

컴퓨터공학종합설계

최종보고서

팀명 : Y2K - 004 분반(이문규 교수님)

프로젝트명 : 악보를 인식하는 기타 연습 어플리케이션

"Jimmy"

팀원

12101520 유석엽 (lightlyu@gmail.com)

12121458 김주원 (wndnjs1769@naver.com)

12121665 윤태원 (nyuan01@gmail.com)

1. 목차

1. 목차
2. 요약
3. 주제 정의
 - 3.1. 최종 목표에 대한 간략한 설명
4. 배경
 - 4.1. 주제 선정 동기
 - 4.2. 이 프로젝트 완료로부터 기대되는 효과
 - 4.3. 관련분야 Survey
 - 4.4. 유사 프로젝트 검색 및 해당 프로젝트 결과물들의 문제점
 - 4.5. 기존 프로젝트와 제안 내용과의 차이 및 기존 문제 해결 방안
5. 개발 내용
 - 5.1. 개발 플랫폼
 - 5.2. 상세 요구사항
 - 5.2.1. 기능적 요구사항
 - 5.2.2. 비기능적 요구사항
 - 5.3. 상세 구조
6. 개발 방법
 - 6.1. 사용한 Library, API
 - 6.1.1. Unity3d Engine
 - 6.1.2. UGUI
 - 6.1.3. Fast Furier Transform
 - 6.1.4. OpenCV for Unity
 - 6.2. 직접 개발한 내용
 - 6.2.1. 악보 변환
 - 6.2.2. 소리 인식
 - 6.2.3. UI
 - 6.3. 개발 일정 및 팀원간 역할 분담
 - 6.3.1. 간트 차트
 - 6.3.2. 프로젝트 마일스톤

7. 개발 결과

7.1. 최종 결과물에 대한 스크린샷

7.1.1. 악보 변환

7.1.2. 소리 인식

7.1.3. UI

7.2. 요구사항에 대한 달성도

7.3. 데모 동영상 링크

7.4. 테스트 결과

8. 결론

8.1. 추가로 설명할 사항

8.1.1. Template matching

8.1.2. 코드의 종류

8.2. 추가 진행 및 개선 방안

8.3. 프로젝트를 진행하면서 느낀 점

8.4. 다음학기 수강생들을 위해 제안할 점

9. 참고 문헌

2. 요약

본 팀은 컴퓨터공학과 3 명으로 구성되어 있으며 다음과 같은 프로젝트를 제안한다. 총 기간 15 주, 개발기간 12 주의 기간동안 사용자가 원하는 노래의 악보를 분석해 사용자의 연주를 실시간으로 평가하는 모바일 어플리케이션을 제작하였다. 그 동안의 어플리케이션들은 기능이 한정적이며 정확성이 낮은 수준이다. 따라서 이를 개선하고 보다 나은 기능을 가진 어플리케이션을 제작하는 것이 본 프로젝트의 목적이다. 개발 기간동안 악보를 분석하여 코드를 추출해 내는 데 성공하였고, 정확도 90% 이상을 보장한다. 또한 사용자의 연주를 분석하여 연주된 코드를 인식하였고, 인식률은 80% 이상 정확하게 인식하였다. 위의 두 기능을 조합하여 사용자가 기타를 연습할 수 있는 UI 를 제공한다.

3. 주제 정의

3.1. 최종 목표(기대 결과물)에 대한 간략한 설명

기타는 우리가 가장 손쉽게 접할 수 있는 악기이다. 많은 사람들이 어디서든 연주할 수 있는 기타를 배우고 싶어한다. 하지만 악기 하나를 자유자재로 다루고 연주하는 것은 쉬운 일이 아니며, 따로 공부를 하지 않으면 악보를 제대로 보는 것도 쉽지 않다. 그래서 한 곡을 완곡 하지도 못하고 배우기를 포기하는 사람들이 많다. 그래서 우리는 악보를 대신 읽어주고 기타 연습을 도와주는 어플리케이션을 개발하기로 하였다.

이용자가 연주하기 원하는 악보 이미지를 입력 받아 영상처리 기술을 통해 악보를 분석한다. 분석한 정보를 토대로 박자, 마디, 음정, 코드 등 연주에 필요한 정보를 추출해 낸다.

마이크로 사용자의 연주를 입력 받아 고속 푸리에 변환(FFT)을 통해 주파수 별로 신호 크기를 분리하여 연주하는 음을 분석하고, 연주하는 코드를 알아낸다. 두 정보를 실시간으로 비교하며 연주를 실시간으로 평가하는 어플리케이션을 제작한다.

4. 배경

4.1. 주제 선정 동기

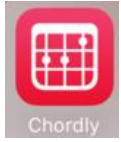


첫 번째로, 자신이 연주하고 싶은 곡의 악보가 있어도 기타 코드를 알아내기는 쉽지 않다. 인터넷을 통해 구할 수 있는 악보들 중에는 코드가 적혀 있지 않은 악보가 많다. 하지만 음악적 지식이 많지 않은 초보 사용자들이 이런 악보를 보고 코드를 알아내는 것은 불가능에 가깝다. 본 프로젝트는 악보 이미지를 입력 받아 해당 부분에 연주해야 할 코드를 분석하여 사용자에게 알려줌으로써 음악적 지식이 부족한 사용자들이 무리 없이 기타 연습을 할 수 있도록 돕는다. 다음으로, 자신이 연주하는 기타의 소리를 직접 평가하는 것이 힘들다. 초보 사용자들은 기타 줄 6 개를 정확히 잘 튕겨 내기가 힘든데, 연주가 익숙하지 않은 초보 연주자들은 코드 운지에 신경 쓰는 것만으로도 집중을 해야 한다. 그래서 모든 줄의 소리가 잘 나는지 확인하기 힘들다. 사용자들은 본 프로그램을 통해 각 줄의 소리가 잘 나는지 확인하며 보다 정확한 코드 연주를 연습할 수 있다.

4.2. 이 프로젝트 완료로부터 기대되는 효과

현재의 기타 관련 어플리케이션 시장에는 튜닝을 도와주는 어플리케이션이 대부분이고 그 외에는 시뮬레이션, 가이드 어플리케이션 등 다양한 수준의 어플리케이션이 출시되어 있다. 시뮬레이션은 화면을 눌러 기타소리를 들어보고 어떤 소리가 나와 있는지 확인할 수 있는 기능이 있다. 가이드 어플리케이션은 코드 별 운지법, 주법 등을 이미지와 텍스트로 나타내어 주는 기능뿐이다. 가장 다운로드 수가 많은 진보된 어플리케이션은 내장된 데이터베이스에 있는 곡들을 코드 정보에 따라 연주하면 기타소리를 인식하여 평가를 해 준다. 하지만 줄 하나를 연주하지 않아도 맞았다고 판정을 하고, 한 음정도의 오차를 허용하는 등 인식률이 완벽하지 않고, 데이터베이스에 있는 악보들만 따라 연주하는 것이 전부이다. 본 프로젝트는 사용자가 원하는 악보를 분석하여 해당 곡에 대한 서비스를 제공하여 사용자의 만족도를 높일 수 있을 것이다. 또한 소리 인식 정확도를 높여 반음정도의 오차만 허용하는 정밀도를 구현하여 사용자의 수준을 높이는 데 기여할 수 있다. 따라서 본 프로젝트가 완성될 경우 기본적으로 튜닝기능을 완전히 포함하며 연습 프로그램 자체의 성능을 기존의 어플리케이션들보다 개선하여 사용자에게 좋은 서비스를 제공하게 될 것이다. 또한 사용자가 원하는 악보들에 대해 서비스를 제공함으로써 타 어플리케이션으로는 연습하지 못하는 곡들까지 연습할 수 있게 한다. 나아가 악보 업체와 연계하여 악보

데이터베이스를 구축하고 결제 시스템을 추가할 경우 높은 수익성을 창출할 수 있을 것으로 예상된다.

4.3. 관련분야 survey

이름	장점	단점	개선점
 Chordly	음원으로부터 기타 코드를 추출하여 변환하는 기능	코드 외의 정보를 출력하지 않음. 지나치게 작은 단위의 코드까지 인식함. 인식률이 좋지 않음.	음원이 아닌, 악보로부터 코드를 추출. 지나치게 작은 단위의 코드를 인식하지 않게 함. 인식률 개선
 Yousician	어쿠스틱 및 전자악기 등의 다양한 악기를 지원함. 다양한 난이도의 곡들을 제공함. 실시간 코드 검사가 가능함.	인식률이 좋지 않음.	악보로부터 코드를 추출하는 기능 추가, 인식률 개선
 Hum	허밍을 멜로디로 변환하고, 멜로디를 반주로 만들어주는 기능을 제공함.	기타에는 적합하지 않음.	프로젝트에서 사용하는 기능이 없음
그 외 튜닝 어플리케이션	단음을 정확하게 인식함.	코드 인식 기능이 없음.	단음 뿐 아니라 코드 인식 기능까지 추가.

4.4. 유사 프로젝트 검색 및 해당 프로젝트 결과물들의 문제점

기존 소리 변환 프로젝트의 대부분이 인식률이 낮은 코드 변환 어플리케이션 프로젝트이거나, 코드가 아닌 단음만을 인식하는 단순한 튜닝 어플리케이션 프로젝트이다. 그리고 악보 인식 프로젝트의 대부분은 간단한 음표들만 인식 가능하거나, 음표 인식률이 낮은 학부 프로젝트로서, 실제 사용이 힘들다는 문제점이 분명하다.

4.5. - 기존 프로젝트와 제안 내용과의 차이 및 기존 문제 해결 방안

본 프로젝트는 이미지 인식과 소리 인식을 결합하여 기존의 어플리케이션들이 제공하지 않던 기능을 제공한다. 악보인식은 관련 어플리케이션이 존재하지 않던 상황이며, 기타의 소리인식은 것은 몇몇 어플리케이션들이 있다. 기존의 어플리케이션이 단음만을 인식하거나 코드인식의 정확도가 높지 않았던 것에 비해 본 프로젝트의 소리인식은 Unity engine 의 Fast Fourier Transform 을 사용하고 그 Window 로는 인식이 거의 실시간으로 이루어지고, 정확성을 보장할 수 있는 Blackman 을 사용하여 개선하였다.

