



프로그래밍언어
Term Project Final Report

전자공학과
21812009
조은영

목차

1. 서론.....	2
프로젝트 개요	2
요구조건 분석	3
해결방향	3
2. 본론	4
해결방법 설명	4
대략적인 순서도	5
프로그래밍 코드 설명(영상처리)	7
실험 결과(영상처리)	11
프로그래밍 코드 설명(전공 시뮬레이터)	21
실험 결과(전공 시뮬레이터)	28
3. 결론	34
4. 부록	35

1. 서론

프로젝트 개요

나만의 영상처리 : MFC의 버튼을 이용해 영상의 밝기를 조절해주는 영상처리 기법 구현.

나만의 전공시뮬레이터 : 제어공학에서 배우는 Routh-Hurwitz Stability Criterion을 전공시뮬레이터로 구현.

Routh-Hurwitz Stability Criterion 이론

• Routh Hurwitz Stability Criterion

특성방정식을 이용해 시스템이 안정한지 불안정한지 판별해줌.

특성방정식의 근이 s-plane의 우반면에 하나라도 존재하는 경우에 불안정하다



Routh array 완성하기

ex) 특성방정식 : $s^3 + 2s^2 + s + 1 = 0$ 인 경우

$$\begin{array}{c|cc|c} s^3 & 1 & 1 & 0 \\ s^2 & 2 & 1 & 0 \\ s^1 & -\frac{1}{2} & 1 & 0 \\ s^0 & 1 & 0 & 0 \end{array}$$

$$\begin{array}{c|cc|c} -\frac{1}{2} & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array} = 0$$

$$= 1$$

ex) $s^3 + 2s^2 + s + 1 = 0$ 인 경우

$$\begin{array}{c|cc|c} s^3 & 1 & 1 & 0 \\ s^2 & 2 & 1 & 0 \\ s^1 & -\frac{1}{2} & 1 & 0 \\ s^0 & 1 & 0 & 0 \end{array}$$

$$\begin{array}{c|cc|c} -\frac{1}{2} & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array} = 0$$

$$= \frac{1}{2}$$

$$\begin{array}{c|cc|c} -\frac{1}{2} & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array} = 1$$

첫번째 열을 보고, 부호변화가 없으면 안정.

부호변화가 있으면 불안정 & 부호변화 횟수 = 우반면에 위치한 근의 개수

ex) 첫번째 열이

$$\begin{array}{l} \text{부호변화} \left\{ \begin{array}{l} 1 \\ -2 \end{array} \right. \\ \text{부호변화} \left\{ \begin{array}{l} 1 \\ 1 \end{array} \right. \end{array}$$

\therefore 불안정 & 우반면에 위치한 근의 개수 : 2개

첫번째 열이

$$\begin{array}{l} 1 \\ 2 \\ 1 \end{array} \quad \text{부호변화 X}$$

\therefore 안정 & 우반면에 위치한 근의 개수 : 0개

요구조건 분석

- Condition 1: Use MFC (50점)
- Condition 2: Load *.pgm (your face) file using imageClass (50점)
- Condition 3: Display image (50)
- Condition 4: Select unique your image processing and show the result (50점)
- Condition 5: GUI interface design-멋지게 독창적으로 꾸미기 (100점)
- Condition 6: 전자공학 전공 시뮬레이터 클래스 생성 및 GUI에 시연(100점)

해결방향

영상처리의 경우 나만의 영상처리 기법을 먼저 클래스로 구현한 다음, MFC를 이용해 각 영상처리 방법마다 버튼을 만들어 버튼과 함수를 연결시킨다. 이후, 나만의 전공시뮬레이터를 클래스로 구현한 다음, 영상처리와 마찬가지로 MFC를 이용해 버튼을 만들어 버튼과 구현한 함수를 연결시킨다.

2. 본론

해결방법 설명 - 영상처리

MFC의 버튼을 이용해 영상의 밝기를 조절해주는 영상처리 기법 구현.

먼저 기존에 영상처리방법을 클래스로 구현했던 imageData.h와 imageData.cpp파일에 추가로 나만의 영상처리 방법을 구현한다.

이때, overflow 혹은 underflow가 발생하지 않게 if문을 사용하여 제어한다.

영상처리할 영상을 MFC로 불러오는 것과 불러온 영상을 MFC 화면에 출력하는 것은 14주차 2차시의 강의를 참고하여 구현한다.

여기에서 추가로 Button들을 생성하여 Inversion기능을 할 Button은 Inversion함수와 연결시키고, Binarization기능을 할 Button은 Binarization 함수와 연결시키고, 나만의 영상처리 기능을 할 Button은 나만의 영상처리 방법을 구현한 함수와 연결시킨다.

해결방법 설명 - 전공 시뮬레이터

제어공학에서 배우는 Routh-Hurwitz Stability Criterion을 전공시뮬레이터로 구현.

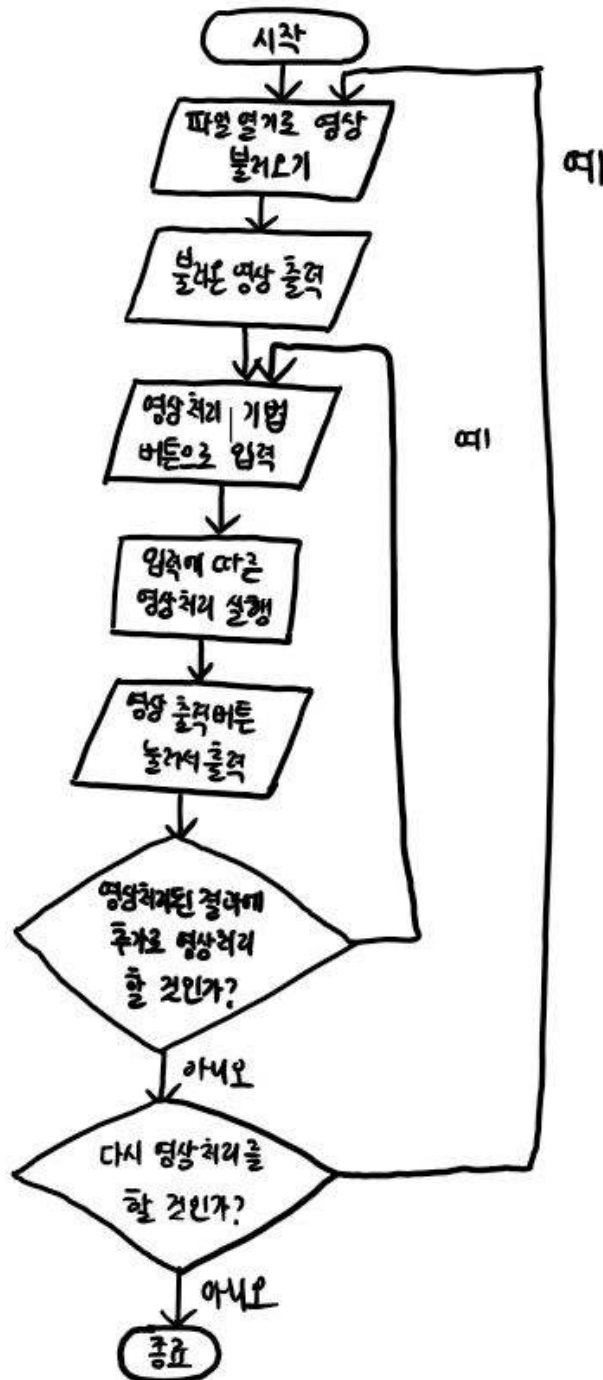
Routh-Hurwitz Stability Criterion을 판별하기 위해서는 특성방정식의 차수와 각 차수항의 계수를 입력으로 받아와야된다. 이를 받아오기 위해서 MFC에서 Edit Control과 Button을 사용하여 값을 입력받아온다. 먼저 이 기능을 구현한다.

그 다음, 입력 받은 값으로부터 Routh-Hurwitz Stability Criterion을 하는 클래스의 생성자 함수와 소멸자 함수, 멤버 변수의 값을 초기화 하고, Routh array를 생성한다.

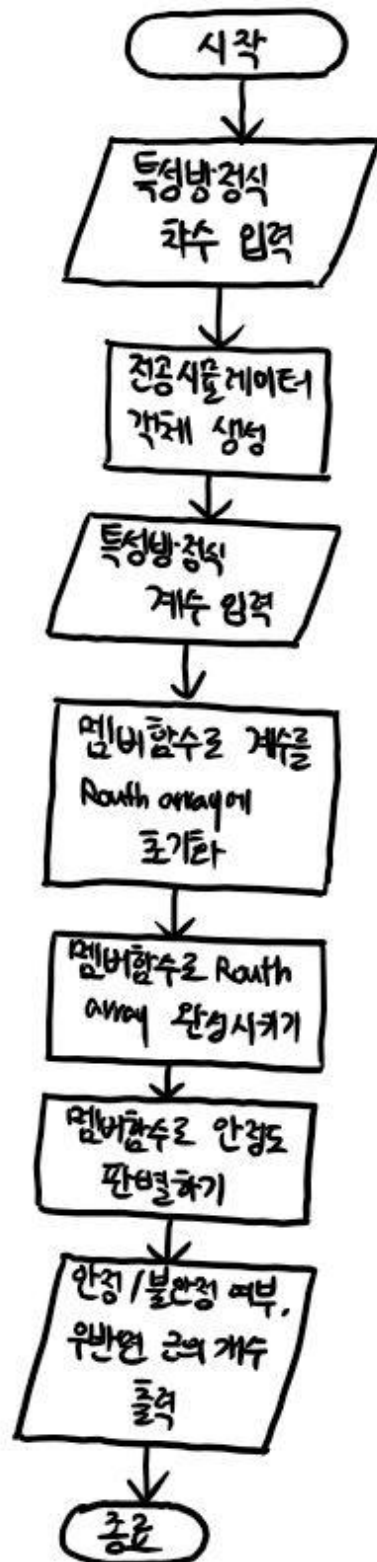
Routh array를 완성시키기 위해 멤버함수로 계산하는 함수를 만든다.

