

# Page Turning Over System For Musicians

2019 공학프로그래밍  
TERM PROJECT

2019. 12. 22

기계시스템디자인공학과  
19110003 김상민  
16100228 최재혁  
17100087 문재균  
18100748 송민규

## 목 차

<b>1</b>	<b>주제 개요</b>	<b>2</b>
	1.1 개발 배경 .....	2
<b>2</b>	<b>코드 block diagram</b>	<b>3</b>
	2.1 코드 블록다이어그램 .....	3
<b>3</b>	<b>코드 설명</b>	<b>4</b>
	3.1 mainfinal.m (메인코드) .....	4
	3.2 LineFinding.m (사용자 정의 함수) .....	7
	3.3 noteselect.m (사용자 정의 함수) .....	8
	3.4 Note2Frequency.m (사용자 정의 함수) .....	12
	3.5 SoundRecord.m (사용자 정의 함수) .....	12
	3.6 Sound2Frequency.m (사용자 정의 함수) .....	13
	3.7 ComparisonF.m(사용자 정의 함수) .....	14
<b>4</b>	<b>개발 결과</b>	<b>15</b>
	4.1 개발 성과 및 활용도 (1) .....	15
	4.2 개발 성과 및 활용도 (2) .....	16
	4.3 개발 성과 및 활용도 (3) 및 코드 실행 결과 .....	17
<b>5</b>	<b>결론</b>	<b>18</b>
	5.1 잘 수행된 부분에 대한 분석 .....	18
	5.2 미흡한 부분에 대한 분석 .....	20
	5.3 참고 문헌 및 자료 .....	21

## 1. 주제 개요

### 1-1. 개발 배경



·노 트리오 2악장-김정원 주미강(Schubert Piano Trio 2nd mov)

그림 1 - 피아노 트리오 연주 장면 / 사진 출처 - <https://youtu.be/skziQ7mZdJU>

관현악에서 트리오나 반주자의 경우, 짧은 시간에 곡을 준비하여 연주하는 경우가 많다. 특히 피아노와 같이 양손이 바쁘고 쉬지 않고 연주하는 악기는 옆에 한 사람이 더 앉아있는 모습을 볼 수 있다. 일명 '넘돌이'라고 불리는 사람인데, 연주가 시작되면 옆에서 대기하다 연주자 대신 악보를 넘겨주는 역할을 한다. 이 역할은 우선 복잡한 악보도 볼 수 있어야한다. 또한 연주자가 원하는 타이밍에 재빠르게 악보를 넘겨줘야 하는 임무를 갖고 있다. 클래식뿐만 아니라 밴드 음악에서도 반주자들 또한 연주할 때 악보를 펼쳐두고 연주를 진행하지만, 혼자 악보를 직접 넘겨야하는 부담을 안고 연주를 하는 경우가 많다.

따라서 우리팀에서는 이번 공학프로그래밍을 통해, 실생활에서 필요한 프로그램을 MATLAB이라는 분석 프로그램을 바탕으로 제작하기로 하였고, 문제점을 해결하기 위해 '연주자를 위한 자동 page turning over 시스템'을 주제로 선정하여 진행하였다.