5. 개발 내용

5.1. 개발 플랫폼

- Target Environment
 - Hardware Device : Android Smartphone
 - OS : Android 4.1 'Jelly Bean'이상
- Development Environment
 - OS : Windows 10
 - Tools : Microsoft Visual Studio 2015, Unity3d 5.6
 - Language : C#

5.2. 상세 요구사항

5.2.1. 기능적 요구사항 (functional requirements)

- 1) 악보 파일(PDF 또는 이미지)을 입력 받는다
- 2) 입력 받은 악보 파일을 영상처리를 통해 악보 구성요소를 추출한다.
- 3) 추출한 정보를 이용해 박자, 음 길이, 코드 등 음악 구성 요소를 저장한다.
- 4) 마이크로 사용자의 연주 소리를 입력 받는다.
- 5) 사용자의 연주 소리를 고속 푸리에 변환(Fast Fourier Transform)을 통해 주파수 별 신호로 분리한다.
- 6) 주파수 별 신호 크기를 분석하여 연주된 음들을 구분해 낸다.
- 7) 연주된 음들을 조합하여 연주된 코드를 인식한다.
- 8) 악보로부터 추출된 정보와 소리로부터 추출된 정보를 비교하여 올바른 연주를 했는지 판단한다.
- 9) 음 하나를 연주했을 때 해당 음을 인식, 기준이 되는 음과 비교하여 튠닝을 돕는다.

5.2.2. 비기능적 요구사항 (non-functional requirements)

- 1) UI 를 알아보기 쉬운 간단한 아이콘으로 구성하여 사용자들이 직관적으로 알아볼 수 있도록 한다.
- 2) 연주 소리를 분석하고 결과를 화면에 표시하는 데 걸리는 시간을 1 초 이내로 한다.
- 3) 점 음표, 쉼표, 쉼표, 쉼표 등 악보상의 특수한 기호들을 오류 없이 정확히 구분할 수 있도록 한다.
- 4) 연주된 음들을 분석할 때 반음 이하의 오차로 구분할 수 있도록 한다.
- 5) Android 모바일 시스템 상에서 오류 없이 구동될 수 있도록 한다.
- 6) Unity3D 5.6 버전에서 UI 및 메인 기능을 구현한다.
- 7) UI 스크립트 작성은 Microsoft Visual Studio 2015 에서 C#를 이용한다.
- 8) 연주 소리 분석 스크립트 작성은 Microsoft Visual Studio 2015 에서 C#를 이용한다.
- 9) 악보 이미지 분석 스크립트 작성은 Microsoft Visual Studio 2015 에서 C# OpenCV 라이브러리를 이용한다.
- 11) AAR 프로젝트를 Unity3D 에서 Import 해서 사용한다.
- 12) 프로그램은 입력된 이미지를 분석할 뿐이므로 악보 파일의 저작권 때문에 발생하는 이용에 따른 법적 책임은 사용자가 지도록 한다.

5.3. 상세 구조 (system components and relation between components)

1) PDF 이미지 변환

Android 내 함수를 사용한 PDF 이미지 변환을 통해, PDF 파일이 이미지로 저장된다.

2) 오선 인식

저장된 이미지 파일에서 OpenCV 내의 Line Hough transform(cv::HoughLines)을 이용하여 오선을 인식한다.



3) 오선 제거

오선 인식 후 검출한 오선들을 제거한다. 이때 특별한 API 를 사용하지 않고, 검출된 Line 을 따라 그 Line 의 대표 굵기값을 구하여 대표 굵기값 x 1.5 미만인 굵기의 Line pixel 을 제거한다. 오선 제거 후, 개체들이 분리되는 현상을 막기 위해 떨어진 픽셀들을 붙여주어 보간한다.



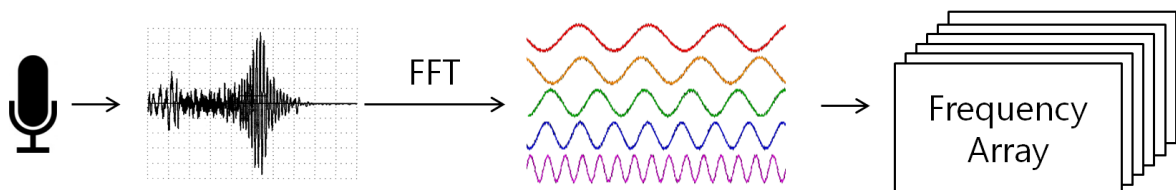
4) 음표 인식 및 분리

오선이 모두 제거되면 이진화된 이미지에서 픽셀값을 가질 경우 인접한 영역끼리 그룹을 지어 Labeling(레이블링)한다. 이후, 템플릿 매칭을 통해 음표를 분리한다.



5) 코드 변환

음표 인식 및 분리 후에는 생성된 음표 데이터를 다이아토닉 코드 등을 이용한 알고리즘을 통해 코드로 변환한다. 음표 데이터는 음의 길이, 음의 높이, 점의 유무, 꾸밈음의 유무 등의 정보들을 포함한 데이터이다.



6) 마이크 입력

분석할 소리를 담는 객체에 마이크를 연결해 마이크 입력을 받는다.

7) 소리 파형 분리

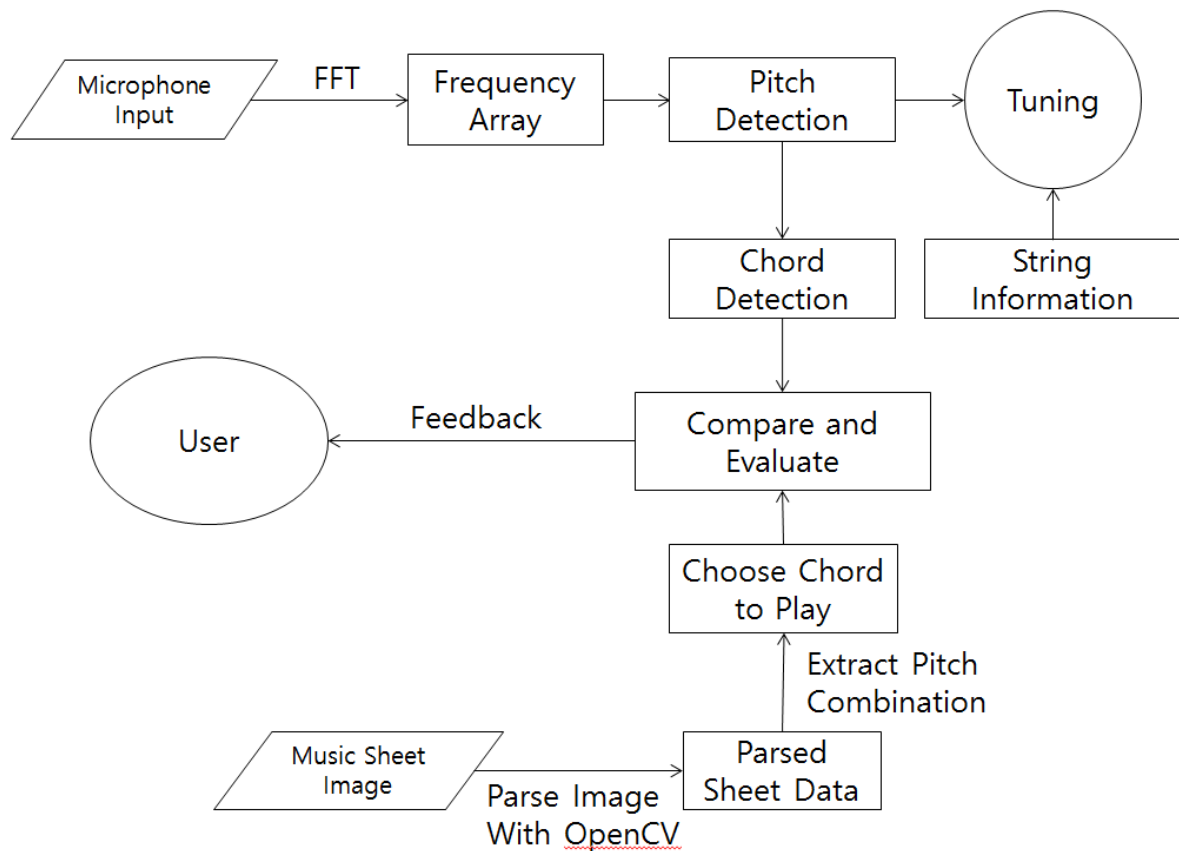
입력 받은 소리를 고속 푸리에 변환(FFT)를 거쳐 각 주파수 별 Sample 로 분리하여 해당 Sample 의 Amplitude 를 저장하는 8192 크기의 배열에 저장한다.

8) 음별 주파수 분석

배열에 저장된 Amplitude 를 분석해서 Sample 이 해당하는 음이 연주되는지 판단한다.

9) 코드 분석

연주된 음들을 조합하여 연주된 코드가 어떤 코드인지 알아낸다.



6. 개발 방법

6.1. 사용한 Library, API

- 6.1.1. Unity3d Engine : 기본적으로 제공되는 UI 도구 덕분에 UI 를 구성하기 편하고, 게임 개발용으로 개발 된 엔진이다 보니 다양한 기능들이 제공되어 여러 기능이 필요한 어플리케이션 개발에 적합하다고 생각되어 선택하였다.



- 6.1.2. UGUI : Unity Engine 에서 제공하는 UI 도구이다.

- 6.1.3. FFT 를 위한 GetSpectrumData 함수 (Unity 엔진 내장) : 참고문헌 1)

- 6.1.3.1. UnityEngine.AudioSource.GetSpectrumData : 참고문헌 6)

`public void GetSpectrumData(float[] samples, int channel, FFTWindow window);`

* samples : FFT 결과(audio samples)를 저장할 배열. 크기는 2 의 거듭제곱이어야 한다.