Routh array를 완성한 다음, Stability Criterion을 판별하기 위한 멤버 함수를 구현한다. 이 함수는 MFC에서 Stability Criterion이라는 Button을 생성하여 연결시킨다.

대략적인 순서도 - 영상처리



대략적인 순서도 - 전공 시뮬레이터



프로그래밍 코드 설명 - 영상처리

```
90 int imageData::imageProc(int Mode) {
91     if (Mode == 0) { //Inversion
92         for (rows = 0; rows < numRows; ++rows) {
93             for (cols = 0; cols < numcols; ++cols) {
94                 cout << (int)pixelValues[rows*numcols + cols] << ' ';
95                 pixelValues[rows*numcols + cols] = 255 - pixelValues[rows*numcols + cols];
96             }
97             cout << rows << endl;
98         }
99     }
100
101     else if (Mode == 1) { //Binarization
102         for (rows = 0; rows < numRows; ++rows) {
103             for (cols = 0; cols < numcols; ++cols) {
104                 cout << (int)pixelValues[rows*numcols + cols] << ' ';
105                 if (pixelValues[rows*numcols + cols] > 128){
106                     pixelValues[rows*numcols + cols] = 255;
107                 }
108                 else {
109                     pixelValues[rows*numcols + cols] = 0;
110                 }
111             }
112             cout << rows << endl;
113         }
114     }
```

먼저, 위의 imageData.cpp에서 멤버함수인 imageProc에 Inversion, Binarization이 존재하는데, 여기에다가 추가로 나만의 영상처리 기법을 구현하여 넣는다.

```
116     else if (Mode == 2) { //밝기 10씩 증가
117         for (rows = 0; rows < numRows; ++rows) {
118             for (cols = 0; cols < numcols; ++cols) {
119                 cout << (int)pixelValues[rows * numcols + cols] << ' ';
120                 if (pixelValues[rows * numcols + cols] <= 245)
121                 {
122                     pixelValues[rows * numcols + cols] = pixelValues[rows * numcols + cols] + 10;
123                 }
124                 else
125                 {
126                     pixelValues[rows * numcols + cols] = 255;
127                 }
128             }
129             cout << rows << endl;
130         }
131     }
```

Mode가 2인 경우 밝기가 10씩 증가하도록 구현하였다. 이때, pixelValues는 unsigned char이므로, 0~255까지의 값만 가질 수 있다. 246이상의 값들에 그냥 10을 더하게 되면 overflow가 생길 수 있으므로

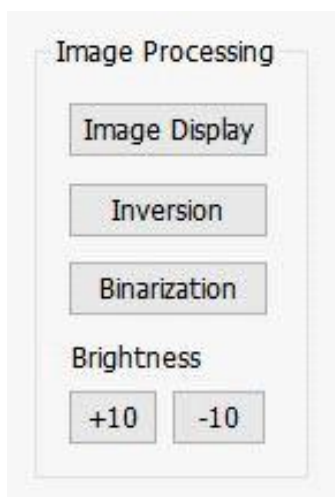
로, if문을 이용해 픽셀값이 245이하일 경우에는 그냥 10씩 더해주고, 246이상일 경우에는 픽셀값을 최댓값인 255가 되도록 설정해주었다.

```

133     else if (Mode == 3) { //밝기 10씩 감소
134         for (rows = 0; rows < numRows; ++rows) {
135             for (cols = 0; cols < numcols; ++cols) {
136                 cout << (int)pixelValues[rows * numcols + cols] << ' ';
137                 if (pixelValues[rows * numcols + cols] >=10)
138                 {
139                     pixelValues[rows * numcols + cols] = pixelValues[rows * numcols + cols] - 10;
140                 }
141                 else
142                 {
143                     pixelValues[rows * numcols + cols] = 0;
144                 }
145             }
146             cout << rows << endl;
147         }
148     }
149
150     else {
151         cout << "Wrong Mode" << endl;
152     }
153
154     pixelpProcessed = pixelValues; // pixelpProcessed can be allocated by new
155     return 0;
156 }

```

Mode가 3인 경우 밝기가 10씩 감소하도록 구현하였다. 이때, pixelValues는 unsigned char이므로, 0~255까지의 값만 가질 수 있다. 9이하의 값들에 그냥 10을 빼게 되면 underflow가 생길 수 있으므로, if문을 이용해 픽셀값이 10이상일 경우에는 그냥 10씩 빼주고, 9이하일 경우에는 픽셀값을 최솟값인 0이 되도록 설정해주었다.



그런 다음, MFC에서 만들어 둔 Button들에 각각의 기능을 할당하기 위

해 각 버튼마다 이벤트 처리기를 추가하여 MFCView.cpp파일에 각 Button별로 함수가 생성되게 만든다.

```
214 void CMFCView::OnBnClickedImageDisplay()
215 {
216     // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
217     AfxMessageBox(_T("ImageDisplay"));
218     CMFCDoc* pDoc = GetDocument();
219     plmg = pDoc->getImg();
220     imgCopy1 = new unsigned char[plmg->getHeight() * plmg->getWidth()];
221     plmg->getImage(imgCopy1); // get image data to display
222     Invalidate(true); // active WM_PAINT message
223 }
224
225 void CMFCView::OnBnClickedInversion()
226 {
227     // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
228     AfxMessageBox(_T("Inversion"));
229     CMFCDoc* pDoc = GetDocument();
230     plmg = pDoc->getImg();
231     imgCopy2 = new unsigned char[plmg->getHeight() * plmg->getWidth()];
232     plmg->imageProc(0);
233     plmg->getImage(imgCopy2); // get image data to display
234     Invalidate(true); // active WM_PAINT message
235 }
236
237
238 void CMFCView::OnBnClickedBinarization()
239 {
240     // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
241     AfxMessageBox(_T("Binarization"));
242     CMFCDoc* pDoc = GetDocument();
243     plmg = pDoc->getImg();
244     imgCopy3 = new unsigned char[plmg->getHeight() * plmg->getWidth()];
245     plmg->imageProc(1);
246     plmg->getImage(imgCopy3); // get image data to display
247     Invalidate(true); // active WM_PAINT message
248 }
```

```

251 void CMFCView::OnBnClickedBrightnessplus()
252 {
253     // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
254     AfxMessageBox(_T("Brightness +10"));
255     CMFCDoc* pDoc = GetDocument();
256     plmg = pDoc->getImg();
257     imgCopy4 = new unsigned char[plmg->getHeight() * plmg->getWidth()];
258     plmg->imageProc(2);
259     plmg->getImage(imgCopy4); // get image data to display
260     Invalidate(true); // active WM_PAINT message
261 }
262
263
264 void CMFCView::OnBnClickedBrightnessminus()
265 {
266     // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
267     AfxMessageBox(_T("Brightness -10"));
268     CMFCDoc* pDoc = GetDocument();
269     plmg = pDoc->getImg();
270     imgCopy5 = new unsigned char[plmg->getHeight() * plmg->getWidth()];
271     plmg->imageProc(3);
272     plmg->getImage(imgCopy5); // get image data to display
273     Invalidate(true); // active WM_PAINT message
274 }

```

MFCView.h파일에서 전처리기로 imageData.h를 include한 다음, 각 Button별로 생성된 함수에다가 imageData.cpp에서 구현한 영상처리 기법들을 넣어 Button과 영상처리 기법을 연결시켰다.

