

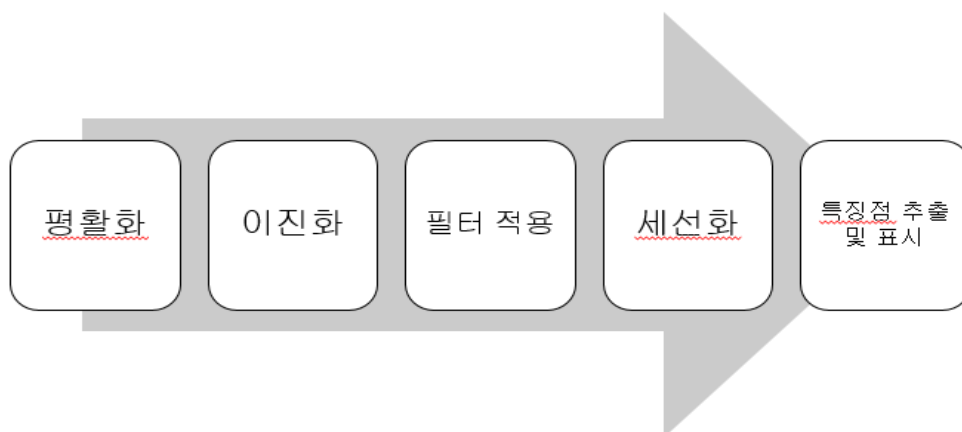
1. 서론

<지문패턴의 특징점>

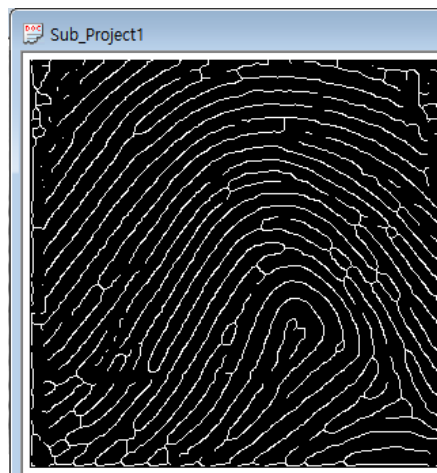
- a. 융선(ridge): 패턴에서 선 모양으로 나타나는 것으로 산맥과 같이 솟아 오른 부분
- b. 골(valley): 융선과 융선 사이에 계곡과 같이 파인 부분
- c. 융선 분기점(ridge bifurcation): 융선이 두 갈래로 갈라지는 점
- d. 융선 끝점(ridge ending point): 융선이 끝나는 곳
- e. 지문 중심점



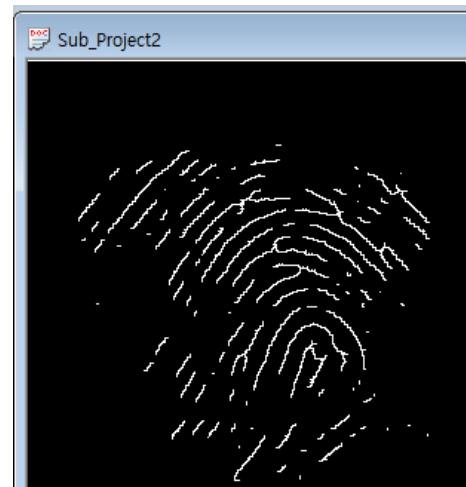
2. 진행 과정



- a. **평활화**: 명암 값의 분포가 균일화 되어 영상이 향상 된다. 이를 통해 영상을 보다 선명하게 만들어 준다.
- b. **블록 이진화**: 기본 이진화 대신 사용. 원하는 크기의 블록을 설정하고 그 블록마다의 명암값을 고려하여 임계값을 설정하는 방법.
- c. **필터**: 미디언 필터를 적용하여 잡음을 제거하여 준다. 또한 침식 연산 팽창 연산을 번갈아 하는 열림연산을 적용하여 준다.
- d. **세션화**: 두꺼운 선을 얇게 만들어 준다.



<블록 이진화 후 세션화>



<기본 이진화 후 세션화>

- e. **특징점 추출 및 표시**: 적절한 조건을 사용하여 지문의 특징점을 찾는다.