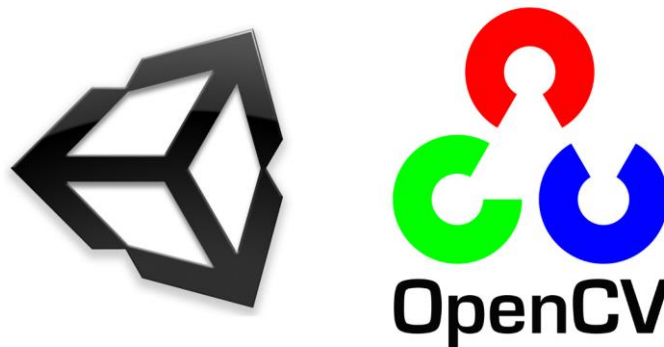
* channel : sampling 수행할 오디오 채널을 말한다.

* window : FFT 를 수행할 때 사용할 FFTWindow 타입을 말한다. : 참고문헌 7)

- 6.1.3.2. FFTWindow : 스펙트럼 분석을 할 때 사용하는 Windowing 타입을 표현한 enumeration 이다. 6 가지 종류의 Window Type 이 있다. 프로그램에서는 실험 결과 음을 확실히 인식 할 수 있고 만 하고 수행시간도 빨랐던 Blackman 를 사용하였다.

Rectangular	$W[n] = 1.0.$
Triangle	$W[n] = \text{TRI}(2n/N).$
Hamming	$W[n] = 0.54 - (0.46 * \cos(n/N)).$
Hanning	$W[n] = 0.5 * (1.0 - \cos(n/N)).$
Blackman	$W[n] = 0.42 - (0.5 * \cos(n/N)) + (0.08 * \cos(2.0 * n/N)).$
BlackmanHarris	$W[n] = 0.35875 - (0.48829 * \cos(1.0 * n/N)) + (0.14128 * \cos(2.0 * n/N)) - (0.01168 * \cos(3.0 * n/N)).$

6.1.4. OpenCV for Unity Asset



adaptiveThreshold : 참고문헌 8), 13) 참조

```
static void
OpenCVForUnity.Imgproc.adaptiveThreshold ( Mat src,
                                           Mat dst,
                                           double maxValue,
                                           int adaptiveMethod,
                                           int thresholdType,
                                           int blockSize,
                                           double C
                                           )
```

src	그레이 스케일 입력 이미지
dst	결과 이미지
maxValue	threshold 값보다 클 경우 결과 이미지의 해당 픽셀에 지정될 값
adaptiveMethod	Adaptive thresholding algorithm ADAPTIVE_THRESH_MEAN_C 또는 ADAPTIVE_THRESH_GAUSSIAN_C
thresholdType	Thresholding type THRESH_BINARY(threshold 보다 크면 maxvalue) 또는 THRESH_BINARY_INV(threshold 보다 작으면 maxvalue)
blockSize	threshold 값을 계산하기 위해 사용되는 블록 크기
C	계산된 평균으로부터 뺀 상수값

각 픽셀에 대한 threshold 값을 이웃하는 픽셀값에 의해 결정한다. (x,y) 위치에 있는 픽셀의 threshold 값을 계산하기 위하여 다음 단계를 거친다.

1. 선택된 픽셀 위치에 대해 b x b 크기의 블록을 설정한다.
2. 블록에 대한 가중 평균(weighted average)을 구한다.
3. 평균값에서 상수 파라미터 값 param1 을 빼 선택된 픽셀 위치에서 Threshold 값을 구한다
4. 구한 값이 Threshold 값보다 클 경우, 해당 픽셀에 지정된 값을 저장한다.

결과적으로, 빛에 의한 반사나 그림자가 있는 부분들에 대해 이진화 결과가 개선된다.

connectedComponentWithStats : 참고문헌 13) 참조

```
static int
OpenCVForUnity.Imgproc.connectedComponentsWithStats ( Mat image,
                                                         Mat labels,
                                                         Mat stats,
                                                         Mat centroids
                                                         )
```

image 레이블링할 이미지

labels 레이블링한 결과 이미지

stats 레이블링 된 이미지 배열

centroids 레이블링 된 이미지의 중심좌표

이진화 한 이미지에서 형체를 알아보기 위해 같은 픽셀 값들끼리 그룹화하여 번호를 매긴 것을 레이블링이라고 한다.


```
static void OpenCVForUnity.Imgproc.HoughLines ( Mat    image,
                                                  Mat    lines,
                                                  double rho,
                                                  double theta,
                                                  int     threshold
                                                  )
```

static

image 직선을 검출하고자 하는 이미지

lines 검출된 직선을 저장할 배열

rho 직선을 찾기 위한 단계별 rho 누적 해상도 크기 (기본 1 pixel)

theta 직선을 찾기 위한 단계별 theta 누적 해상도 크기 (기본 1 radian)

- 단계별로 가능한 모든 각도로 반지름의 선을 찾는다

threshold 직선임을 결정하기 위한 투표(vote) 최대 개수

$$p = x \cos \theta + y \sin \theta$$

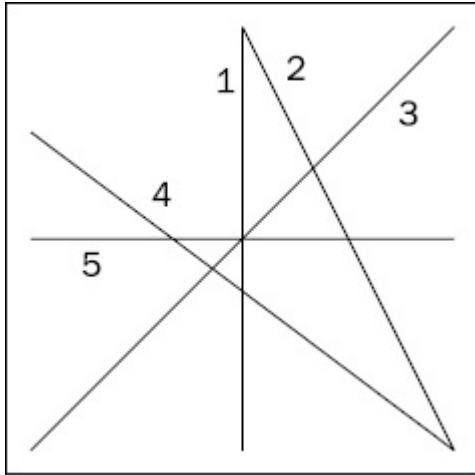
(x, y 평면을 rho, theta 평면으로 변환하여 직선을 검출한다)

- p : 직선과 영상 원점(왼쪽 상단 모서리)간의 거리.

- θ : 선에 대한 수직 각도.

- 식에서 영상 내에 보이는 선은 0 과 π (파이) 라디안 사이의 각도인 θ 를 갖고, p 반지름은 영상 대각선의 길이와 같은 최대값을 가진다.

- 다음 선 집합을 보았을 때



- 1 인 선인 수평선은 0 과 동일한 θ 각도 값을 갖고, 수평선(5 인 선)은 $\pi/2$ 와 동일한 θ 값을 가짐.
- 즉, 3 인 선은 $\pi/4$ 와 동일한 각도 θ 를 갖고, 4 인 선은 0.7π 에 가깝다.
- 가능한 모든 선을 $[0, \pi]$ 범위에 있는 θ 로 표현하기 위해서는 반지름 값을 음수로 만들 수 있음.
- 2 인 선의 경우에는 p 가 음수 값으로서 값이 0.8π 와 동일.

HoughLines 는 Hough transform(허프 변환)을 사용한다. 허프 변환을 간단히 설명하면, 평면(이미지)에서 수식으로 표현할 수 있는 도형(직선, 원, 타원, ...)을 검출할 수 있는 특징 추출법이고 "점 \leftrightarrow 직선" 변환을 하는 방법이다. HoughLines 는 이미지에서 수식으로 표현할 수 있는 직선을 검출하는 API 이다.

6.2. 직접 개발한 내용

6.2.1. 악보 변환

```
imgproc.adaptiveThreshold(~img, img, 255, imgproc.ADAPTIVE_THRESH_MEAN_C,
imgproc.THRESH_BINARY, 15, -20);
```

악보의 해상도나 화질이 좋지 않을 경우를 생각하여 adaptiveThreshold 를 적용하였다. 결과적으로, 빛에 의한 반사나 그림자가 있는 부분들에 대해 이진화 결과가 개선되었다. 참고문헌 8) 참조

```

Mat labels = new Mat(), stats = new Mat(), centroids = new Mat();
int numOfLabels = imgproc.connectedComponentsWithStats(img, labels, stats,
    centroids);

int width_thr = img.cols() / 2;

for (int j = 1; j < numOfLabels; j++)
{
    int[] left = new int[3];
    int[] top = new int[3];
    int[] width = new int[3];
    int[] height = new int[3];
    stats.get(j, imgproc.CC_STAT_LEFT, left);
    stats.get(j, imgproc.CC_STAT_TOP, top);
    stats.get(j, imgproc.CC_STAT_WIDTH, width);
    stats.get(j, imgproc.CC_STAT_HEIGHT, height);

    Mat staff = img.adjustROI(top[0], top[0] + height[0], left[0],
        left[0] + width[0]);

    if (width[0] > width_thr)
        lineList.Add(staff);
}

```

그 후, 악보를 각 소절(멜로디 + 반주) 단위로 분리하기 위하여 악보 이미지 전체에 레이블링을 한번 수행하였다. 레이블링 후, 각 소절 Mat 을 List 에 저장하였다.

```

Mat lines = new Mat();

List<KeyValuePair<Point, Point>> staffList = new List<KeyValuePair<Point, Point>>();

for (int idx = 0; idx < lineList.Count; idx++)
{
    int[] none = { 0, 0, 0 };
    staffList.Clear();
    Imgproc.HoughLines(lineList[idx], lines, 1, Mathf.PI / 180,
        (int)Mathf.Round(lineList[idx].cols() / 1.3f));
    float y_step = 0.0f;

    for (int i = 0; i < lines.rows(); i++)
    {
        float[] data = new float[3];
        lines.get(0, i, data);
        float rho = data[0];
        float theta = data[1];

        float a = Mathf.Cos(theta), b = Mathf.Sin(theta);
        float x0 = a * rho, y0 = b * rho;
        Point pt1 = new Point(Mathf.Round(x0 + 1000 * (-b)), Mathf.Round(y0 + 1000 * (a)));
        Point pt2 = new Point(Mathf.Round(x0 - 1000 * (-b)), Mathf.Round(y0 - 1000 * (a)));

        y_step = Mathf.Abs((2000.0f * (a)) / lineList[idx].cols());

        if (y_step < 0.4)
        {
            KeyValuePair<Point, Point> pts = new KeyValuePair<Point, Point>(pt1, pt2);
            staffList.Add(pts);
        }
    }

    staffList.Sort(delegate (KeyValuePair<Point, Point> A, KeyValuePair<Point, Point> B)
    {
        if (A.Key.y < B.Key.y)
            return 1;
        else if (A.Key.y > B.Key.y)
            return -1;
        else
            return 0;
    });

    staffStep = (int)(staffList[1].Key.y - staffList[0].Key.y);
}

```

Hough Transform 을 사용하여 각 소절에서 오선들을 검출하고, 검출된 오선들을 y 좌표를 기준으로 정렬한다. 그리고, 2 번째 오선줄과 1 번째 오선줄의 차를 오선줄의 step 으로 저장한다.