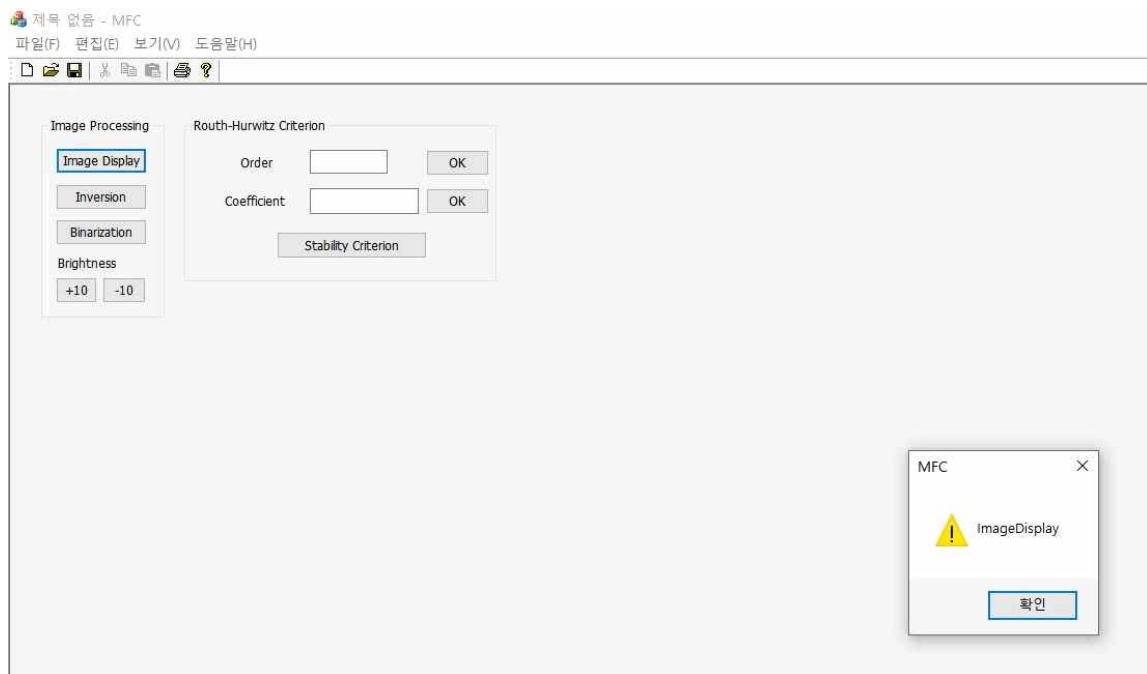
실험 결과 - 영상처리

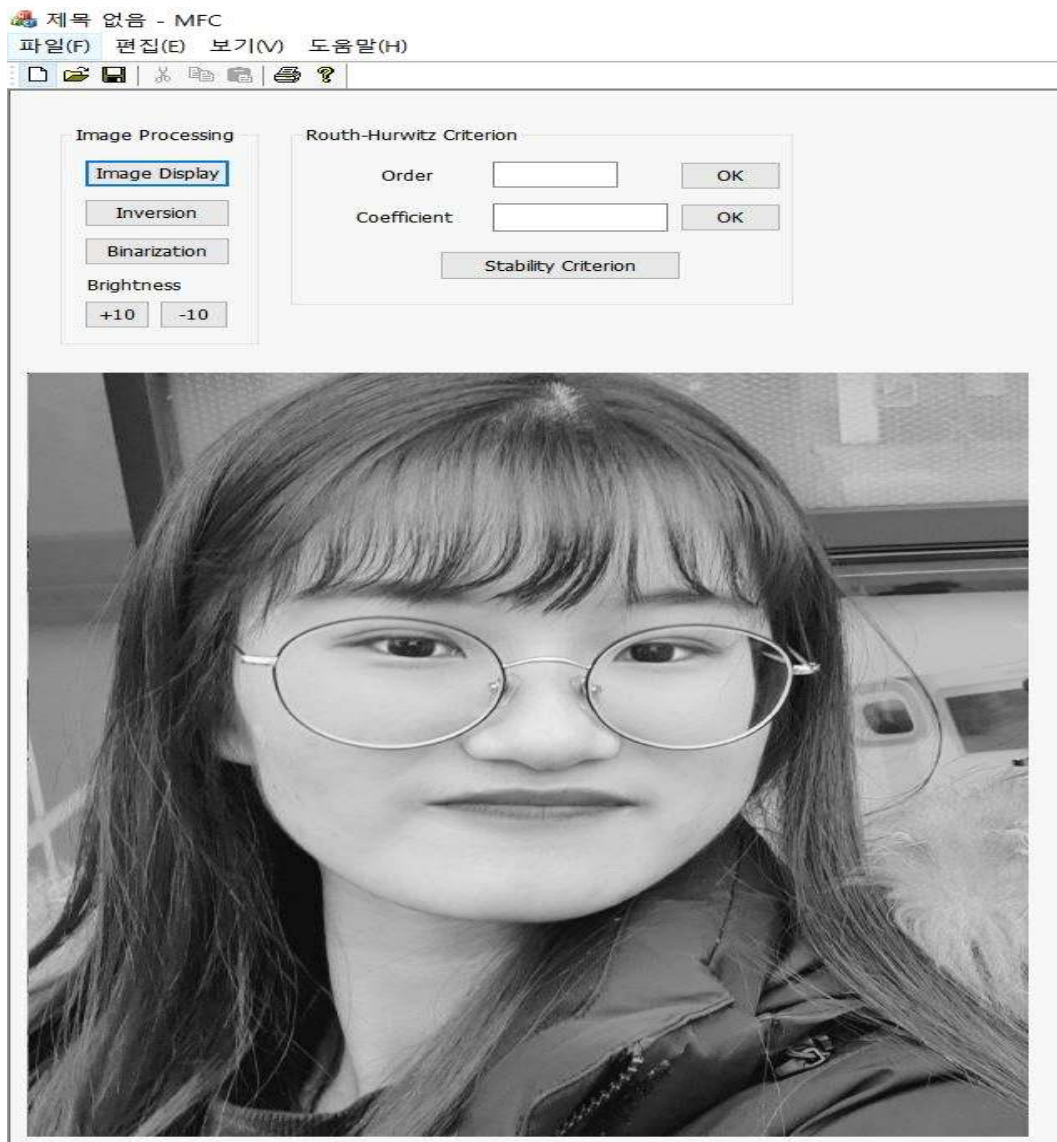
영상처리를 할 파일을 불러와서 특정 영상처리 기법을 사용하여 영상을 처리하고 난 다음, 다른 영상처리 기법을 사용하면 중첩이 되어 나타나게 하였다. 그래서 이를 영상의 밝기를 변환할 때 적용하거나 여러 가지 영상처리를 같이 처리하고 싶을 때 이를 적용한다.

Inversion과 Binarization등을 각각 따로 처리하고 싶을 경우에는 영상처리를 할 파일을 불러와서 Inversion으로 영상을 처리하고, 그 다음 영상처리를 할 파일을 다시 불러와서 Binarization으로 영상을 처리해야 된다.

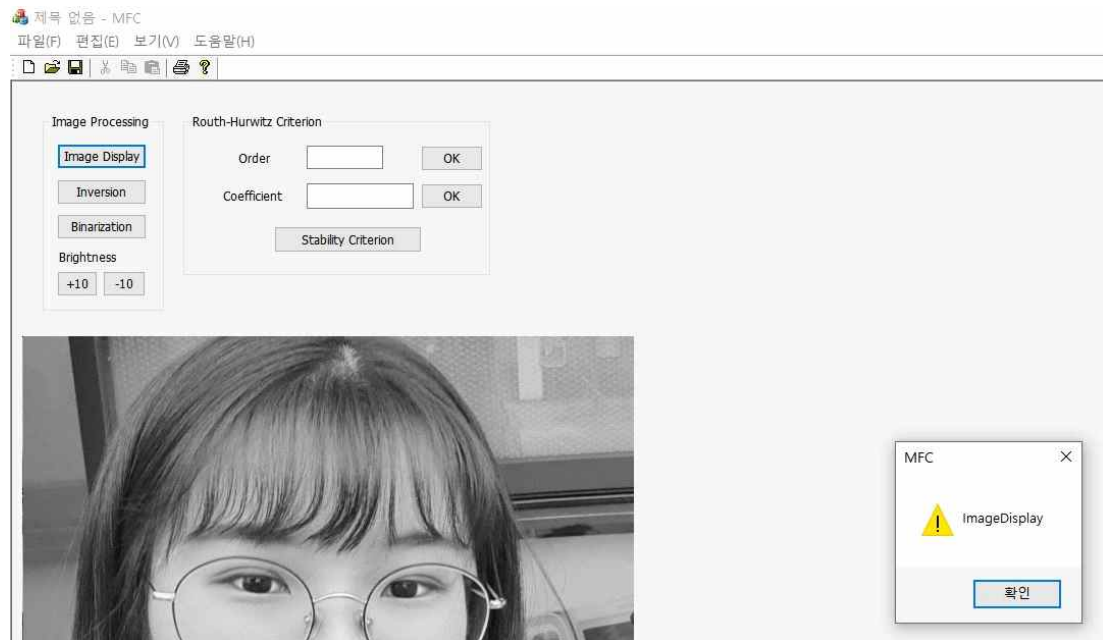
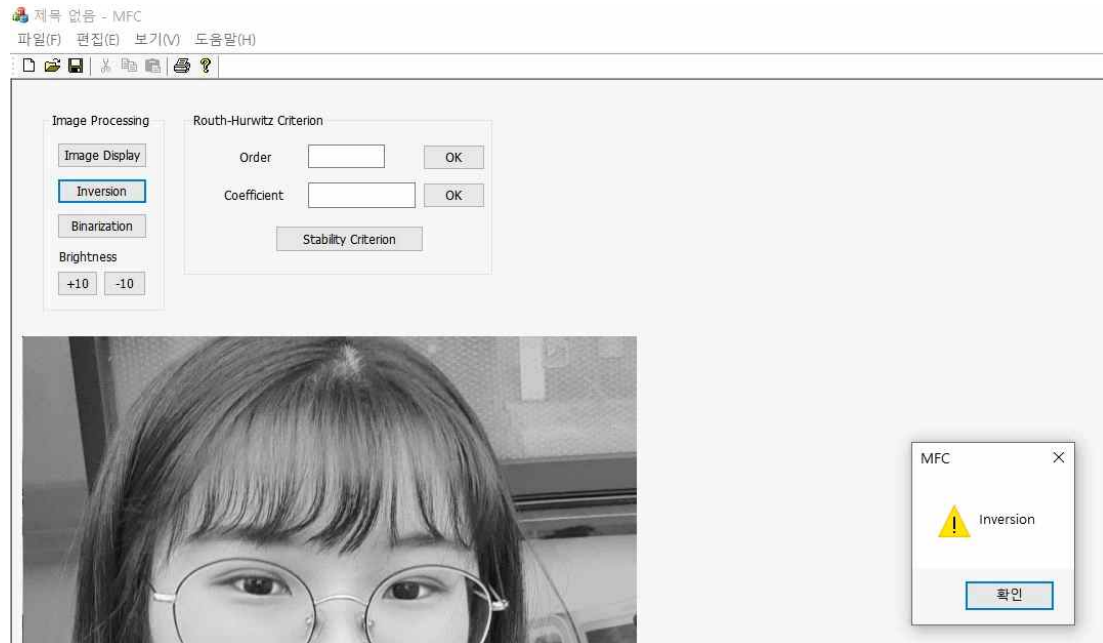
여기서 실험 결과는, 영상 밝기 변환을 제외하고는 영상 처리를 할 파일을 각각 불러와서 중첩이 되지 않은 결과이다.