3. 코드

a. 평활화

```
LOW = 0;
HIGH = 255;
for (i = 0; i < 256; i++) {
    m_HIST[i] = LOW; //초기화
}
for (i = 0; i < m_size; i++) {
    value = (int)m_InImg[i];
    m_HIST[value]++; // 빈도수 조사
}
for (i = 0; i < 256; i++) {
    SUM += m_HIST[i]; //합을 조사
    m_Sum_Of_HIST[i] = SUM;
    //쓰레기 값이 저장될 수도 있으니 초기화를 해준다.
    //존재하는 밝기 값을 최대한 고르게 분포되게 한다.
}
for (i = 0; i < m_size; i++) {
    Temp = m_InImg[i];
```

```
mid_Image[i] = (unsigned char)(m_Sum_Of_HIST[Temp] * HIGH / m_size);
}
```

● 결과



<원 영상>



<평활화 영상>

b. 블록 이진화

단순 임계값으로 이진화를 시키는 방법과 달리 적당한 크기의 블록을 취하여 그 블록 내의 평균을 이용하여 이진화를 시킨다. 본 코드에서는 4*4의 블록 단위로 이진화하였다.

6	5	7	8	$\text{Sum} = 6+5+7+ \dots + 5 = 106$ $\text{Threshold} = 106 / 16 = 6.625$		0	0	9	9
9	3	8	2			9	0	9	0
9	9	9	6			9	9	9	0
7	7	6	5			9	9	0	0

```
double threshold = 0.0;
for (i = 0; i < m_height; i = i + 4) {
    for (j = 0; j < m_width; j = j + 4) {
        SUM = 0.0;
        for (int r = 0; r < 4; r++) {
            for (int s = 0; s < 4; s++) {
                SUM += m_tempImage[i + r][j + s];
            }
        }
        threshold = SUM / 16;
        for (int r = 0; r < 4; r++) {
            for (int s = 0; s < 4; s++) {
                if (m_tempImage[i + r][j + s] > threshold)
                    m_tempImage[i + r][j + s] = 255;
                else
                    m_tempImage[i + r][j + s] = 0;
            }
        }
    }
}
```

```

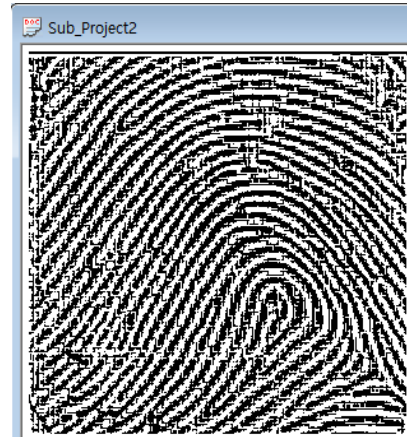
}
}

```

- 결과



<평활화 영상>



<블록 이진화 영상>

c. 필터 적용

잡음 제거를 위하여 median filter를, 선을 명확하게 하기 위하여 침식 연산과 팽창 연산을 같은 횟수만큼 번갈아 하는 열림 연산을 수행한다. 여기서는 각 1회씩 적용하였다.

```

unsigned char* CSub_ProjectDoc::OnMedianSub_bw(unsigned char* m_Image)
{
    int i, j, n, m, M = 5, index = 0; // M = 서브샘플링비율
    double *Mask, Value;
    Mask = new double[M*M]; // 마스크의크기결정
    unsigned char* filter_Image;
    filter_Image = new unsigned char[m_size];
    m_tempImage = Image2DMem(m_height + 4, m_width + 4);
    for (i = 0; i < m_height; i++) {
        for (j = 0; j < m_width; j++) {
            m_tempImage[i+2][j+2] = (double)m_Image[i*m_width + j];
        }
    }
    for (i = 0; i < m_height + 4; i++) {
        m_tempImage[i][0] = 0;
        m_tempImage[i][m_width + 2] = 0;
    }
    for (j = 0; j < m_width + 4; j++) {
        m_tempImage[0][j] = 0;
        m_tempImage[m_height + 2][j] = 0;
    }
    for (i = 0; i < m_height; i++) {
        for (j = 0; j < m_width; j++) {
            for (n = 0; n < M; n++) {
                for (m = 0; m < M; m++) {
                    Mask[n*M + m] = m_tempImage[i + n][j + m];
                    // 입력영상을블록으로잘라마스크배열에저장
                }
            }
        }
    }
}