참고문헌 9), 13) 참조

```

numOfLabels = imgproc.connectedComponentsWithStats(img, labels, stats, centroids);

for (int j = 1; j < numOfLabels; j++)
{

    int[] left = new int[3];
    int[] top = new int[3];
    int[] width = new int[3];
    int[] height = new int[3];
    stats.get(j, Imgproc.CC_STAT_LEFT, left);
    stats.get(j, Imgproc.CC_STAT_TOP, top);
    stats.get(j, Imgproc.CC_STAT_WIDTH, width);
    stats.get(j, Imgproc.CC_STAT_HEIGHT, height);

    if (lines.rows() == 5)
    {
        if (((top[0] > (staffList[4].Key.y + staffStep)
            || top[0] < (staffList[0].Key.y - staffStep))
            && width[0] < lineList[idx].cols() / 3)
        {
            for (int x = left[0]; x < left[0] + width[0]; x++)
                for (int y = top[0]; y < top[0] + height[0]; y++)
                    lineList[idx].put(y, x, none);
        }
    }
    else
    {
        if (((top[0] > (staffList[4].Key.y + staffStep)
            && top[0] < (staffList[5].Key.y - staffStep))
            || top[0] < (staffList[0].Key.y - staffStep)
            || top[0] > (staffList[9].Key.y + staffStep))
            && width[0] < lineList[idx].cols() / 3)
        {
            for (int x = left[0]; x < left[0] + width[0]; x++)
                for (int y = top[0]; y < top[0] + height[0]; y++)
                    lineList[idx].put(y, x, none);
        }
    }
}

```

악보 변환에서 필요없는 오선 바깥쪽에 있는 개체(가사, p, mp, mf 등의 악상 기호, ...)들을 레이블링을 통해 제거한다.

```

for (int k = 0; k < lines.rows(); k++)
{
    int mx = 0, mxidx = -1;
    float[] data = new float[3];
    lines.get(0, k, data);
    float rho = data[0];
    float theta = data[1];

    float a = Mathf.Cos(theta), b = Mathf.Sin(theta);
    float x0 = a * rho, y0 = b * rho;

    y_step = Mathf.Abs((2000.0f * (a)) / lineList[idx].cols());

    if (y_step > 0.4)
        continue;

    int[] vcl = new int[30];

    for (int i = 0; i < 30; i++)
        vcl[i] = 0;
    for (int col = 0; col < lineList[idx].cols(); col++)
        getVCL(lineList[idx], ref vcl,
            (int)Mathf.Round(y0 - Mathf.Abs(1000 * (a)) + y_step * col), col);
    for (int i = 0; i < 30; i++)
        if (mx < vcl[i])
        {
            mx = vcl[i];
            mxidx = i;
        }
    for (int col = 0; col < lineList[idx].cols(); col++)
        removeStaff(lineList[idx], (int)Mathf.Round((float)mxidx * 1.5f), |
            (int)Mathf.Round(y0 - Mathf.Abs(1000 * (a)) + y_step * col), col);
}

```

```

static int cntVCL(Mat src, int y, int x)
{
    int VCL = 0;
    bool Up = true, Down = true;
    int[] data = new int[3];
    src.get(y, x, data);
    if (data[0] != 0)
        VCL++;
    else
    {
        src.get(y - 1, x, data);
        if (data[0] != 0)
        {
            VCL++;
            y = y - 1;
        }
        else
        {
            src.get(y + 1, x, data);
            if (data[0] != 0)
            {
                VCL++;
                y = y + 1;
            }
            else
                return 0;
        }
    }

    for (int i = 0; i < 10; i++)
    {
        if (Up | Down)
        {
            if ((y + i) <= src.rows())
            {
                src.get(y + i, x, data);
                if (data[0] != 0 || !Up)
                    Up = false;
            }
            else
                VCL++;
            if ((y - i) >= 0)
            {
                src.get(y - i, x, data);
                if (data[0] != 0 || !Down)
                    Down = false;
            }
            else
                VCL++;
        }
        else
            break;
    }
    return VCL;
}

```

오선을 제거 과정에서는 VCL 를 사용하였다. VCL 은 위 방향과 아래 방향으로의 연결된 점들을 의미하고, 오선의 라인들을 따라서 VCL 의 길이를 계산한다. 결과적으로 이 VCL 길이의 최빈값을 오선의 두께로 추정하여 오선을 제거한다. : 참고문헌 10), 11)

```
List<List<bool>> ipList = new List<List<bool>>();

for (int row = 0; row < lineList[idx].rows(); row++)
{
    List<bool> bools = new List<bool>();
    for (int col = 0; col < lineList[idx].cols(); col++)
        bools.Add(false);
    ipList.Add(bools);
}

for (int y = 1; y < lineList[idx].rows() - 1; y++)
    for (int x = 1; x < lineList[idx].cols() - 1; x++)
    {
        int[] data = new int[3];
        lineList[idx].get(y, x, data);
        if (data[0] == 0)
            ipList[y][x] = interpolation(lineList[idx], y, x);
    }

for (int y = 1; y < lineList[idx].rows() - 1; y++)
    for (int x = 1; x < lineList[idx].cols() - 1; x++)
        if (ipList[y][x])
            lineList[idx].put(y, x, none);

ipList.Clear();
```

1 2 3
4 5 6
7 8 9

오선을 제거한 후, 혹시라도 오선이 아닌 픽셀이 제거가 되었을 경우를 생각하여 이미지를 보간한다(예를 들면 그림의 5 번 픽셀이 호출되었을 경우 (1, 6), (1, 8), (1, 9), (2, 7), ... 등).

```

List<Mat> subLines = new List<Mat>( );
List<List<bar>> bars = new List<List<bar>>( );

if (staffList.Count > 5)
{
    Mat sub1 = lineList[idx].adjustROI(0,
        (int)(staffList[5].Key.y + staffList[4].Key.y) / 2, 0, lineList[idx].cols());
    Mat sub2 = lineList[idx].adjustROI((int)(staffList[5].Key.y + staffList[4].Key.y) / 2 + 1,
        lineList[idx].rows(), 0, lineList[idx].cols());
    List<bar> bar1 = new List<bar>( );
    List<bar> bar2 = new List<bar>( );

    subLines.Add(sub1);
    subLines.Add(sub2);
    bars.Add(bar1);
    bars.Add(bar2);

    int y = (int)(staffList[5].Key.y + staffList[4].Key.y) / 2;
    for (int x = 0; x < lineList[idx].cols(); x++)
        lineList[idx].put(y, x, none);
}
if (staffList.Count > 10)
{
    Mat sub1 = lineList[idx].adjustROI(0, (int)(staffList[5].Key.y + staffList[4].Key.y) / 2,
        0, lineList[idx].cols());
    Mat sub2 = lineList[idx].adjustROI((int)(staffList[5].Key.y + staffList[4].Key.y) / 2 + 1,
        (int)(staffList[10].Key.y + staffList[9].Key.y) / 2, 0, lineList[idx].cols());
    Mat sub3 = lineList[idx].adjustROI((int)(staffList[9].Key.y + staffList[10].Key.y) / 2 + 1,
        lineList[idx].rows(), 0, lineList[idx].cols());
    List<bar> bar1 = new List<bar>( );
    List<bar> bar2 = new List<bar>( );
    List<bar> bar3 = new List<bar>( );

    subLines.Add(sub1);
    subLines.Add(sub2);
    subLines.Add(sub3);
    bars.Add(bar1);
    bars.Add(bar2);
    bars.Add(bar3);

    int y = (int)(staffList[10].Key.y + staffList[9].Key.y) / 2;
    for (int x = 0; x < lineList[idx].cols(); x++)
        lineList[idx].put(y, x, none);
}

```

이후, 오선의 y 좌표를 기준으로 각 소절을 오선별(멜로디, 반주 오른손, 반주 왼손)로 자르고 각 오선들의 마디 List 를 할당한다.

```

for (int i = 0; i < 17; i++)
{
    if (File.Exists(i+".jpg"))
    {
        fileData2 = File.ReadAllBytes(filePath2);
        tmp = new Texture2D(2, 2);
        tmp.LoadImage(fileData2);
    }
    Mat tem = new Mat(tmp.height, tmp.width, CvType.CV_8UC1);
    Utils.texture2DToMat(tmp, tem);
    Imgproc.matchTemplate(subLines[index], tem, result, Imgproc.TM_CCORR);
    Imgproc.threshold(result, result, 0.8, 1.0, Imgproc.THRESH_TOZERO);
}

```

전처리 된 악보에서 악보를 구성하는 요소 17 개에 대하여 Template matching 을 실시하여 각 요소가 어떤 요소인지 구분하였다. Template matching 의 method 로는 CV_TM_CCORR 을 사용하여 Sample image 와의 유사도에 따른 판단면에서 높은 인식결과를 나타내었다. 위의 matchTemplate 함수의 결과값으로 일치하면 0 또는 작은 값, 일치한다고 판단되면 큰 값이 나오는데, 일정한 값 이상이 나온다면 그 요소로 결정하게 하였다.