1. 원본 영상을 불러와서 출력한 결과





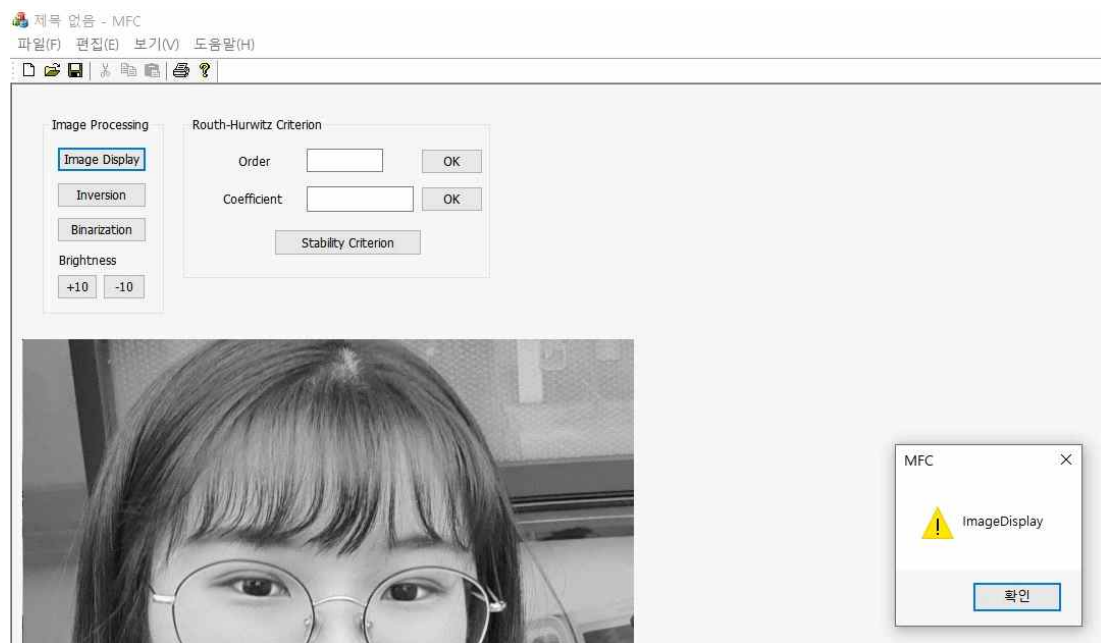
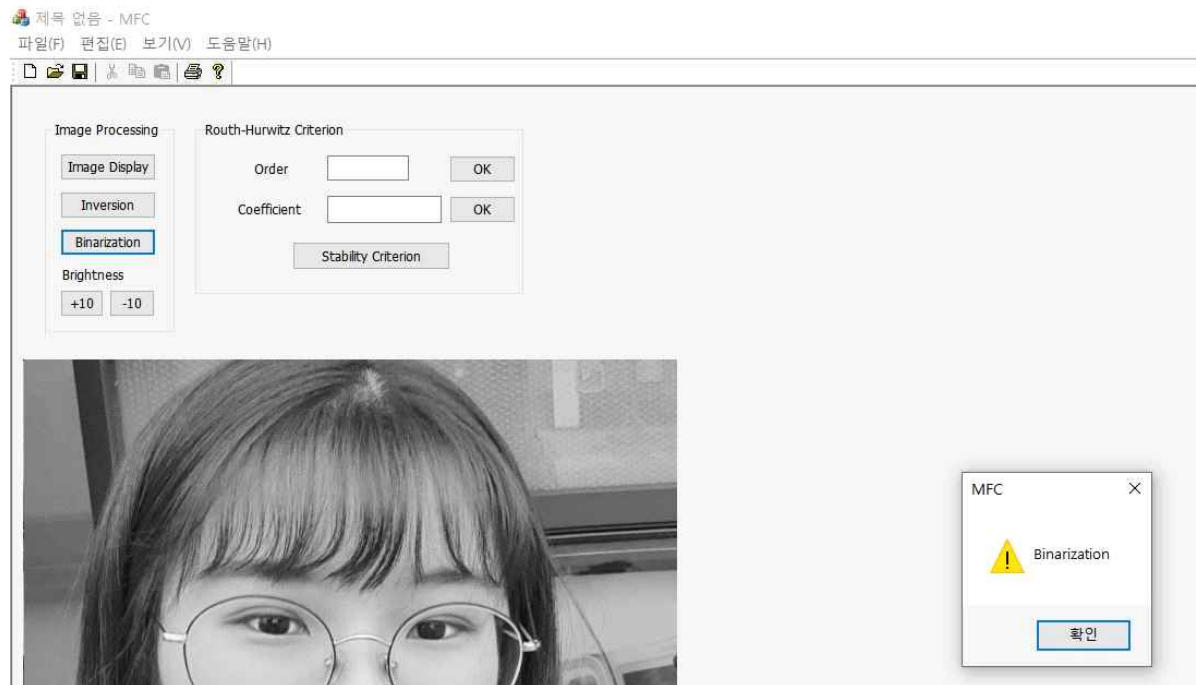
2. 원본 영상을 불러와서 Inversion을 실행한 다음 영상을 출력한 결과



Inversion Button을 누른 다음 Image Display를 눌러야 처리된 결과가 출력되도록 하였다.

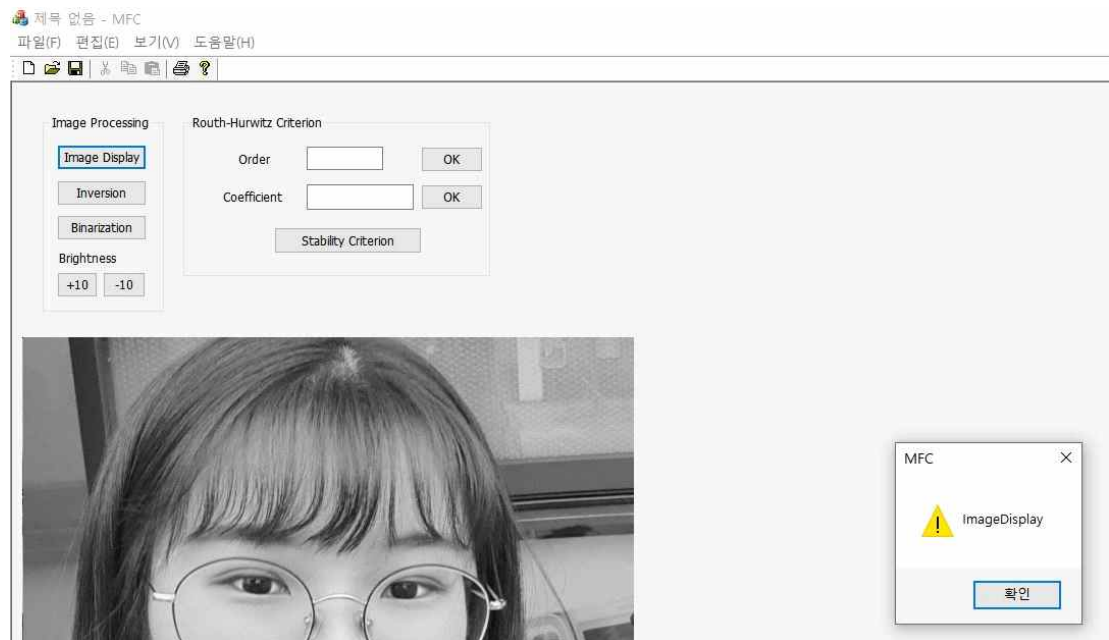
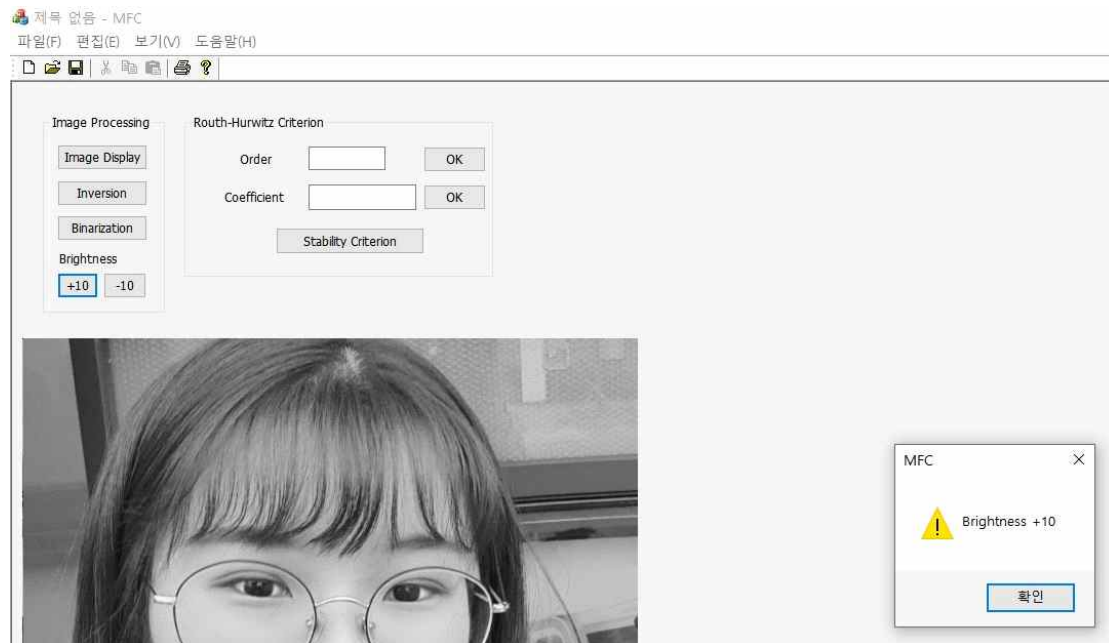