## 2. 코드 block diagram



그림 2 - 알고리즘 과정

### 3. 코드 설명

#### 3-1. mainfinal.m (메인코드)

```
clear;
clc;
LineLocation=[];
NoteLocation=[];
LineLocation1=[];
NoteLocation1=[];
LineLocation = LineFinding('ice.png');           % 위치를 [a]로 출력
NoteLocation = noteselect('ice.png');           % 음표의 좌표를 [x y]로 출력
LineLocation1 = LineFinding('littlestar.jpg');    % 다른 악보에 대해 반복
NoteLocation1 = noteselect('littlestar.jpg');
SoundVectorOfScore = Note2Frequency(LineLocation, NoteLocation); %해당
                                                    soundvector 추출

imshow('ice.png')
page=0;

%% 음향 분석
% fft에 소요되는 시간 측정 필요
% 데이터 녹음에 소요되는 시간 측정 필요
% 녹음
% recoder객체 생성
a = 1;           % 페이지가 넘어가기 전까지 record와 비교 수행
count = zeros(1,2); % (1)에는 몇 번 일치했는지 기록, 2에는 연속 녹음 진행 횟수
기록
initialRecordingTime = 60/90; % 원하는 Recording 주기
RecordingTime = initialRecordingTime; % 실제 Recordingtime 초기화
SoundVectorOfRecorder = [];
waitingCreteria =(length(NoteLocation)-length(SoundVectorOfScore))*2/3;
% 마지막 줄에 도착하기 전, 같은 phrase에 반응하지 않도록 대기
countMatch = 0;

while a == 1 % 녹음 반복구간
tic % 데이터 처리시간 확인 / 녹음시간 체크 timer On
SoundData = SoundRecord(RecordingTime);
SoundVectorOfRecorder = [SoundVectorOfRecorder Sound2Frequency(SoundData)];
```

```
if length(SoundVectorOfRecorder(1,:))<3
count(2)= count(2)+1;
continue
else
```

#### %% 비교 과정

% 일정시간 연주된 후, 논리를 비교하도록 설정

```
[countMatch,count(2)]= ComparisonF(SoundVectorOfScore, SoundVectorOfRecorder,
count);
if (count(2) > waitingCreteria) && (countMatch>=1)
% 빠른 test용은 count(2) > waitingCreteria를 count(2)> 0으로 수정
% 실전용은 count(2)> 0을 count(2) > waitingCreteria로 수정
count(1) = count(1)+1; %연속적으로 음이 일치하는 정도 check
fprintf('count(일치횟수, 음 감지 횟수) = (%d,%d)\n',count(1),count(2))
if count(1) >= length(SoundVectorOfScore(1,:))*0.8
page = page+1;
count = [0 0]; % 다음페이지 count 초기화
if page == 2 %마지막 페이지에서 끝냄
disp('FINISHED')
break
end
toc; % 작동되는 timer 초기화
close all; % 열려있는 악보 닫기
disp('nextpage')
imshow('littlestar.jpg') % 다음 악보 열기
SoundVectorOfScore = Note2Frequency(LineLocation1, NoteLocation1);
continue
end
else
count(1) = 0; % 틀리면 다시 count(1)을 0으로 초기화
end
end
```

```
time = toc; % 녹음 주기 저장
fprintf("녹음시간 %f\n",time);
timeover = time - initialRecordingTime; % 녹음 시간 보정을 위한 timeover 계산
```

```

if timeover<10^-3
continue;
else
%           균일화를 위해 가중치를 1보다 작은 가중치로 진동 없이 수렴시킨다.
RecordingTime = RecordingTime - 0.75*timeover;
end
end

```

### 3-2. LineFinding.m (사용자정의 함수) - 오선 찾기

```

function [y] = LineFinding(filename)

Image1 = imread(filename);           %이미지 불러오기
ImageGray = rgb2gray(Image1);        %RGB를 회색조로 변환
ImageBW = imbinarize(ImageGray,0.6); %회색조 이미지를 이진화
ImageRBW = (ImageBW==0);              %흑백 반전
mask1 = ones(1,20);                  % 가로로 긴 마스크 생성
ImageLine = imerode(ImageRBW, mask1); %침식을 이용한 백색 가로선 제외 제거
ImageThinLine = bwmorph(ImageLine,'thin'); % 이미지를 세선화
temp1 = sum(ImageThinLine');          % sum은 열방향으로 합을 구함
                                         따라서 transpose해서 행의 합을 구함

[a b] = size(Image1);
temp2 = temp1>0.5*a;                  %이미지의 행 길이의 50%보다 짧은 선 제거
y = find(temp2);                      %가로선이 존재하는 y축의 좌표 추출
end