참고문헌 15), 17)참조


```

public enum Kind
{
    CLEF, NODE, ACCIDENTAL, NOTE, REST, BEAT, BAR
}

public enum Accidental
{
    NONE, SHARP, FLAT, NATURAL
}

public enum Clef
{
    TREBLE, LOW
}

public enum TimeSignature
{
    FOUR_FOUR, THREE_FOUR, TWO_FOUR, TWELVE_EIGHT, NINE_EIGHT, SIX_EIGHT
}

public enum KeySignature
{
    Cf, C, Cs, Df, D, Ef, E, F, Fs, Gf, G, Gs, Af, A, Bf, B
}

public struct location
{
    public int left;
    public int top;
    public int width;
    public int height;
}

public struct note
{
    public int tone;
    public Accidental acci;

    public note(int t, Accidental a)
    {
        this.tone = t;
        this.acci = a;
    }
}

public struct obj
{
    public Kind k;
    public location loc;
    public Clef c;
    public Accidental acci;
    public note n;
    public TimeSignature ts;
}

```

그리고 그 요소에 해당하는 정보를 각 obj struct 에 저장하여 위치, 종류에 따른 정보를 data 로 변환하였다.

```

for (int i = 0; i < informOfObjects.Count; i++)
{
    if (informOfObjects[i].k == Kind.CLEF)
    {
        if (informOfObjects[i].c == Clef.TREBLE)
            clef = Clef.TREBLE;
        else
            clef = Clef.LOW;
        isAcci = false;
        ksCnt = 0;
    }
    else if (informOfObjects[i].k == Kind.ACCIDENTAL)
    {
        if (isAcci)
            acc = informOfObjects[i].acci;
        else
            ksCnt++;
    }
    else if (informOfObjects[i].k == Kind.BEAT)
    {
        ts = informOfObjects[i].ts;
        if (ts == TimeSignature.FOUR_FOUR)
            System.IO.File.AppendAllText(@"%save%test.txt", "4 4" + "\n",
                System.Text.Encoding.Default);
        else if (ts == TimeSignature.THREE_FOUR)
            System.IO.File.AppendAllText(@"%save%test.txt", "3 4" + "\n",
                System.Text.Encoding.Default);
        else if (ts == TimeSignature.TWO_FOUR)
            System.IO.File.AppendAllText(@"%save%test.txt", "2 4" + "\n",
                System.Text.Encoding.Default);
        else if (ts == TimeSignature.TWELVE_EIGHT)
            System.IO.File.AppendAllText(@"%save%test.txt", "12 8" + "\n",
                System.Text.Encoding.Default);
        else if (ts == TimeSignature.NINE_EIGHT)
            System.IO.File.AppendAllText(@"%save%test.txt", "9 8" + "\n",
                System.Text.Encoding.Default);
        else if (ts == TimeSignature.SIX_EIGHT)
            System.IO.File.AppendAllText(@"%save%test.txt", "6 8" + "\n",
                System.Text.Encoding.Default);
        if (acc == Accidental.NONE)
            ks = KeySignature.C;
        else if (acc == Accidental.FLAT)
        {
            if (ksCnt == 1)
                ks = KeySignature.F;
            else if (ksCnt == 2)
                ks = KeySignature.Bf;
            else if (ksCnt == 3)
                ks = KeySignature.Ef;
            else if (ksCnt == 4)
                ks = KeySignature.Af;
            else if (ksCnt == 5)
                ks = KeySignature.Df;
            else if (ksCnt == 6)
                ks = KeySignature.Gf;
            else
                ks = KeySignature.Cf;
        }
        else if (acc == Accidental.SHARP)
        {
            if (ksCnt == 1)
                ks = KeySignature.G;
            else if (ksCnt == 2)
                ks = KeySignature.D;
            else if (ksCnt == 3)
                ks = KeySignature.A;
            else if (ksCnt == 4)
                ks = KeySignature.E;
            else if (ksCnt == 5)
                ks = KeySignature.B;
            else if (ksCnt == 6)
                ks = KeySignature.Fs;
            else
                ks = KeySignature.Cs;
        }
        acc = Accidental.NONE;
    }
}

```

저장된 object 들에 대해서 x 좌표로 정렬하고, 각 object 의 종류에 따라 각각 다른 결과를 내도록 설정하였다. 예를 들면, 박자를 뜻하는 BEAT 에서는 박자를 txt 파일로 출력하고 이전에 만났던 #이나 b 의 개수에 따라 조를 결정한다.

참고문헌 12) 참조

```

else if (informOfObjects[i].k == Kind.NOTE)
{
    int topCnt = 0, botCnt = 0, head = -1, tone = -1;
    int upTop = informOfObjects[i].loc.top;
    int upBottom = (informOfObjects[i].loc.top
        + informOfObjects[i].loc.height) / 4;
    int downTop = (informOfObjects[i].loc.top
        + informOfObjects[i].loc.height) * 3 / 4;
    int downBottom = informOfObjects[i].loc.height;
    int[] data = new int[3];
    // top
    for (int y = upTop; y < upBottom; y++)
    {
        for (int x = informOfObjects[i].loc.left;
            x < informOfObjects[i].loc.left + informOfObjects[i].loc.width; x++)
        {
            lineList[idx].get(y, x, data);
            if (data[0] != 0)
                topCnt++;
        }
    }
    // bot
    for (int y = downTop; y < downBottom; y++)
    {
        for (int x = informOfObjects[i].loc.left;
            x < informOfObjects[i].loc.left + informOfObjects[i].loc.width; x++)
        {
            lineList[idx].get(y, x, data);
            if (data[0] != 0)
                botCnt++;
        }
    }

    if (topCnt > botCnt)
        head = (upTop + upBottom) / 2;
    else
        head = (downTop + downBottom) / 2;

    tone = getTone(lineList[idx].rows(), head, staffStep, (int)staffList[0].Key.y, true);
    note n = new note();
    n.acci = acc;
    n.tone = tone;

    bb.notes.Add(n);
}

```

```

static int getTone(int height, int head, int staffStep,
    int firstStaff, bool isTreble)
{
    int tone = -1, curr;
    int start = firstStaff - staffStep * 3 / 2;
    int octave = staffStep * 7 / 2;
    int threshold = staffStep / 2;
    bool goOut = false;

    for (int i = 11; i >= 0; i--)
    {
        if (start < 0)
        {
            start += threshold;
            continue;
        }
        curr = start;
        while (curr < height)
        {
            if (Mathf.Abs(head - curr) < threshold)
            {
                goOut = true;
                break;
            }
            curr += octave;
        }
        tone = i;
        if (goOut)
            break;
    }
    if (!isTreble)
    {
        // 낮은음자리표일 경우
        tone += 4;
        tone = tone % 12;
        if (tone == 1 || tone == 3 || tone == 6
            || tone == 8 || tone == 10)
            tone--;
    }

    return tone;
}

else if (informOfObjects[i].k == Kind.BAR)
{
    bb.ks = ks;
    bb.ts = ts;
    bars[index].Add(bb);
    bb.notes.Clear();
}

```

음표를 만났을 경우, 음표의 머리를 구하여 머리의 중간이 오선 중 어느 곳에 가까운지를 구하여 음 높이를 구하고, 마디줄을 만났을 경우에는 마디를 마디 List 에 저장한다.

```

static string getChord_(bar b, int ks)
{
    int[,] chord = new int[12, 5];
    int t = -1, c = -1, mx = -1;
    string chordStr = "";

    for (int i = 0; i < 12; i++)
        for (int j = 0; j < 5; j++)
            chord[i, j] = 0;

    for (int i = 0; i < b.notes.Count; i++)
    {
        t = b.notes[i].tone;
        for (int x = 0; x < 12; x++)
        {
            if (t == x || t == ((4 + x) % 12) || t == ((7 + x) % 12))
                chord[x, 0]++;
            if (t == x || t == ((3 + x) % 12) || t == ((7 + x) % 12))
                chord[x, 1]++;
            if (t == x || t == ((5 + x) % 12) || t == ((7 + x) % 12))
                chord[x, 2]++;
            if (t == x || t == ((4 + x) % 12) || t == ((7 + x) % 12)
                || t == ((11 + x) % 12))
                chord[x, 3]++;
            if (t == x || t == ((4 + x) % 12) || t == ((7 + x) % 12)
                || t == ((10 + x) % 12))
                chord[x, 4]++;
        }
    }

    for (int i = 0; i < 12; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            if (mx < chord[(i + ks), j])
            {
                t = (i + ks) % 12;
                c = j;
                mx = chord[(i + ks), j];
            }
        }
    }

    if (t == 0)
        chordStr = "C";
    else if (t == 1)
        chordStr = "C#";
    else if (t == 2)
        chordStr = "D";
    else if (t == 3)
        chordStr = "D#";
    else if (t == 4)
        chordStr = "E";
    else if (t == 5)
        chordStr = "F";
    else if (t == 6)
        chordStr = "F#";
    else if (t == 7)
        chordStr = "G";
    else if (t == 8)
        chordStr = "G#";
    else if (t == 9)
        chordStr = "A";
    else if (t == 10)
        chordStr = "A#";
    else if (t == 11)
        chordStr = "B";

    if (c == 1)
        chordStr = chordStr + "m";
    else if (c == 2)
        chordStr = chordStr + "sus4";
    else if (c == 3)
        chordStr = chordStr + "7";
    else if (c == 4)
        chordStr = chordStr + "m7";

    return chordStr;
}

```

마지막으로 저장된 마디들을 코드변환한다. 마디 안에 있는 음표를 가장 많이 포함하고 있는 코드가 최종 코드로 선택된다. 선택된 코드는 String 형태로 .txt 파일로 출력된다.
참고문헌 12) 참조

6.2.2. 소리 인식

6.2.2.1. 소리 파형 변환 : 마이크로 들어온 소리를 FFT(고속 푸리에 변환)을 통해 8192 개의 sample 로 나누어 저장한다.