```

```

        OnBubbleSort(Mask, M*M); // 마스크에 저장된값을정렬
        Value = Mask[(int)(M*M / 2)]; // 정렬된값중가운데값을선택
        filter_Image[index] = (unsigned char)Value;
        // 가운데값을출력
        index++;
    }
}
return filter_Image;
}
unsigned char* CSub_ProjectDoc::OnDilation_bw(unsigned char* Img) {
    int i, j, n, m;
    double Mask[3][3] = { { 0., 0., 0. }, { 0., 0., 0. }, { 0., 0., 0. } };
    // 팽창처리를위한마스크
    double **tempInput, S = 0.0;
    unsigned char * filter_Image;
    filter_Image = new unsigned char[m_size];
    tempInput = Image2DMem(m_height + 2, m_width + 2);
    for (i = 0; i < m_height; i++) {
        for (j = 0; j < m_width; j++) {
            tempInput[i + 1][j + 1] = (double)Img[i * m_width + j];
        }
    }
    for (i = 0; i < m_height; i++) {
        for (j = 0; j < m_width; j++) {
            for (n = 0; n < 3; n++) {
                for (m = 0; m < 3; m++) {
                    if (Mask[n][m] == tempInput[i + n][j + m]) {
                        // 마스크와같은값이있는지조사
                        S += 1.0;
                    }
                }
            }
        }
        if (S == 9.0)
            filter_Image[i * m_width + j] = (unsigned char)0.0;
        // 모두일치하면출력값은 0
        else
            filter_Image[i * m_width + j] = (unsigned char)255.0;
        // 모두일치하지않으면출력값은 255
        S = 0.0;
    }
}
delete[] tempInput;
return filter_Image;
}
unsigned char* CSub_ProjectDoc::OnErosion_bw(unsigned char* Img)
{
    int i, j, n, m;
    double Mask[3][3] = { { 255., 255., 255. },
        { 255., 255., 255. },
        { 255., 255., 255. } };

    // 침식연산을위한마스크
    double **tempInput, S = 0.0;
    unsigned char* filter_Image;
    filter_Image = new unsigned char[m_size];
    tempInput = Image2DMem(m_height + 2, m_width + 2);
    unsigned char * tt_Image = new unsigned char[m_size];
    for (i = 0; i < m_size; i++) {
        if (Img[i] > 127)
            tt_Image[i] = 255;
        else
            tt_Image[i] = 0;
    }
    for (i = 0; i < m_height; i++) {

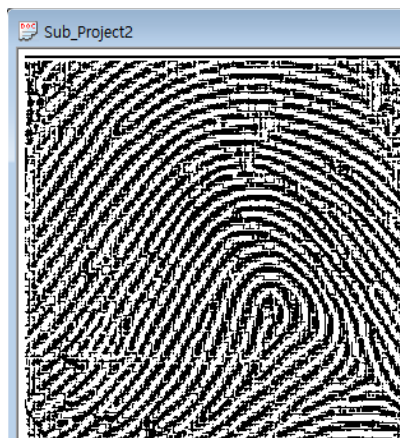
```

```

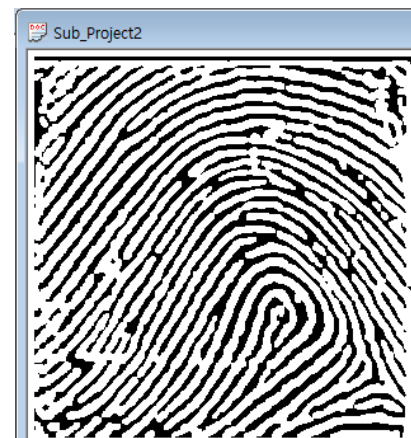
        for (j = 0; j < m_width; j++) {
            tempInput[i + 1][j + 1] = (double)tt_Image[i * m_width + j];
        }
    }
    for (i = 0; i < m_height; i++) {
        for (j = 0; j < m_width; j++) {
            for (n = 0; n < 2; n++) {
                for (m = 0; m < 2; m++) {
                    if (Mask[n][m] == tempInput[i + n][j + m]) {
                        // 마스크와 같은 값이 있는지 조사
                        S += 1.0;
                    }
                }
            }
            if (S == 9.0)
                filter_Image[i * m_width + j] = (unsigned char)255.0;
                // 값이 모두 일치하면 출력값은 255
            else
                filter_Image[i * m_width + j] = (unsigned char)0.0;
                // 모두 일치하지 않으면 출력값은 0
            S = 0.0; // reset
        }
    }
    delete[] tempInput;
    return filter_Image;
}