```

### 3-3. noteselect (사용자정의 함수) - 음표 찾기

```
function result = noteselect(filename)
```

```
%      , 이진화, 반전
```

```
img = imread(filename);           %이미지 불러오기
imgGRAY = rgb2gray(img);          % RGB에서 흑백조
imgREV= imcomplement(imgGRAY);    %흑백 반전
imgFNL=imbinarize(imgREV);         %이진화
```

```
%모서리 무늬 지우기
```

```
[imgX, imgY] = size(imgFNL);
imgFNL(1:3,:) = 0;
imgFNL(imgX-3:imgX,:) = 0;
imgFNL(:,1:3) = 0;
imgFNL(:,imgX-3:imgX) = 0;        %모서리 3번째까지 0으로 만든다.
```

```
se1 = strel('line', 50, 0);        %가로로긴 선을 이용해 오선 검출
IMG= imopen(imgFNL,se1);
IMG_thin = bwmorph(IMG, 'thin');
```

```
IMG1 = bwmorph(imgFNL,'thin');
IMG_Re = imreconstruct(IMG_thin, IMG1);
IMG_sub = IMG_Re - IMG_thin;       %오선 빼내기(기존악보에서 오선만 빼기)
```

```
IMG_BR = bwmorph(IMG_sub, 'bridge'); %오선빼고 끊긴 사이 1개 간격 매꾸기
```

```
se2 = strel('disk',1);             % 머리가 octagon크기 3보다 작으면 검출이 안되므로 팽창시킨다.
```

```
imgimg = imdilate(IMG_BR,se2);
```

```
imgimgimg = bwmorph(imgimg, 'thin'); % 선을 얇게함
```

```
IMG_holes = imfill(imgimgimg, 4, 'holes'); % 구멍채우기
```

```
%%      음자리표 길이차이로 구하기
```

```
% 오선 개수 검출
```

```
THIN_number = bwlabel(IMG_thin);
```

```
MAX_thin = sum(THIN_number');
```

```
MAX_thin_one = nnz(MAX_thin);
```

```
MAX_thin_number =5*round(MAX_thin_one/5);%튀는 값이 존재하므로 평균으로 구함
```

```
%오선 사이 간격 구하기
```

```
Pos_line1 = find(sum((THIN_number==1)));
```

```
Pos_line5 = find(sum((THIN_number==5)));
```

```
Interval_line = Pos_line5-Pos_line1;
```

```
%구멍만 나오게하기
```

```
hoehoe = imdilate(IMG_holes, se2);
```

```
se3=strel('octagon' ,3);
```

```
IMG_Head22 = imopen(hoehoe, se3);
```

```
IMG_2Hole = imreconstruct(IMG_Head22,IMG_holes);
```

```
IMG_3Hole = imerode(IMG_2Hole,se2); %세로줄만 검출
```

```
% 음표만 남기기 위한 for문에 필요한 값들
```

```
A = bwlabel(IMG_holes);
```

```
MAX = max(max(A));
```

```
A_select_zero = (zeros([imgX,imgY])); %모든 값이 0에서 시작해서 라벨 값이 작으면 합친다.
```

```
cnt=1; % Counter
```

```
for i=1:1:MAX % 라벨 부여하고 하나씩 행합을 사용해 원하는값 검출.
```

```
Select_Label = (A==i);
```

```
HC_select = sum>Select_Label'); %열합을 행합으로
```

```
[LSx,LSy] = max(sum>Select_Label)); %열합으로 각 라벨의 큰 곳을 찾아서 +1위치 부터의 꼬리지움
```

```
Select_Label(:,(LSy+2):imgY) = 0;
```

```
[XX,Line_select] = size(find(sum>Select_Label')==1));
```

```

% 행합에서 1의 개수가 오선길이*0.45보다 작으면 지운다.
if (Line_select < ((Interval_line)*0.45))
    %꼬리가 오른쪽으로 모습이 잡히니 오른쪽을 지우고 행합을한다.
    Select_Label = (zeros([imgX,imgY]));
end
A_select_zero = A_select_zero + Select_Label;
end

A_select_zero = imreconstruct(imbinarize(A_select_zero),IMG_holes);
%검출한 객체들 모양 복원

%% 구멍 존재하는 머리 빼내기
A_select_zero = imdilate(A_select_zero,se2); % 머리검출 쉽게 하기위해 크기를 키움

se4=strel('octagon',3);
IMG_Head = imopen(A_select_zero, se4); %머리만 남김

se5 = strel('disk', 1);
B_select_zero1 = imdilate(IMG_BR, se5); % 구멍있는 머리만 찾기 위한 작업
B_select_zero2 = bwmorph(B_select_zero1, 'skel');
B_select_zero3 = bwmorph(B_select_zero2, 'fill');
B_select_zero4 = bwmorph(B_select_zero3, 'remove');
B_select_zero = imfill(B_select_zero4, 'holes');
IMG_HOLE = imopen(B_select_zero, se4);

IMG_LEFT = imbinarize(B_select_zero-B_select_zero1); % 구멍있는 객체로
% 빈 공간 채운 객체를 빼서 남는 곳 검출
NOTE_HOLE = imreconstruct(IMG_LEFT,IMG_Head); %빈 음표만 가져온다.

P1 = IMG_Head;
P2 = imreconstruct(NOTE_HOLE,P1);

se6=strel('disk',3);

IMG_Head2 = imclose(P1, se6); %너무 각지지 않고 둥글하게 만듦
IMG_Head3 = imclose(P2, se6);