참고문헌 1), 6), 7)

```

public static float[] samples = new float[8192];
source.GetSpectrumData(samples, 0, FFTWindow.Blackman);

```

6.2.2.2. 음별 주파수 인식 : 각 옥타브에 대해서 A(라) 음의 주파수를 기준으로 각 음들의 주파수를 계산한 후, 해당 음이 samples 의 몇번째 index 에 sampling 되었는 지 계산한다.

```
float[] freqOfA = { 440f, 880f, 1760f, 3520f };
float[][] freqs = new float[4][];
int[][] indexs = new int[4][];
```

```
float rt = Mathf.Pow(2, 1f / 12f);
for (int i=0; i<4; i++)
{
    indexs[i] = new int[12];
    freqs[i] = new float[12];
    freqs[i][9] = freqOfA[i];

    for (int j = 9; j > 0; j--)
        freqs[i][j - 1] = freqs[i][j] / rt;

    for (int j = 10; j < 12; j++)
        freqs[i][j] = freqs[i][j - 1] * rt;

    for (int j = 0; j < 12; j++)
        indexs[i][j] = (int)(Mathf.Round(freqs[i][j] / fftSound.dist));
}
```

6.2.2.3. 튠닝 : 모든 sample 들의 신호 크기를 scan 하여 가장 큰 Amplitude 를 보이는 음을 찾아 현재 연주되는 음을 인식한다.

```
max = 0;
for(int i=0; i<8192; i++)
{
    if(fftSound.samples[i] > fftSound.samples[max])
        max = i;
}
```

6.2.2.4. 코드 추출 : 각 코드를 구성하는 음들의 정보를 저장한 후, 연주하는 소리에서 해당 코드를 구성하는 음들의 신호 크기가 일정 수준 이상이면 해당 코드를 연주하고 있다고 판단한다.

6.2.2.4.1. 코드 구성음 저장

```

void initializeChords()
{
    bases = new int[24];
    chords = new chord[12] [];
    for (int i = 0; i < 12; i++)
    {
        bases[i] = i;
        bases[i + 12] = i;
        // chords[0][0] : C, [1] : Cm, [2] : C7, [3] : Cm7, [4] : Csus4
        chords[i] = new chord[5];

        for(int j=0; j<5; j++)
            chords[i][j] = new chord();

        chords[i][0].majorChord(i);
        chords[i][1].minorChord(i);
        chords[i][2].dominant7Chord(i);
        chords[i][3].minor7Chord(i);
        chords[i][4].sus4Chord(i);
    }
}

```

* majorChord, minorChord, dominant7Chord, minor7Chord, sus4Chord
함수는 근음을 기준으로 각 코드를 구성하는 음들을 세팅해 주는 함수이다.

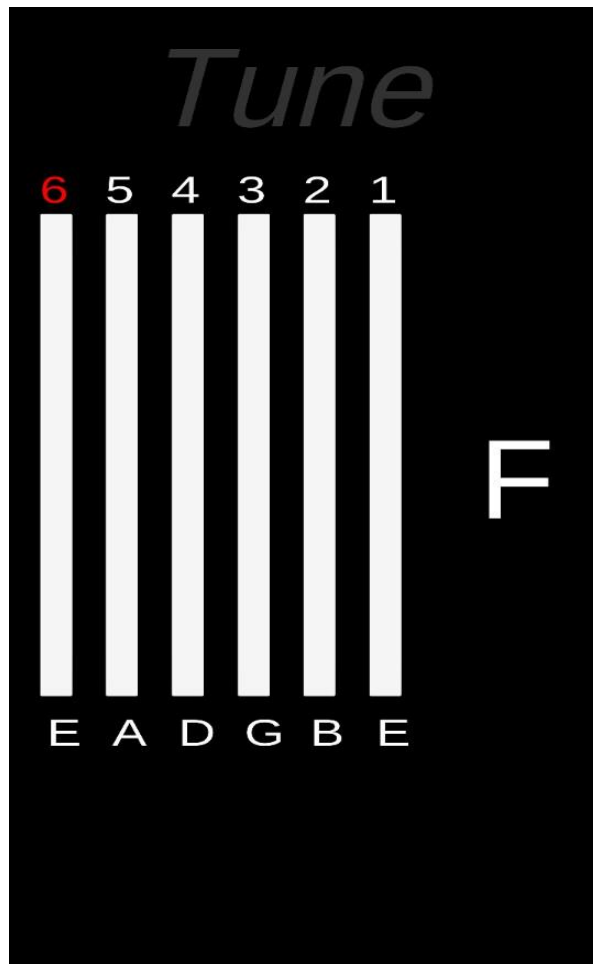
6.2.3. UI

6.2.3.1. 튠닝 : 6 번 줄부터 1 번 줄까지 정확한 음을 맞추면 해당 줄을
초록색으로 표시하고, 다음 줄로 넘어가도록 한다.

```

if (selected < 6)
{
    for (int i = 0; i < 4; i++)
    {
        stringNums[selected].color = Color.red;
        if (max == indexs[i][pitches[selected]])
        {
            stringNums[selected].color = Color.green;
            strings[selected].GetComponent<Image>().color = Color.green;
            selected++;
        }
    }
}

```



6.2.3.2. 악보 선택

- 1) 디렉토리를 스캔하여 pdf 또는 이미지 파일 목록을 보여준다.


```

public static string m_sdCardPath;
public static string CurrentSDCardPath
{
    get
    {
        if (m_sdCardPath == null)
        {
            AndroidJavaClass jc = new AndroidJavaClass("android.os.Environment");
            IntPtr getExternalStorageDirectoryMethod = AndroidJNI.GetStaticMethodID(jc.GetRawClass(),
                "getExternalStorageDirectory", "()Ljava/io/File;");
            IntPtr file = AndroidJNI.CallStaticObjectMethod(jc.GetRawClass(),
                getExternalStorageDirectoryMethod, new jvalue[] { });
            IntPtr getPathMethod = AndroidJNI.GetMethodID(AndroidJNI.GetObjectClass(file),
                "getPath", "()Ljava/lang/String;");
            IntPtr path = AndroidJNI.CallObjectMethod(file, getPathMethod, new jvalue[] { });
            m_sdCardPath = AndroidJNI.GetStringUTFChars(path);
            AndroidJNI.DeleteLocalRef(file);
            AndroidJNI.DeleteLocalRef(path);
            Debug.Log("m_sdCardPath = " + m_sdCardPath);
        }
        return m_sdCardPath;
    }
}

```

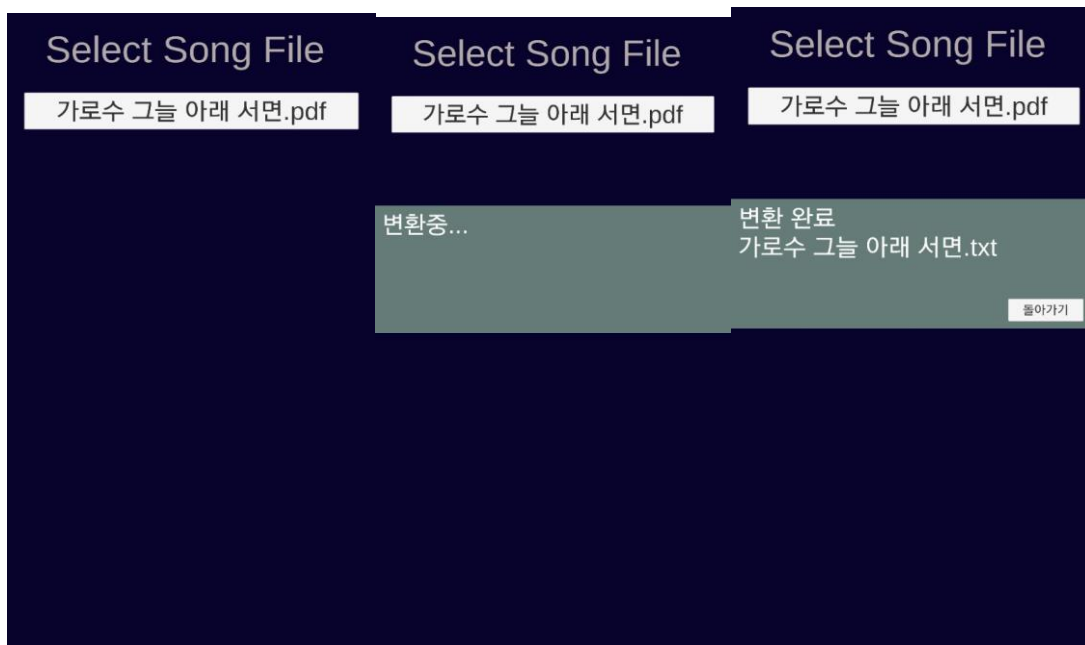
```

// 파일 스캔
for (int i = 0; i < files.Length; i++)
{
    filename = files[i].Substring(path.Length+1);

    GameObject obj = Instantiate(songPrefab);
    obj.transform.SetParent(this.GetComponent<RectTransform>());
    obj.transform.position = this.transform.position;
    obj.transform.position -= new Vector3(0, -860 + 200*i);
    obj.GetComponentInChildren<Text>().text = filename;
    obj.GetComponent<Button>().onClick.AddListener(() => songClicked(obj));
}

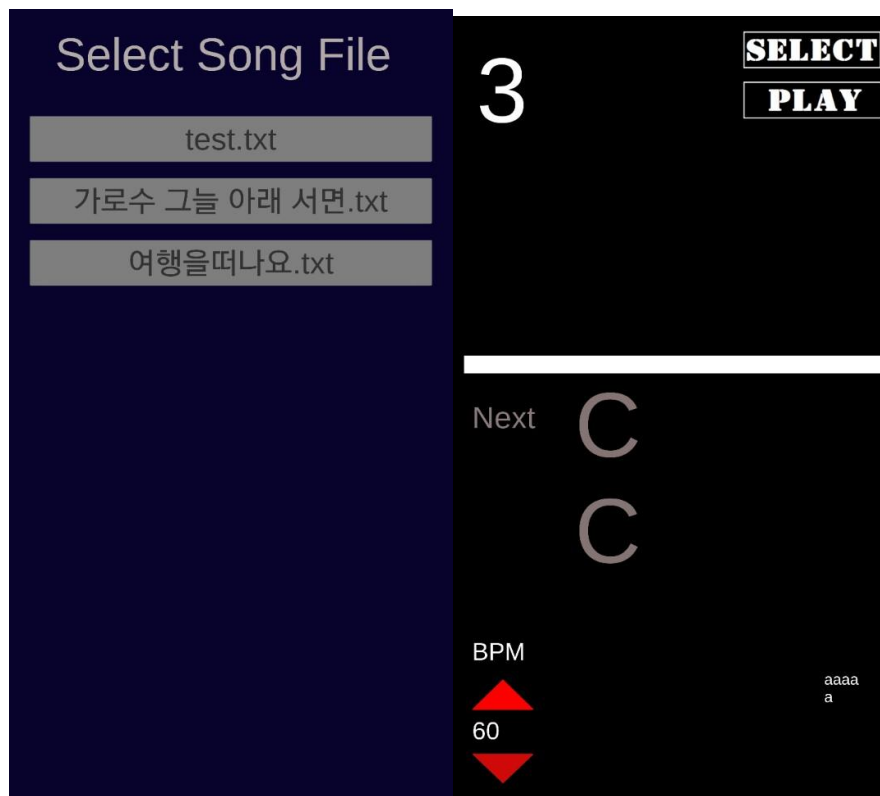
```

- 2) 변환하고 싶은 악보 파일을 선택하면 변환 알고리즘을 통해 코드를 추출한다.
- 3) 코드를 추출한 결과를 txt 파일에 출력하여 저장한다.
- 4) 변환이 완료되면 돌아가기 버튼이 활성화되어 메인 화면으로 돌아갈 수 있다.



