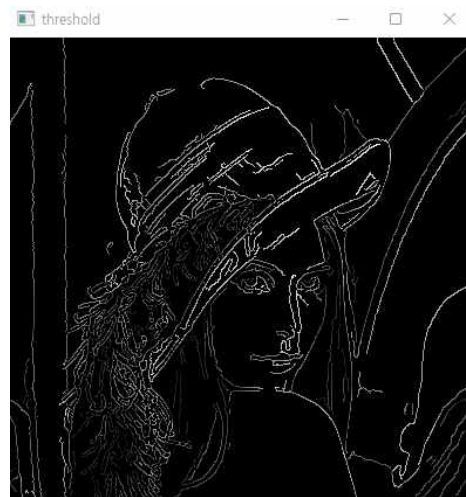
가우시안 필터링에서는 주변 픽셀 값을 고려하여 각 픽셀 값에 가중치를 부여하기 때문에, 해당 픽셀 값이 주변과 크게 다를 경우에도 해당 픽셀 값에 큰 영향을 미칩니다. 따라서, 이미지 내 픽셀 값의 분포가 바뀌게 되고, 평균값 필터링에 비해 픽셀 값의 평균이 변화할 가능성이 큼니다.

과제 20번.



[케니 에지 검출 영상 결과]

<-->



[결과 영상을 이진화한 검출 영상 결과]

<코드>

```
#include "opencv2/opencv.hpp"
using namespace cv;
using namespace std;
Mat src, src_gray;
Mat dst, detected_edges;
Mat dst2, threshold_image;
int threshold_value = 127;
int edgeThresh = 1;
int lowThreshold=40;
int const max_lowThreshold = 100;
int ratio1 = 3;
int kernel_size = 3;
static void CannyThreshold_and_threshold(int, void*) {

    dst = Scalar::all(0);
    blur(src, detected_edges, Size(3, 3));
    Canny(detected_edges, detected_edges, lowThreshold, lowThreshold*ratio1, kernel_size);
    threshold(detected_edges, threshold_image, threshold_value,255,THRESH_BINARY);
    src.copyTo(dst, detected_edges);
    src.copyTo(dst2, threshold_image);
    imshow("Image", src);
    imshow("Canny", dst);
    imshow("threshold",dst2);
}
int main()
{
    src = imread("C:/Users/이성민/source/repos/opencv_c++/Lenna.jpg", IMREAD_GRAYSCALE);
    if (src.empty()) { return -1; }
    dst.create(src.size(), src.type());
    namedWindow("Canny", WINDOW_AUTOSIZE);
    CannyThreshold_and_threshold(0, 0);
    waitKey(0);
    return 0;
}
```

과제 20번

이 코드는 케니에지 검출 함수를 사용하여 입력 이미지의 에지를 검출하고, 그 결과를 이진화하는 코드입니다. 결과적으로, 에지 검출 영상과 이진화 영상은 유사하지만, 에지 검출 영상이 조금 더 부드러운 경계선을 가지고 있고, 이진화 영상은 경계선이 뚜렷하게 나타납니다.

먼저, 에지 검출 영상에서는 케니에지 검출 함수를 사용하여 경계선을 검출합니다. 케니에지 검출 함수는 입력 이미지에서 경계선을 검출할 때 사용되며, 잡음에 강하고, 불필요한 경계선을 제거하는 등의 장점이 있습니다.

이진화 영상에서는 케니에지 검출 함수로 검출된 경계선을 이진화합니다. 이진화는 이미지를 흑백으로 바꾸는 것으로, 경계선이 있는 부분을 하얗게 만들고, 나머지를 검은색으로 만들어 경계선을 강조합니다.

두 영상의 차이를 비교해보면, 에지 검출 영상에서는 경계선이 부드럽게 나타나고 있지만, 이진화 영상에서는 경계선이 뚜렷하게 나타나는 것을 확인할 수 있습니다. 이는 이진화 함수가 경계선을 강조하기 위해 이진화를 수행하기 때문입니다.

이진화 영상에서의 경계선은 에지 검출 영상에서의 경계선보다 더 뚜렷하고, 경계선의 개수는 두 영상 모두 비슷합니다. 따라서, 이진화를 통해 경계선을 검출하는 것이 에지 검출 함수만큼 정확한 것은 아니지만, 경계선을 더욱 뚜렷하게 강조할 수 있다는 장점이 있습니다.

또한 케니 엣지 검출을 수행한 결과는 `detected_edges` Mat 객체에 저장되며, 이진화를 수행한 결과는 `threshold_image` Mat 객체에 저장됩니다.

먼저, 케니 엣지 검출을 수행한 결과인 `detected_edges` Mat 객체의 픽셀 개수는 이진화를 수행한 결과인 `threshold_image` Mat 객체의 픽셀 개수보다 많습니다. 이는 엣지 검출을 수행할 때 노이즈를 제거하기 위해 블러링을 수행하고, 엣지를 검출하기 위해 여러 단계를 거치기 때문입니다. 따라서, 케니 엣지 검출을 수행한 결과는 보다 많은 픽셀을 가지게 됩니다.