```

```

%%
[L, n] = bwlabel(IMG_Head2); %무게중심 좌표 저장과정
result = zeros(n,2);
for k = 1:n
    [r,c] = find(L == k);
    result(k,1) = mean(c);
    result(k,2) = mean(r);
end

[L2, n2] = bwlabel(IMG_Head3); %빈 노트 무게중심 좌표 저장과정
resultN = zeros(n2,2);
for k = 1:n2
    [r,c] = find(L2 == k);
    resultN(k,1) = mean(c)+1;
    resultN(k,2) = mean(r);
end
result = [result; resultN];
result = sortrows(result); % x축 순서대로 음표들을 정렬
end

```

### 3-4. Note2Frequency (사용자정의 함수) - 음의 좌표를 주파수로

```
function frequency = Note2Frequency(LineLocation, NoteLocation)

Last5Line = LineLocation(end-4:end); % 5개 선의 위치 추출
%음표의 위치를 마지막 줄로 선정하기 위한 기준
LineCriteria = (LineLocation(end-4) - LineLocation(end-5))/2 + LineLocation(end-5);

InnerCriteria = find(NoteLocation(:,2)>LineCriteria); %기준 내 존재하는 음표 추출
NoteInLastLine5 = NoteLocation(InnerCriteria,:); %기준 내에 존재하는 음표 위치행렬

%% 주파수 찾는 과정
% 한 음 당 간격 찾기
Distance2 = abs(sum(Last5Line-Last5Line(5))/10);
Distance = Distance2/2;

% frequency가 총 12개라고 가정 (가운도부터 온음들만 비교)
RepNoteY = repmat(NoteInLastLine5(:,2),1,12);
RepNoteX = repmat(NoteInLastLine5(:,1),1,12);
TempDistance = Distance*repmat([-2:9],length(NoteInLastLine5(:,2)),1);
%선과 비교해서 각 음의 존재여부 파악(열번호는 몇번째 음인지 나타냄)
WhichFrequencyIsON = abs((RepNoteY+TempDistance-Last5Line(end)))<(Distance/2);
frequency = (RepNoteX.*WhichFrequencyIsON)';>0;
```