```

● 결과



<블록 이진화 영상>



<필터 영상>

d. 세선화

Zhang-suen의 세선화 알고리즘을 사용하여 세선화를 하였다.

- 두꺼운 선을 한 픽셀의 선으로 표현 하는 알고리즘으로 선의 가운데 성분을 추출하여 세선화를 한다. 경계점들을 삭제하여 영역의 골격을 만든 후 두 개 영역을 연결하는 픽셀은 지우지 않는 두 번의 sub-iterating을 거친다.

```
m_tempImage = Thin(mid_Image);
```

```

double** CSub_ProjectDoc::Thin(unsigned char* InImage)
{
    int i, j;
    double ** img, **timg;

    img = Image2DMem(m_width + 2, m_height + 2);
    timg = Image2DMem(m_width + 2, m_height + 2);
    for (i = 0; i < m_height; i++) {
        for (j = 0; j < m_width; j++) {
            img[i + 1][j + 1] = InImage[i * m_width + j];
        }
    }
    // 영상의 가장자리에 0으로 자리를 채워줌.
    for (i = 0; i < m_height + 2; i++) {
        img[i][0] = 0;
        img[i][m_width + 1] = 0;
    }
    for (j = 0; j < m_width + 2; j++) {
        img[0][j] = 0;
        img[m_height + 1][j] = 0;
    }
    //영상의 물체를 1, 배경을 0로 바꿈.
    for (i = 1; i < m_width + 1; i++)
    {
        for (j = 1; j < m_height + 1; j++)
        {
            if (img[i][j] > 0) img[i][j] = 0; //black
            else img[i][j] = 1; //white
            timg[i][j] = 0; //초기화
        }
    }
    bool again = 1;
    int N;
    while (again)
    {
        again = 0;
        for (i = 1; i < m_width + 1; i++)
        {
            for (j = 1; j < m_height + 1; j++)
            {
                if (img[i][j] != 1) continue;
                //영상의 중앙 부분을 중심으로 1인 영상의 개수 계산.
                N = nays(img, i, j);
                if ((N >= 2 && N <= 6) && Connect(img, i, j) == 1)
                {
                    if ((img[i][j + 1] * img[i - 1][j] * img[i][j - 1] ==
0) && (img[i - 1][j] * img[i + 1][j] * img[i][j - 1] == 0))
                    {
                        timg[i][j] = 1;
                        again = 1;
                    }
                }
            }
        }
        DeleteA(img, timg, m_width, m_height);
        if (again == 0)
            break;
        for (i = 1; i < m_width + 1; i++){
            for (j = 1; j < m_height + 1; j++){
                if (img[i][j] != 1) continue;
                N = nays(img, i, j);
                if ((N >= 2 && N <= 6) && Connect(img, i, j) == 1)
                {