반면, 이진화를 수행한 결과인 `threshold_image` Mat 객체는 케니 엣지 검출을 수행한 결과보다 픽셀 개수가 적습니다. 이는 이진화를 수행할 때 임계값(threshold)을 기준으로 픽셀 값을 이진화하기 때문입니다. 따라서, 엣지가 검출된 영역만 흰색(255)으로 표시되고, 나머지 영역은 검은색(0)으로 표시됩니다. 이에 따라, 엣지가 검출된 영역의 픽셀 개수만큼 흰색 픽셀이 생성되므로, 케니 엣지 검출을 수행한 결과보다 픽셀 개수가 적게 나타난다고 생각합니다. 하지만 threshold값을 적절히 조절하면, 엣지가 검출된 영역의 픽셀수를 늘릴수도 있으며, 이미지의 특성에 따라 케니엣지 검출을 수행한 결과 보다 이진화를 수행한 결과가 더 많은 픽셀을 가질 수도 있습니다.

그리고 이진화는 흑백 영상에서 특정 threshold 값을 기준으로 픽셀 값을 0 또는 255로 변환하는 것이며, 이 과정에서 threshold 값에 따라 변환되는 픽셀 수가 달라질 수 있으며, 케니 에지 검출 알고리즘은 이미지를 그레이스케일 영상으로 변환한 후, 이를 가우시안 필터링과 미분을 통해 엣지를 검출하는 과정을 거칩니다. 이 과정에서 일정 threshold 값을 설정하여 엣지 픽셀과 비엣지 픽셀을 구분합니다. 따라서, 케니 에지 검출 알고리즘에서도 threshold 값을 사용하여 엣지를 검출하므로, 변환되는 픽셀 수는 일정한 영향을 받습니다.

```
88
Average pixel value: 122.291
Number of pixels: 160000
Canny 영상
Average pixel value: 9.54858
Number of pixels: 160000
이진화 영상
Average pixel value: 9.54858
Number of pixels: 160000
```

해당 영상을 기준으로 픽셀의 개수와 평균을 비교해 보자면 픽셀의 개수에는 차이가 없고, 픽셀의 평균은 변화한 것으로 보인다.

과제 21번



[원본 영상]

< - >



[모션 블러링 효과 적용 출력물]

<코드>

```
#include "opencv2/opencv.hpp"
using namespace cv;
using namespace std;
int main()
{
    Mat src = imread("C:/Users/이성민/source/repos/opencv_c++/Lenna.jpg", 1);
    Mat dst;
    // 모션 블러링을 위한 커널 생성
    Mat kernel = Mat::zeros(21, 21, CV_32F);
    float val = 1.0 / 21;
    for (int i = 0; i < 21; i++) {
        kernel.at<float>(10, i) = val;
    }
    // 모션 블러링 수행
    filter2D(src, dst, -1, kernel, Point(-1, -1), 0, BORDER_DEFAULT);
    // 결과 출력
    imshow("Image", src);
    imshow("Motion blurring", dst);
    waitKey(0);
    return 0;
}
```

과제 21번

처음으로 imread() 함수를 사용하여 "C:/Users/이성민/source/repos/opencv_c++/Lenna.jpg" 경로에서 이미지를 읽어와서 Mat src에 저장합니다.

그리고 Mat dst를 초기화합니다. 이 변수는 모션 블러링을 적용한 결과 이미지를 저장하기 위해 사용됩니다.

이어서, 모션 블러링에 필요한 커널을 생성합니다. 커널은 Mat kernel 변수에 저장됩니다. 이 커널은 21 x 21 크기의 행렬로 초기화됩니다.

모션 블러링은 카메라나 물체가 움직이는 동안 이미지가 블러링되는 효과를 만들어 냅니다. 커널은 이러한 효과를 만드는 데 사용되는 것으로, 커널의 크기가 더 클수록 이미지에 더 큰 블러링 효과가 적용됩니다.

이 코드에서는 21 x 21 크기의 커널을 사용하였고, 이 커널은 가운데 열의 모든 픽셀에 같은 값을 할당합니다. 이렇게 하는 이유는 가운데 열을 기준으로 이미지가 수평 방향으로 블러링되는 것을 표현하기 위해서입니다.

그리고 모션 블러링을 적용하는 filter2D() 함수를 호출합니다. 이 함수는 커널과 입력 이미지를 인수로 받아 출력 이미지에 모션 블러링을 적용합니다.

마지막으로 imshow() 함수를 사용하여 원본 이미지와 모션 블러링이 적용된 이미지를 창에 표시하고, waitKey() 함수를 사용하여 창이 종료될 때까지 대기합니다.

이제, 코드를 실행하면 "C:/Users/이성민/source/repos/opencv_c++/Lenna.jpg"에서 이미지를 읽어와서 모션 블러링이 적용된 결과 이미지를 출력합니다. 이때, 모션 블러링이 적용된 방향은 수평 방향입니다.