3. 원본 영상을 불러와서 Binarization을 실행한 다음 영상을 출력한 결과





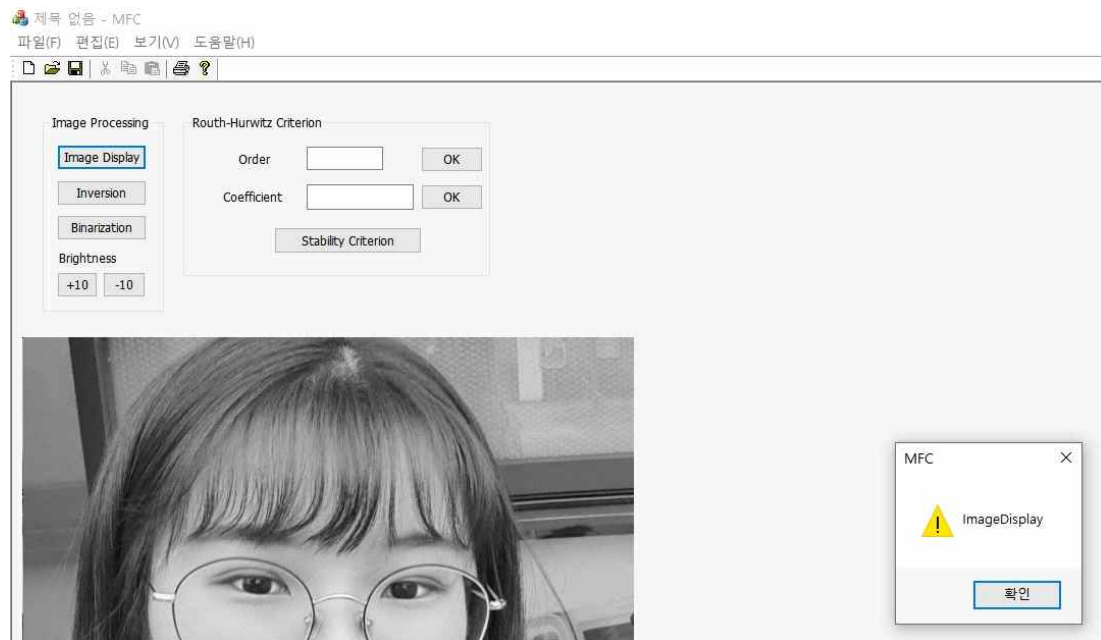
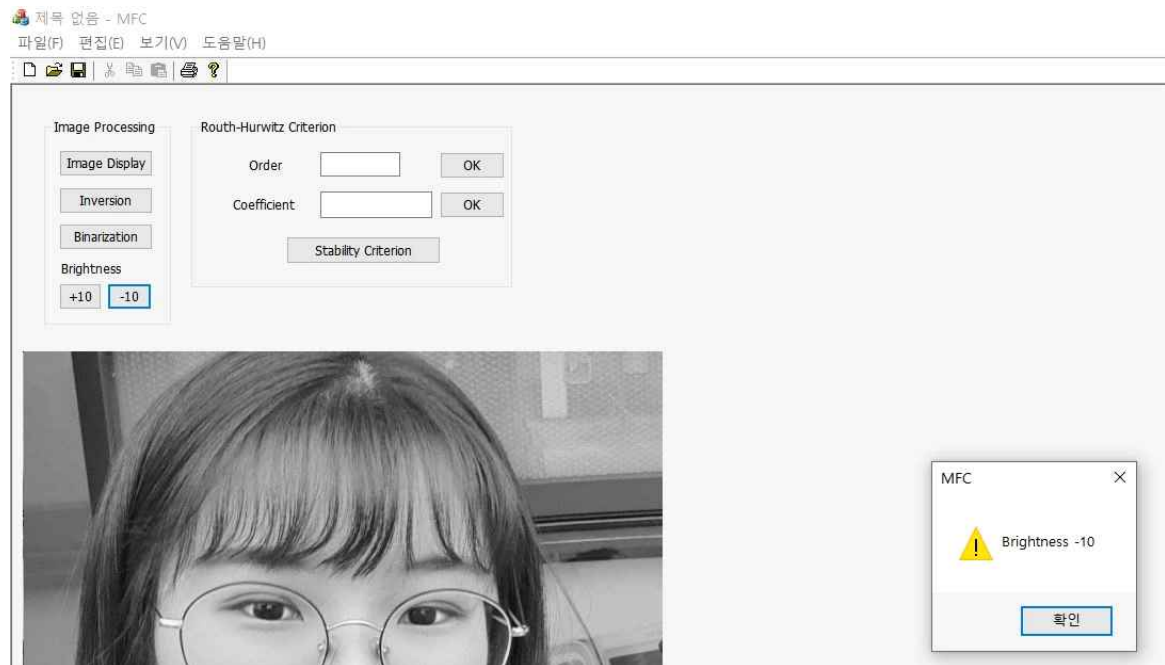
4. 원본 영상을 불러와서 Brightness +10을 다섯 번 실행한 다음 영상을 출력한 결과

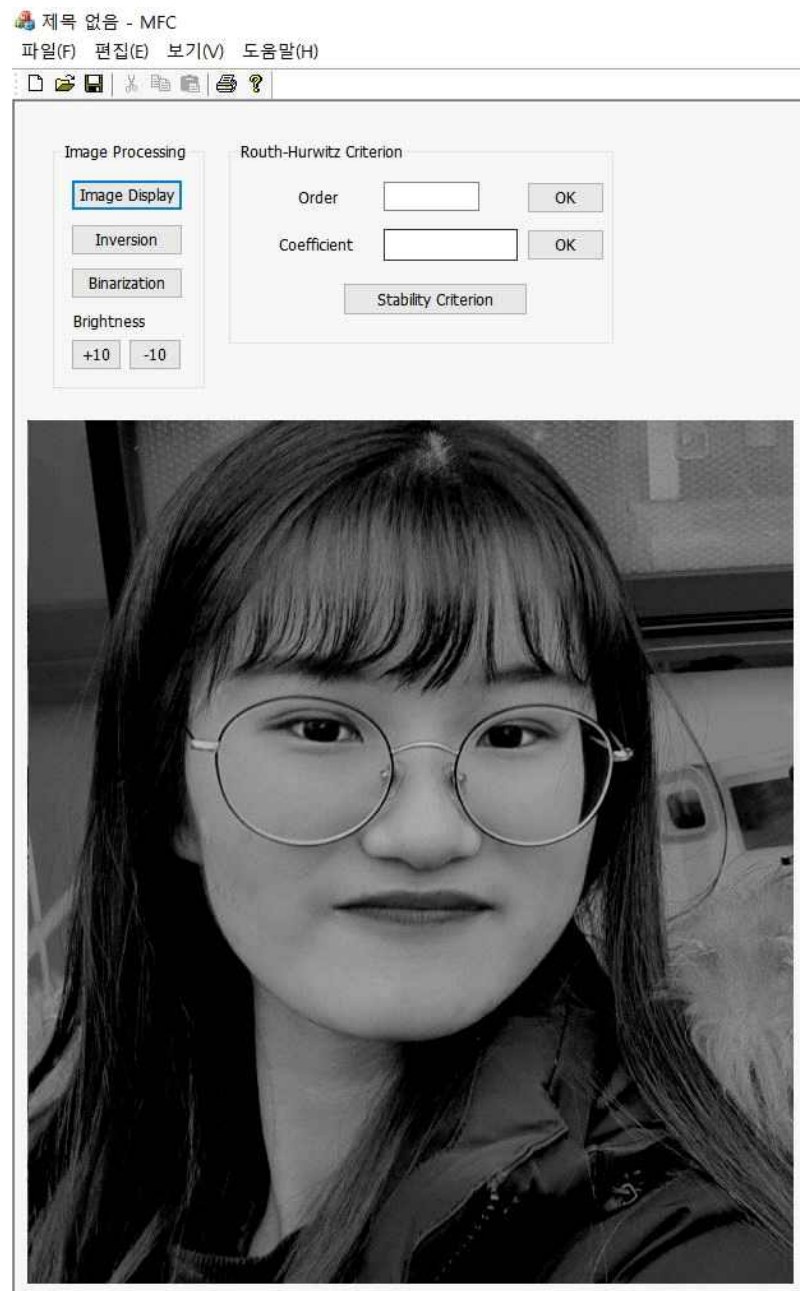




+10 Button을 다섯 번 클릭한 다음 Image Display를 해서 출력된 결과이다.

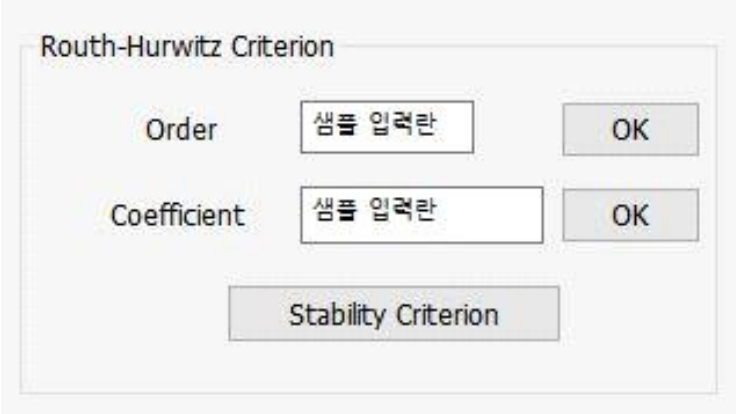
5. 원본 영상을 불러와서 Brightness -10을 다섯 번 실행한 다음 영상을 출력한 결과





-10 Button을 다섯 번 클릭한 다음 Image Display를 해서 출력된 결과이다.

프로그래밍 코드 설명 - 전공 시뮬레이터



The image shows a dialog box titled "Routh-Hurwitz Criterion". It contains two input fields, each labeled "샘플 입력란" (Sample Input Field). The first input field is preceded by the label "Order", and the second is preceded by "Coefficient". To the right of each input field is an "OK" button. Below these two input fields is a single button labeled "Stability Criterion".