```

```

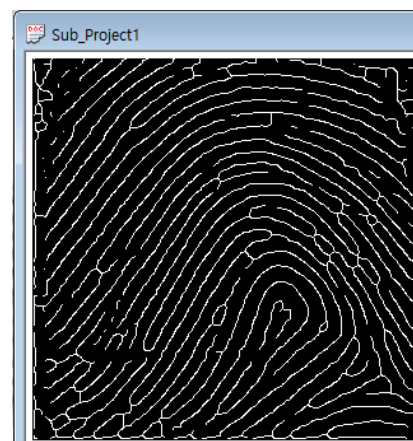
        if ((img[i - 1][j] * img[i][j + 1] * img[i + 1][j] == 0) &&
            (img[i][j + 1] * img[i + 1][j] * img[i][j - 1] == 0)) {
            timg[i][j] = 1;
            again = 1;
        }
    }
}
DeleteA(img, timg, m_width, m_height);
}
//다시 환원
for (i = 0; i < m_width + 2; i++)
{
    for (j = 0; j < m_height + 2; j++)
    {
        if (img[i][j] > 0) img[i][j] = 255;
        else img[i][j] = 0;
    }
}
return img;
}

```

● 결과



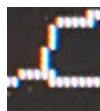
<필터 영상>

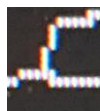


<세션화 영상>

e. 특징점 추출 및 표시

지문의 주 특징인 분기점과 끝점을 주로 찾으려고 하였다. 분기점의



세션화된 코드를 확대하여 보면  과 같은 모습을 나타내는 것을 알 수 있기에 다음과 같은 마스크를 디자인 하였다. 원 영상과 마스크의 차이를 이용하여 해당 형태를 찾을 수 있게 하였다. 위 형태와 비슷할 경우 1을 return하고 그렇지 않은 경우 0을 return하여 위 형태와 비슷한 경우에만 찾을 수 있도록 하였다.