6.2.3.3. 텍스트 선택

- 1) 악보선택과 마찬가지로 디렉토리를 스캔하여 곡 정보를 담은 텍스트 파일 목록을 가져온다.
- 2) 연습할 파일을 클릭하면 해당 텍스트파일을 열어 코드 정보를 읽어온 후 메모리에 저장한다.
- 3) 이후 연습화면으로 넘어간다.



6.2.3.4. 연습

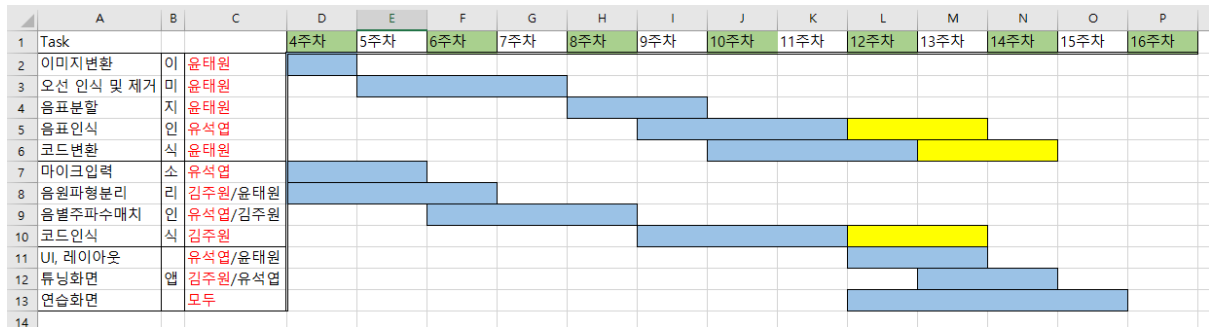
- 1) 곡 선택 이후 연습화면이 로딩되면 3 박자 후 곡이 시작된다.
- 2) 한 박자당 코드가 하나씩 출력되며 다음 두 박자에 해당하는 코드가 미리 보여진다.
- 3) 해당 코드와 연주한 코드가 일치한 경우 녹색, 일치하지 않은 경우 빨간색 배경으로 사용자의 연주 결과를 알려준다.



4) BPM 을 조절하여 연습 속도를 변경할 수 있다.

6.3. 개발 일정 및 팀원간 역할 분담

6.3.1. 간트 차트



6.3.2. PROJECT MILESTONE

4,5 주차

-전체

프로젝트 환경 준비 완료

-악보변환

이미지 변환(android pdfrender) 기능 구현 완료

오선 인식(Hough transform) 구현 완료

-소리변환

마이크 입력 기능 구현 완료

음원 파형 분리 함수 구현 시작

6,7 주차

-악보변환

오선 인식, 제거 및 보간 완료

-소리변환

음원 파형 분리 구현 완료

음별 주파수 매치 구현 시작

8,9 주차

-악보변환

음표 분할(Labeling) 완료

음표 데이터 클래스 설계

음표 인식(Template matching 사용) 함수 구현 시작

-소리변환

음별 주파수 매치 구현 완료

코드 인식 구현 시작

10, 11 주차

-악보변환

음표 인식 구현 완료

음표 데이터 구현 완료

코드 변환 함수 구현 시작

-소리변환

코드 인식 기능 구현 완료

12, 13 주차

-악보변환

코드 변환 구현 완료

악보 변환 기능 구현 완료

안드로이드 프로젝트를 Unity 에 포팅

-UI 구현

UI, 레이아웃 구현 시작

14, 15 주차

-UI 구현

튜닝화면 및 연습화면 구현 완료

-전체

프로젝트 완성

-> 음표인식, 코드변환, 코드인식 파트에서 당초 계획보다 2 주의 기간이 더 소요되었다.

7. 개발 결과

7.1. - 최종 결과물에 대한 스크린샷

7.1.1. 악보 변환

여행을 떠나요

7.1.2. 소리 인식 및 UI



7.2. 기능적 요구사항 및 비기능적 요구사항에 대한 달성도

- 2 GB ram, 2.5 GHz CPU, Android 4.4 KitKat 사양 이상의 스마트폰에서 1초 내에 기타 코드를 인식하여 처리. (성공)
- 위의 사양의 디바이스에서 1초 내에 온음 이상의 입력 음 오차를 6줄 중에 5줄 이상 인지하여 처리. (성공)
- 위의 사양의 디바이스에서 악보 파일을 입력하여, 코드 변환 오차를 5% 미만으로 처리. (성공, 개선 완료)

7.3. demo 동영상 youtube 링크

<https://youtu.be/svYB1viptiU>

7.4. - 테스트 결과 (정량적 수치 및 객관적 데이터 제시)

음	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
(R/T)	C	C#	D	D#	E	F	F#	G	G#	A	A#	B

튜너 기능에서 각 음에 대해 반음 단위로 인식이 가능한지(총 12 가지) 테스트 해보았을 때, 수행 결과가 정확함을 볼 수 있었다.

근음 / 코드	Major	minor	7	minor 7	Sus 4
C	C	Cm	C7	Cm7	Csus4
C#	C#	C#m	C#7	C#m7	C#sus4
D	D	Dm	D7	Dm7	Dsus4
D#	D#	D#m	D#7	D#m7	D#sus4
E	E	Em	E7	Em7	Esus4
F	F	Fm	F7	Fm7	Fsus4

근음 / 코드	Major		minor		7		minor 7		Sus 4	
F#	F#	F#	F#m	F#m	F#7	F#7	F#m7	F#m7	F#sus4	F#sus4
G	G	G	Gm	Gm	G7	G7	Gm7	Gm7	Gsus4	Gsus4
G#	G#	G#	G#m	G#m	G#7	G#7	G#m7	G#m7	G#sus4	G#sus4
A	A	A	Am	Am	A7	A7	Am7	Am7	Asus4	Asus4
A#	A#	A#	A#m	A#m	A#7	A#7	A#m7	A#m7	A#sus4	A#sus4
B	B	B	Bm	Bm	B7	B7	Bm7	Bm7	Bsus4	Bsus4

실시간 코드 인식 기능에서 각 코드에 대해 반음 단위로 5 가지의 코드 종류가 인식이 가능한지(총 60 가지) 테스트 해보았을 때, 수행 결과가 정확함을 볼 수 있었다.

8. 결론

8.1. 위에 포함되지 않은, 추가로 설명할 사항

8.1.1. Template matching 과 Feature matching

당초 Image matching 방식으로 Feature matching 을 사용하려 하였다. Feature matching 은 유사도를 분석하는데에 제일 적합하기 때문이다. 따라서 악보인식의 범용성을 높이기 위해 Feature matching 을 사용하려 했으나 악보의 요소들을 labling 하여 Feature matching 을 진행한 결과, 인식의 정확성이 90%정도로 나타나게 되었다. 이 또한 낮은 비율은 아니지만, 악보 인식의 특성상 10%정도의 오차로도 그 결과에 영향을 줄 수 있다고 판단, Template matching 방식으로 진행하게 되었다. Feature matching 의 정확성이 낮은 이유로는, 그 특성상 복잡하고 크기가 큰 이미지에 대해서는 98%이상의 높은 인식률을 나타내지만, labling 된 악보의 요소들은 그 크기가 작아 Feature detecting 과정에서 많은 Feature 들이 detecting 되지 않는다는 점이 발견되었다.

참고문헌 17) 참조

따라서 Template matching 으로 진행하게 되었고, 그 method 에 대한 설명은 다음과 같다.

1) method=CV_TM_SQDIFF

픽셀값의 제곱차를 이용하는 제곱차 매칭 방법이다. 템플릿 T 을 탐색 영역 I 에서 이동시켜가며 차이의 제곱의 합계를 계산한다. 매칭되는 위치에서 작은 값을 갖는다. 완벽하게 일치하면 0 을 반환하지만, 일치하지 않으면 값이 커진다.

$$R(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2$$

2) method=CV_TM_SQDIFF_NORMED

제곱차 매칭 방법에서 정규화 계수를 나눈 것이다. 정규화된 방법은 입력 영상과 템플릿 영상 사이에 조명의 차이가 존재할 때(ex. 야외에서 촬영한 경우), 그 영향을 크게 줄여주기 때문에 유용하게 사용된다.

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

3) method=CV_TM_CCORR

상관관계 방법으로 템플릿과 입력 영상의 곱을 제공하여 모두 더한다. 매칭되는 위치에서 큰 값을 갖는다. 완벽하게 일치하면 값이 크게 나오고, 일치하지 않으면 작은 값이 나오거나 0 이 나온다.

$$R(x, y) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))$$

4) method=CV_TM_CCORR_NORMED

상관관계 방법에서 정규화 계수를 나눈 것.