먼저, MFC에서 입력 받을 값들은 Edit control과 Button을 이용해 받아오고, 안정도 판별은 Stability Criterion Button을 누르면 판별이 될 수 있도록 구상하였다.

```
1      #pragma once
2      #include <string>
3
4      class majorsimul
5      {
6      private:
7          int order; //특성방정식 차수
8          int* coefficient; //Routh array의 값 저장
9          int col1;
10         int col2;
11         int col; //Routh array의 column
12
13     public:
14         majorsimul(); //생성자 함수
15         void setmajorsimul(int ord);
16         void setfirstarray(int coef, int rows, int cols);
17         void routharray();
18         int criterion();
19         ~majorsimul(); //소멸자 함수
20     };
```

위를 바탕으로 Order와 Coefficient를 입력값으로 받으면 , 전공 시뮬레이터 클래스 선언부에서 setmajorsimul()함수를 이용해 멤버변수들을 초기화하고, coefficient라는 이름을 가진 포인터 변수에다가 동적 메모리를 할당한 공간의 주소를 return하게된다.


```

1  #include "pch.h"
2  #include "majorsimul.h"
3  #include <iostream>
4  #include <string>
5
6  using namespace std;
7
8  majorsimul::majorsimul()
9  {
10     order = 0;
11     coefficient = NULL;
12     col1 = 0;
13     col2 = 0;
14     col = 0;
15 }
16
17 void majorsimul::setmajorsimul(int ord)
18 {
19     order = ord;
20     col1 = (order + 1) / 2;
21     col2 = (order + 1) % 2;
22     col = col1 + col2;
23     coefficient = new int[(order+1)*col];
24 }
25
26 void majorsimul::setfirstarray(int coef, int rows, int cols)
27 {
28     coefficient[rows * col + cols] = coef;
29 }
30
31
32 int majorsimul::criterion()
33 {
34     int criterion = 0;
35     for (int i = 1; i < order+1; i++)
36     {
37         if ((coefficient[(i - 1) * col] * coefficient[i * col]) < 0)
38             criterion = criterion + 1;
39     }
40 }
41
42 void majorsimul::routharray()
43 {
44     for (int i = 2; i <= order; i++)
45     {
46         coefficient[i * col + col - 1] = 0;
47     }
48     for (int i = 2; i <= order; i++)
49     {
50         for (int j = 0; j < col - 1; j++)
51         {
52             coefficient[i * col + j] = (-1 / (coefficient[(i - 1) * col])) * ((coefficient[(i - 2) * col] * coefficient[(i - 1) * col + (j + 1)]) - (coefficient[(i - 1) * col] * coefficient[(i - 2) * col + (j + 1)]));
53         }
54     }
55 }
56
57
58
59
60 majorsimul::~majorsimul()
61 {
62     delete[] coefficient;
63 }

```

위는 전공 시뮬레이터 클래스의 구현부이다.

생성자 함수는 사용하지 않을 예정이지만, 무조건 있어야 되므로 모든 멤버변수를 0이나 NULL로 초기화 하도록 하였다.

setmajorsimul()함수를 이용해 모든 멤버변수를 초기화한다. 이때, coefficient라는 포인터 멤버변수는 동적메모리 할당으로 주소를 가리키고, 이는 routh array의 값을 담는 역할을 한다.

setfirstarray()함수를 이용해 MFC의 Coefficient에 있는 edit control로부터 받은 입력값으로 특성방정식의 각 차수항의 계수를 routh array 공간에 할당한다.

criterion()함수는 routh array를 모두 완성시킨 다음, routh array의 첫 번째 열만을 가지고 부호변화를 판단하는 역할을 한다.

routharray()함수는 setfirstarray()함수를 사용해 특성방정식의 각 차수항의 계수를 routh array공간에 할당한 것을 바탕으로 계산하여 routh array를 완성하는 역할을 한다.

소멸자 함수는 동적메모리를 다 사용하고 난 다음, 동적메모리 할당을 해지하기위한 역할을 한다.

먼저 이렇게 전공 시뮬레이터 클래스를 다 만들고, 그 다음 MFCview.cpp와 MFCview.h에 전공 시뮬레이터 클래스의 헤더부분을 include한 다음, MFC와 클래스 멤버함수들을 연결할 수 있도록 만든다.


```

277 void CMFCView::OnBnClickedOrder()
278 {
279     // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
280     UpdateData(TRUE); // 에디트 박스 값을 불러오기
281     CString strName; // 에디트 값이 저장될 변수선언
282     GetDlgItemText(IDC_EDIT1, strName);
283     o = _ttoi(strName);
284     routh.setmajorsimul(o);
285     CString output;
286     output.Format(L"특성방정식의 차수 : %d", o);
287     AfxMessageBox(output);
288 }

```

MFC에서 Order의 Edit control에 값을 입력하고 OK Button을 누르면 구동되는 멤버함수이다. o라는 정수형 변수와 routh라는 전공 시뮬레이터 클래스로 만든 객체는 MFCview.h파일에 private 멤버변수로 선언해 두었다.

_ttoi()함수를 사용하여 CString값을 정수형 변수 o에 할당해주고, 이를 이용해 전공 시뮬레이터 클래스의 멤버변수들을 모두 초기화한다.

285~287행의 명령은 OK Button을 누르는 순간 내가 입력한 값을 제대로 받았는지 확인해주는 용도로 만들었다.

```

291 void CMFCView::OnBnClickedCoefficient()
292 {
293     // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
294     UpdateData(TRUE); // 에디트 박스 값을 불러오기
295     CString strName2; // 에디트 값이 저장될 변수선언
296     GetDlgItemText(IDC_EDIT2, strName2);
297
298     CString strTok;
299     int sepCount = GetFindCharCount(strName2, ',', ' ');
300     //CString* temp = new CString[sepCount + 1];
301     //int* temp2 = new int[sepCount + 1];
302     temp = new CString[sepCount + 1];
303     temp2 = new int[sepCount + 1];
304
305     int cnt = 0;
306     int i = 0;
307     while (AfxExtractSubString(strTok, strName2, cnt, ','))
308     {
309         temp[cnt++] = strTok;
310         temp2[i] = _ttoi(temp[i]);
311         CString output2;
312         output2.Format(L"%d차항 계수 : %d", o - i, temp2[i]);
313         AfxMessageBox(output2);
314         i = i + 1;
315     }
316
317     int col1 = (o + 1) / 2;
318     int col2 = (o + 1) % 2;
319     int col = col1 + col2;
320     int p = 0;
321
322     for (int i=0; i<col; i++)
323     {
324         for (int j = 0; j < 2; j++)
325         {
326             routh.setfirstarray(temp2[p], j, i);
327             p++;
328         }
329     }
330     if (col != col1)
331     {
332         routh.setfirstarray(0, 1, col - 1);
333     }

```

MFC에서 Coefficient의 Edit control에 값을 입력하고 OK Button을 누르면 구동되는 멤버함수이다.

만약 특성방정식이 2차방정식이고 2차항의 계수가 1, 1차항의 계수가 2, 0차항의 계수가 0인 경우에는 1,2,0 으로 입력하고 OK Button을 누르면 ‘,’를 기준으로 값을 구분하여 입력받도록 하였다. 그러기 위해 299행의 GetFindCharCount()함수를 전역함수로 만들어 사용하였다.

```

335 int CMFCView::GetFindCharCount(CString parm_string, char parm_find_char)
336 {
337     int length = parm_string.GetLength();
338     int find_count = 0;
339     for (int i = 0; i < length; i++)
340     {
341         if (parm_string[i] == parm_find_char)
342         {
343             find_count++;
344         }
345     }
346     return find_count;
347 }

```

311~313행의 명령은 OK Button을 누르는 순간 내가 입력한 값들을 제대로 받아서 값을 제대로 구분하였는지 확인해주는 용도로 만들었다.

317~332행의 명령은 특성방정식의 각 차수항의 계수를 입력받은 것을 Routh array에다 값을 할당해주는 역할을 한다.

```

350 void CMFCView::OnBnClickedCriterion()
351 {
352     // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
353     routh.routharray();
354
355     int roots;
356     roots = routh.criterion();
357
358     if (roots == 0)
359     {
360         AfxMessageBox(_T("우반면에 위치한 근의 개수 : 0개, 시스템 안정/불안정 여부 : 안정.));
361     }
362     else if (roots > 0)
363     {
364         CString output3;
365         output3.Format(L"우반면에 위치한 근의 개수 : %d개, 시스템 안정/불안정 여부 : 불안정.", roots);
366         AfxMessageBox(output3);
367     }
368 }

```

MFC에서 Stability Criterion Button을 누르면 구동되는 멤버함수이다. Button을 누르는 순간 routharray()함수가 구동되어 Routh array를 완성시킨다.

roots라는 정수형 변수를 선언한 다음, criterion()함수를 사용해 Routh array의 첫 번째 열에 부호변화가 있는 개수만큼 값을 반환하여 roots 변수에 할당한다.

이를 바탕으로 if else문과 AfxMessageBox를 사용하여 안정도를 판별한 결과를 화면으로 출력해준다.