0	255	0
255	255	255
0	255	0

마스크

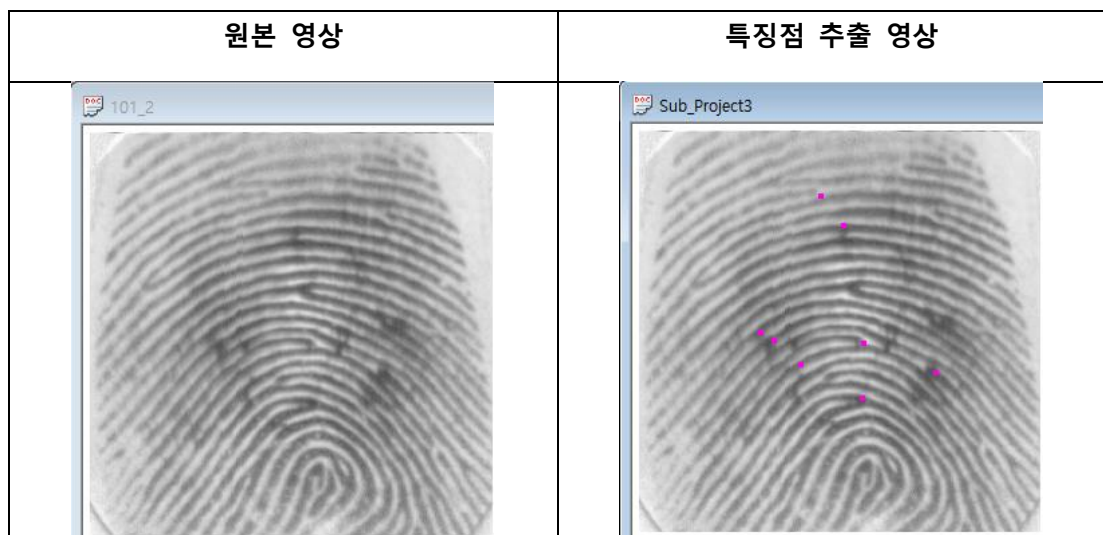

```

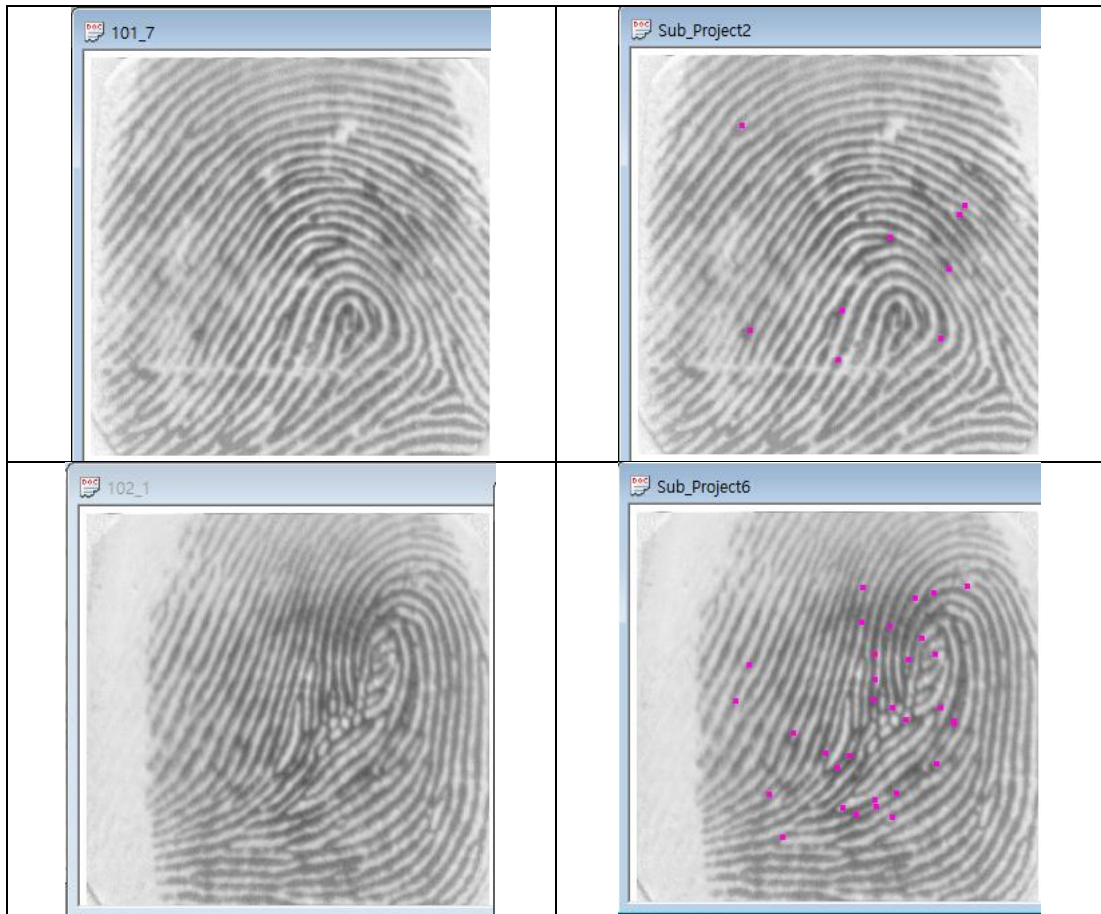
int CSub_ProjectDoc::Searching(unsigned char* m_Img, int i, int j)
{
    double Searching_M[3][3] = { {0.,255.,0.},{255.,255.,255.},{0.,255.,0.} };
    int sear = 0;
    int r, s;
    for (r = -1; r < 2; r++) {
        for (s = -1; s < 2; s++) {
            if (Searching_M[r + 1][s + 1] - m_Img[(i + r) * m_width + j + s] == 0)
                sear++;
        }
    }
    if (sear >= 8)
        return 1;
    else
        return 0;
}

//해당 위치에 그리기
if(search ==1){
    for (int k = 0; k < 4; k++) {
        for (int l = 0; l < 4; l++) {
            m_OutputImage[((i + k) * m_Re_width + j + 1) * 3 + 0] = 221;
            m_OutputImage[((i + k) * m_Re_width + j + 1) * 3 + 1] = 0;
            m_OutputImage[((i + k) * m_Re_width + j + 1) * 3 + 2] = 255;
        }
    }
}

```

4. 결과





이진화가 잘 되지 않은 부분, 영상이 흐릿한 부분(선들이 뭉쳐있는 부분 등)에서는 어쩔 수 없는 오류가 발생한다. 하지만 분기점등 갈라지는 부분, 끝점을 그래도 잘 찾을 수 있음을 알 수 있다. 아직 searching부분의 논리가 많이 부족한 것 같다. 지문의 선들을 좀 더 뚜렷하게 분리할 수 있다면 좀 더 나은 결과가 나올 것 같다.