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

5) method=CV_TM_CCOEFF

상관계수 방법으로 템플릿과 입력 영상 각각의 평균을 고려한 매칭을 수행한다. 그러므로 완벽하게 일치하면 1 을 반환하고, 완전히 불일치하면 -1 을 반환한다. 0 을 반환하는 경우는 두 영상 사이에 전혀 연관성이 없음을 의미한다. 보통 상관계수 방법이 가장 정확한 형태의 검출 결과를 보여주나 그만큼 연산량이 제일 많다.

$$R(x, y) = \sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))$$

$$T'(x', y') = T(x', y') - 1/(w \cdot h) \cdot \sum_{x'', y''} T(x'', y'')$$

$$I'(x + x', y + y') = I(x + x', y + y') - 1/(w \cdot h) \cdot \sum_{x'', y''} I(x + x'', y + y'')$$

6) method=CV_TM_CCOEFF_NORMED

상관계수 방법에서 정규화 계수를 나눈 것.

$$R(x, y) = \frac{\sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} T'(x', y')^2 \cdot \sum_{x', y'} I'(x + x', y + y')^2}}$$

따라서 Method 중 본 음표인식에 제일 적합한 것은 CV_TM_CCORR 이라 판단하고 사용하였다. 참고문헌 15) 참조



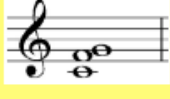
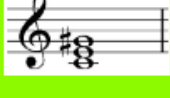
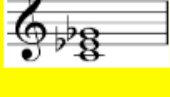
8.1.2. 코드의 종류 : 참고문헌 14) 참조

3 화음(Triad/트라이어드)

3 화음은 루트 + 3 도 + 5 도 의 음정으로 만들어 지고 Major Triad, Minor Triad, Suspended Fourth, Augmented, Diminished 총 5 개의 종류가 있다.



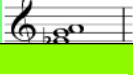


이들 구성음중 3 도의 음정에 따라 메이저,마이너를 구분하게 되고 이 음을 성격음 이라 부른다.

Suspended Fourth 는 완전 4 도의 음이지만 증 3 도로 이해하고 트라이어드 코드에 포함한다.

Root	3rd	5th	표기	읽기	구성음
C	장3도	완전5도	C	C, C Major, C Major Triad	
	단3도	완전5도	Cm	C Minor, C Minor Triad	
	증3도 (=완전4도)	완전5도	Csus4	C Suspended Fourth	
	장3도	증5도	Caug, C+5, C+	C Augmented	
	단3도	감5도	Cdim, Co	C Diminished	

4 화음(Seventh Chord)

4 화음은 루트 + 3 도 + 5 도 + 7 도의 음정으로 만들어지고 기본적으로 3 화음에서 7 도의 음정이 추가된다.

Root	3rd	5th	7th	표기	읽기	구성음
C	장3도	완전5도	장7도	CMaj7, CM7, C△7등	C Major Seventh	
	장3도	완전5도	단7도	C7	C Seventh	
	장3도	완전5도	장6도	C6	C Sixth	
	단3도	완전5도	장7도	CmM7, Cm△7	C Minor Major Seventh	
	단3도	완전5도	단7도	Cm7	C Minor Seventh	
	단3도	완전5도	장6도	Cm6	C Minor Sixth	
	증3도 (완전4도)	완전5도	단7도	C7sus4	C Seventh Suspended Fourth	
	장3도	증5도	단7도	Caug7, C7+5, C7+	C Augmented Seventh	
	단3도	감5도	단7도	Cm7b5, CØ	C Minor Seventh Flat Five, C Half Diminished	
	단3도	감5도	감7도 (=장6도)	Cdim, Cdim7, Co	C Diminished Seventh, C Diminished	

8.1.3. 음별 주파수

기준음 440Hz 를 A4 라고 한다. 한 옥타브 차이가 나는 음은 주파수가 2 배 차이가 나고, 한 옥타브에는 12 개의 음이 존재한다. 각 음들 사이의 주파수 비율은 일정하다.

옥타브	0	1	2	3	4	5	6	7	8
도(C)	16	33	65	131	262	523	1047	2093	4186
도#(C#)	17	35	69	139	278	554	1109	2218	4435
레(D)	18	37	73	147	294	587	1175	2349	4699
레#(D#)	20	39	78	156	311	622	1245	2489	4978
미(E)	21	41	82	165	330	659	1319	2637	5274
파(F)	22	44	87	175	349	699	1397	2794	5588
파#(F#)	23	46	93	185	370	740	1475	2960	5920
솔(G)	25	49	98	196	392	784	1568	3136	6272
솔#(G#)	26	52	104	208	415	831	1661	3322	6645
라(A)	28	55	110	220	440	880	1760	3520	7040
라#(A#)	29	58	117	233	466	932	1865	3729	7459
시(B)	31	62	124	247	494	988	1976	3951	7902

8.2. 추가 진행 및 개선 방안

- UI 개선

본 어플리케이션의 메인 화면 및 각종 기능의 페이지들의 디자인 및 UI를 개선해야 할 필요가 있다.

- 코드 Database 및 검색기능 구축

사용자가 연주하고 싶은 노래를 인터넷에서 직접 찾는 것이 아닌 어플리케이션상에서 검색하여 바로 연습할 수 있도록 노래의 코드파일을 Database에 저장하여 검색하고 바로 연습할 수 있도록 구축한다. 이는 본 어플리케이션의 수익모델이 될 수 있다.

- 회원가입 및 로그인 기능

회원가입 및 로그인 기능을 구현하여 각 회원들이 각자 보유한 연습곡을 로그인을 통해 연습할 수 있도록 한다.

- App store 등록

본 어플리케이션의 개선이 완료되면 Android app store에 등록한다.

8.3. 프로젝트 진행하면서 느낀 점

일을 진행하는 데 철저한 사전 조사와 계획이 필요하다는 것을 느꼈다. 조사가 미흡하면 예상치 못한 오류나 난관에 부딪힐 확률이 증가하고, 대처 방법을 찾는 데에도 시간이 많이 소요된다.

또한 데이터를 가공하여 원하는 형태로 바꾸어 쓸 수 있도록 만드는 것이 쉽지 않다는 것을 느낄 수 있었다. 특히 아날로그 데이터를 분석하는 동안 자연계에 존재하는 잡음으로 인해 분석 과정이 영향을 받을 수 있다는 것을 알게 되었다. 그리고 팀원간 역할 분배가 적절히 되지 않으면 프로젝트가 원활히 진행되는 데 차질이 생길 수 있다는 것을 알게 되었다.

8.4. 다음 학기 수강생들을 위해 제안할 점

프로젝트를 진행하면서 진행 전 계획의 중요성을 많이 느꼈다. 진행 중 부딪혔던 장애요소들과 그로 인한 계획 수정과 일정의 차질 등은 계획과 사전조사 더욱 철저히 되었다면 발생하지 않았을 문제들이 많았다. 다음 학기의 수강생들은 부디 사전조사와 계획에 더 심혈을 기울여서 이런 시행착오를 겪지 않았으면 좋겠다.

9. 참고 문헌

- 1) J.W. Cooley, P.A.W. Lewis, and P.D. Welch, "Historical notes on the fast Fourier transform," Proceedings of the IEEE 55, no. 10 (1967): 1675-1677.
<http://dx.doi.org/10.1109/PROC.1967.5959>
- 2) 위키백과 "장 3 화음",
<https://ko.wikipedia.org/wiki/%EC%9E%A53%ED%99%94%EC%9D%8C> (2017.06.07. 방문)
- 3) 위키백과 "단 3 화음",
<https://ko.wikipedia.org/wiki/%EB%8B%A83%ED%99%94%EC%9D%8C> (2017.06.07. 방문)
- 4) 나무위키 "7 화음", <http://namu.mirror.wiki/namu/7%ED%99%94%EC%9D%8C>
(2017.06.07. 방문)

- 5) 식스라인 통기타이야기 다음카페 "[03 강] 서스펜디드/오그먼트드/슬래시 코드",
http://m.cafe.daum.net/mhguitar/EKwC/3?q=D_jDGwPIH16FY0& (2017.05.28. 방문)
- 6) Unity Documentation "Audiocource.GetSpectrumData",
<https://docs.unity3d.com/ScriptReference/AudioSource.GetSpectrumData.html>
(2017.05.27. 방문)
- 7) Unity Documentation "FFTWindow",
<https://docs.unity3d.com/ScriptReference/FFTWindow.html> (2017.05.27 . 방문)
- 8) OpenCV Documentation "Basic Thresholding Operations"
<http://docs.opencv.org/3.0-last-rst/doc/tutorials/imgproc/threshold/threshold.html#basic-threshold> (2017.05.07
방문)
- 9) 위키백과 "허프 변환",
https://ko.wikipedia.org/wiki/%ED%97%88%ED%94%84_%EB%B3%80%ED%99%98
(2017.05.02. 방문)
- 10) 나인섭 외 1 명, "악보인식 전처리를 위한 강건한 오선 두께와 간격 추정 방법",
Journal of Internet Computing and Services(JICS) 2015. Feb.: 16(1): 29-37 - 오선
제거 vcl
- 11) 싸이월드, "악보 인식"
<http://cy.cyworld.com/home/38996462/post/4BBFFB935E15792B09768401>
(2017.04.22. 방문)
- 12) 백병동, "화성학", 수문당(2012), p.40-151
- 13) Enox software, "OpenCVforUnity Document"
<https://enoxsoftware.github.io/OpenCVForUnity/3.0.0/doc/html/annotated.html>
(2017.06.02. 방문)

- 14) 나눔일기, "코드의 종류와 표기법", <http://spmusic.tistory.com/95> (2017.06.04. 방문)
- 15) OpenCV Documentation, "matchtemplate"
http://docs.opencv.org/2.4/modules/imgproc/doc/object_detection.html?highlight=matchtemplate#matchtemplate (2017.05.25 방문)
- 16) 김동근, C++ API OpenCV 프로그래밍, 가메출판사(2015), p.177-203
- 17) Robert Laganier, OpenCV Computer Vision Application Programming Cookbook,
2nd Edition, PacktPublishing(2013), p.276-314