실험 결과 - 전공 시뮬레이터

1. 안정한 경우

- 특성방정식의 근이 $-1, -2$ 인 경우

$$(s+1)(s+2) = s^2 + 3s + 2$$

Routh array

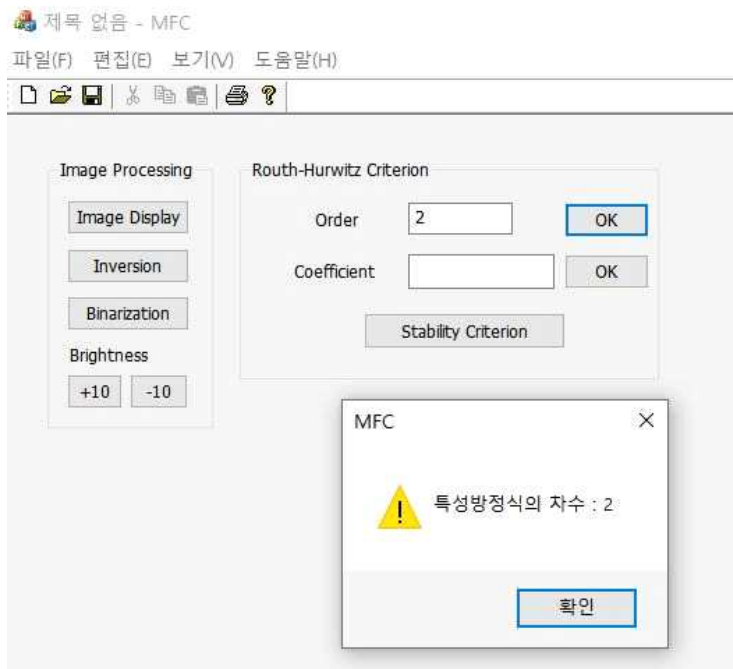
s^2	1	2
s^1	3	0
s^0	2	0

$$\frac{-1}{3}(-6) = 2$$

부호변화 $\times \Rightarrow$ 시스템은 안정

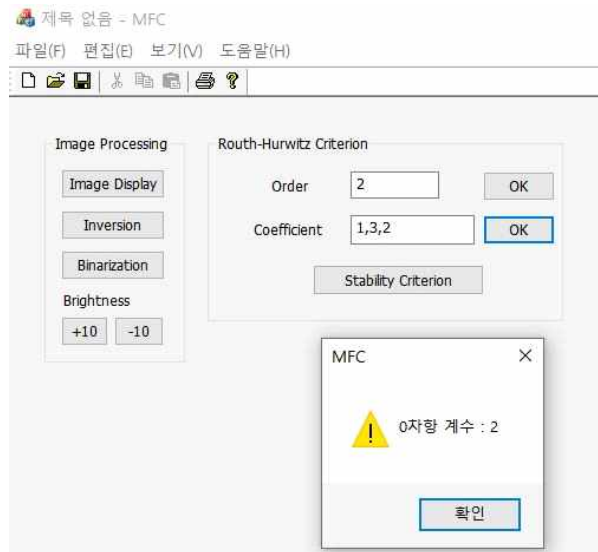
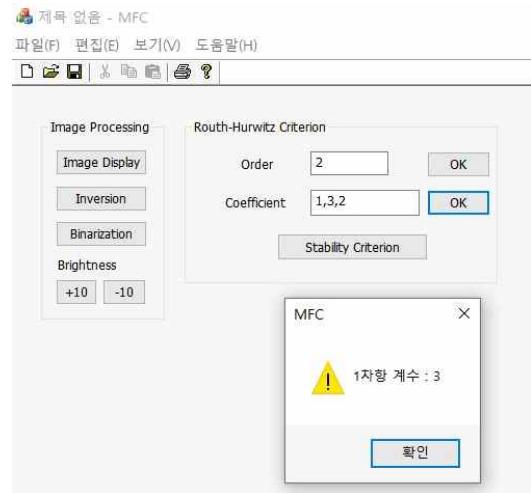
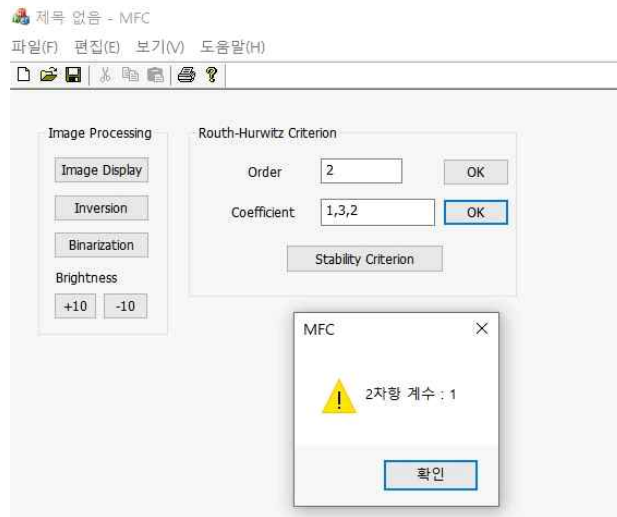
우반면에 있는 근의 개수 : 0개

위의 경우를 구현한 전공 시뮬레이터에 적용해보았다.

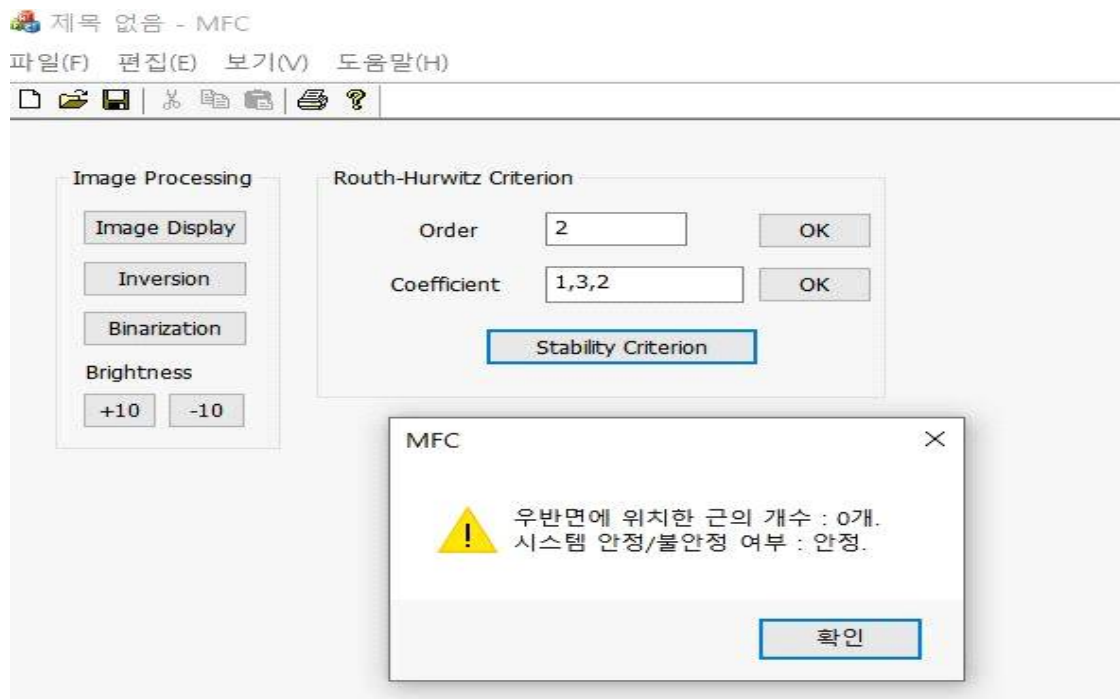


먼저 특성방정식의 차수를 Edit control에 입력하고 OK Button을 누른

결과이다.



Coefficient의 Edit control에 '1,3,2'를 입력하고 OK Button을 누른 결과이다.



Stability Criterion Button을 누른 결과이다.

2. 불안정한 경우

- 특성방정식의 근이 1, -2인 경우

$$(s-1)(s+2) = s^2 + s - 2$$

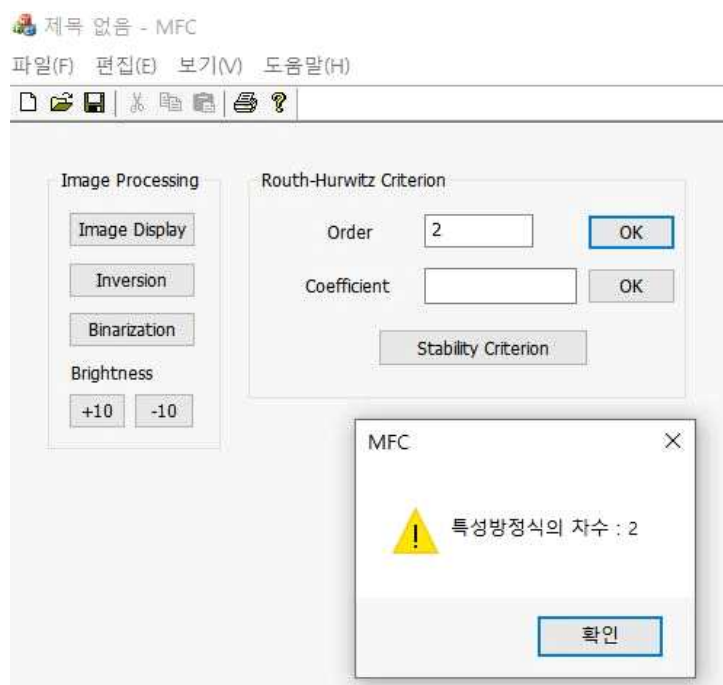
Routh array

s^2	1	-2
s^1	1	0
s^0	-2	0

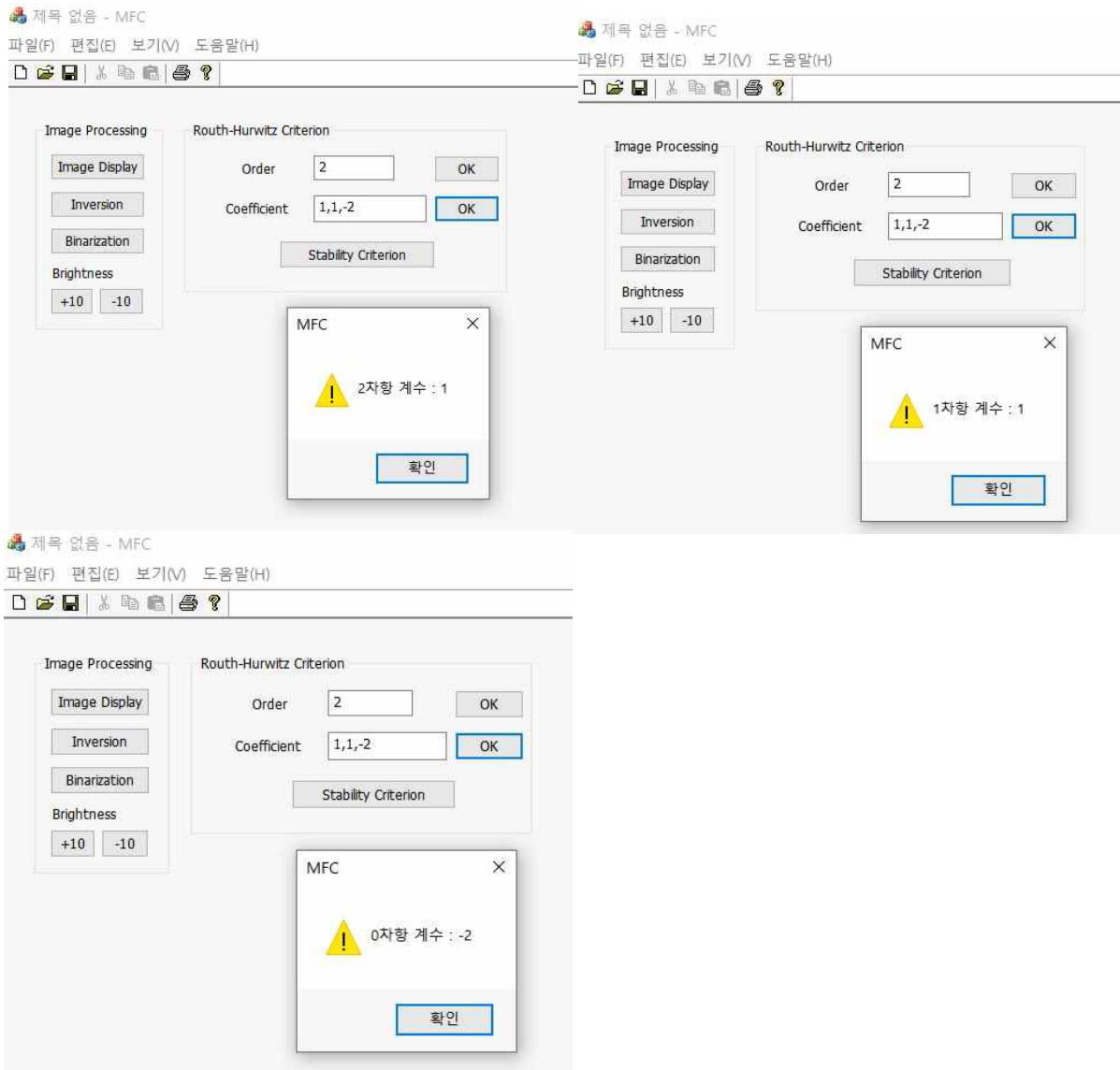
부호변화 0 \Rightarrow 시스템은 불안정

우반면에 있는 근의 개수 = 부호변화 횟수 : 1개

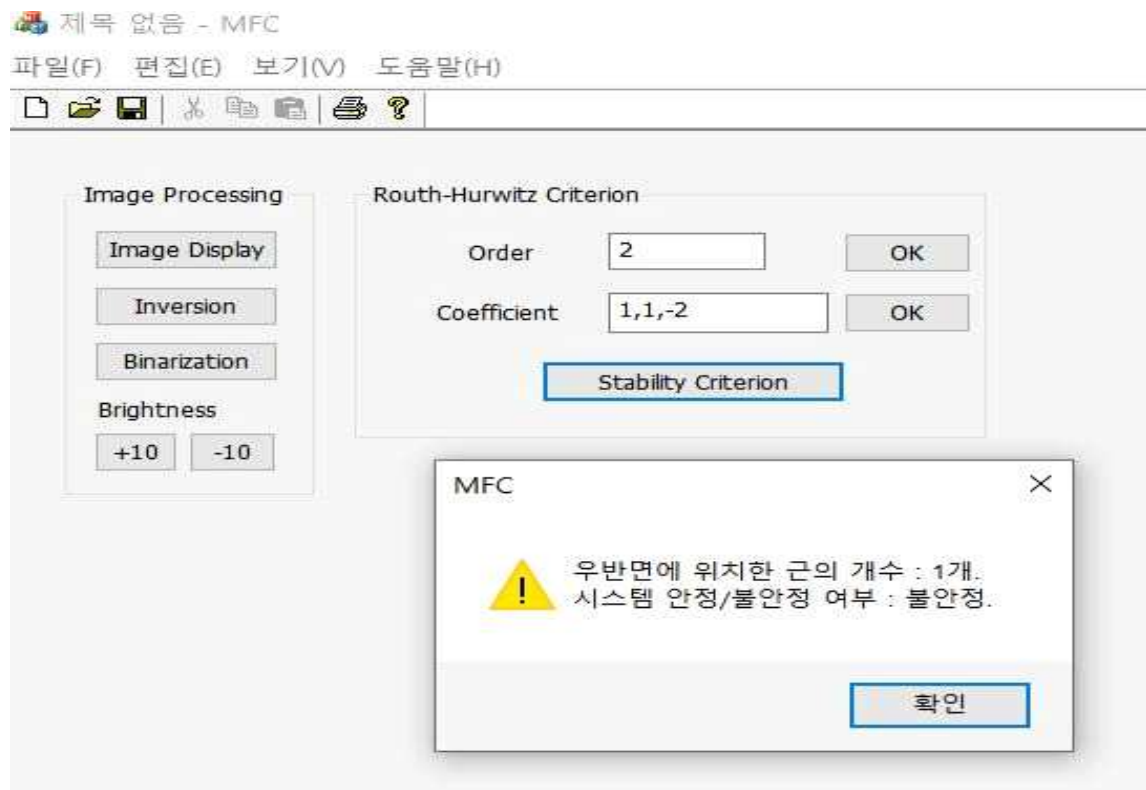
위의 경우를 구현한 전공 시뮬레이터에 적용해보았다.



먼저 특성방정식의 차수를 Edit control에 입력하고 OK Button을 누른 결과이다.



Coefficient의 Edit control에 '1,1,-2'를 입력하고 OK Button을 누른 결과이다.



Stability Criterion Button을 누른 결과이다.

3. 결론

학기중 영상처리를 클래스로 구현했던 것에 추가로 나만의 영상처리 기법을 추가하고, 이를 MFC와 연동해보았다.

영상처리할 파일을 불러와 특정 영상처리 기법을 수행할 Button을 누르면 영상처리가 되고, Image Display Button을 누르면 영상처리된 결과 영상이 출력되는 것을 확인할 수 있었다.

추가적으로, 나만의 전공 시뮬레이터로 이번 학기에 수강한 과목인 자동 제어에서 배운 Routh-Hurwitz array에 대한 클래스를 만들어 MFC와 연동해 전공 시뮬레이터를 만들어 보았다.

Edit control box에 값을 넣고 OK Button을 누르면 값이 입력되고, 내가 입력한 값이 제대로 입력되었는지 확인할 수 있도록 messagebox로 내가 입력한 값이 출력이 된다.

Routh-Hurwitz array 클래스를 이용해 Routh-Hurwitz array를 완성하고, Stability criterion Button을 누르면 안정도가 판정된 결과가 출력되는 것을 확인할 수 있었다.

이번 텀프로젝트를 통해 특정 클래스를 처음부터 직접 구상하고 구현해 봄으로써 클래스를 만드는 법을 한 번 더 실습할 수 있었다.

또한, MFC를 처음 사용해 보는 것이어서 처음 3일간은 제대로 구동이 되지 않고 오류만 떠서 힘들었는데, 스스로 인터넷에 검색도 해보고 교수님께 질문을 해보면서 하나씩하나씩 해결해나감으로써 혼자서는 구현하기 불가능할 것 같았던 것을 구현하게 되어 한단계 더 성장한 느낌이 들었다. MFC가 많이 어렵긴 하지만, 계속 하다보면 더 잘할 수 있을 것 같고, 지금 배운 MFC 개념을 바탕으로 다른 프로그래밍언어에서도 한 번 적용해보면 좋을 것 같다는 생각이 든다.

4. 부록

프로그램 사용매뉴얼 - 영상처리

1. 영상처리할 파일을 불러온다
2. Image Display Button을 눌러 불러온 영상 파일을 출력하여 확인한다.
3. 여러 가지 영상처리 기법들
(Inversion/Binarization/Brightness+10/Brightness-10)중 원하는 영상처리 Button을 누른 다음, Image Display Button을 누르면 영상처리된 결과가 출력된다.
4. 3에서 영상처리를 한 결과에다가 추가로 영상처리를 하고싶은 경우 3의 방법처럼 다시 실행하면 중첩이 되어 영상처리가 된다.
5. 3에서 영상처리를 한 결과와 별개로 1에서 불러온 원본 영상에다 새로운 영상처리를 하고싶은 경우 영상처리할 파일을 다시 불러와서 3의 방법처럼 실행하면 원하는 영상처리 결과가 나오게 된다.

프로그램 사용매뉴얼 - 전공 시뮬레이터

1. Routh-Hurwitz Stability Criterion을 적용할 특성방정식의 차수를 첫 번째 Edit control box에 입력하고 OK Button을 누른다.
예) 특성방정식의 차수가 3차인 경우 Edit control box에 '3'을 입력한다.
2. 위의 특성방정식의 각 차수항 계수를 두 번째 Edit control box에 입력하고 OK Button을 누른다.
예) 특성방정식의 차수가 3차이고, 3차항의 계수가 1, 2차항의 계수가 2, 1차항의 계수가 3, 0차항의 계수가 4인 경우 Edit control box에 '1,2,3,4'를 입력한다.
3. Stability Criterion Button을 누르면 안정/불안정 여부와 우반면에 위치한 근의 개수가 출력된다.(안정도가 판별이 된다.)