### 3-5. SoundRecord (사용자정의 함수) - 녹음시행

```
function result = SoundRecord(time)
recorder = audiorecorder(44100,16,1); % 오디오 객체 초기 설정
recordblocking(recorder,time);
%녹음 도중에 스크립트가 진행되지 않도록 hold 하면서 녹음
result = getaudiodata(recorder); % 녹음데이터를 반환
end
```

### 3-6. Sound2Frequency (사용자정의 함수) - 녹음을 주파수로 변환

```
function result = Sound2Frequency(record)
Frequency = abs(fft(record)/length(record)); %주파수별 음압의 크기만 추출
% 쉼표복소수로 인한 대칭부분 함으로 연산
Frequency2 = Frequency(1:fix(end/2)+1);
Frequency2(2:end-1) = 2*Frequency2(2:end-1);
F = 44100*[0:length(Frequency)/2]/length(Frequency);
temp = F(find(Frequency2>max(Frequency2)*0.65));
% 임계치 이상으로 큰 기본 주파수만 뽑아냄
% 얻은 주파수가 어느 음에 일치하는지 12개에 음에 대해 check
tempresult = zeros(length(temp),12) ;
for k=1:length(temp) % 음 별로 나누어 주파수를 확인
reptemp = repmat(temp(k),1,12);
% Octaveband 범위 구하기
expcenterFrequency = [-9 -7 -5 -4 -2 0 2 3 5 7 9 11]/12;
bandwidth = 440*(2.^(expcenterFrequency+1/24)-2.^(expcenterFrequency-1/24));
CenterFrequency = 440*(2.^(expcenterFrequency));
IsFrequency = abs(reptemp-CenterFrequency)<bandwidth;
tempresult(k,:) = IsFrequency;
end

if length(tempresult(:,1))==1
result = tempresult';
else
result = (sum(tempresult'))>0;
end

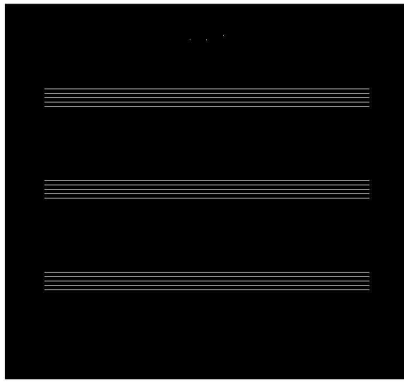
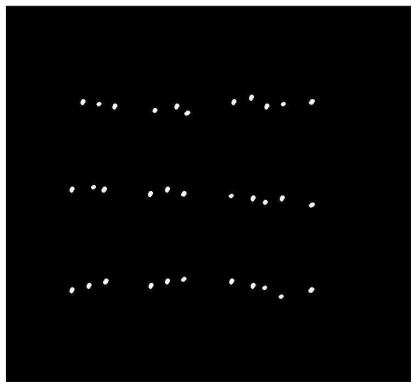
end
```

### 3-7. ComparisonF (사용자정의 함수) - 녹음과 악보 비교 검출

```
function [result1,result2] =
ComparisonF(SoundFromScore,SoundFromRecording,ComparisonNum)
%% 비교 방법 : 3개의 record행에 2개의 score와 일치여부를 비교
% 우선 record를 3개의 행과 비교할 수 있게 score와 record를 복제
% ComparisonNum은 일치횟수를 외부로부터 받아옴 / 1에는 일치횟수, 2에는 연주
횟수 기록
if sum(SoundFromRecording(:,end))~=0 % 음이 연주된 경우에만 비교
    if ComparisonNum(1) == 0
        TempScore = repmat(SoundFromScore(:,ComparisonNum(1)+1),1,3);
    % 악보에서 n번째음을 비교
        TempSound = SoundFromRecording(:,end-2:end);
        IsCorrect = (sum(TempScore == TempSound)>=11);
        else % 연주되지 않았을 때
            TempScore = repmat(SoundFromScore ( : , ComparisonNum(1)+1) ,1,3);
            TempScore = [TempScore ; repmat(SoundFromScore(:,ComparisonNum(1)),1,3)];
            TempSound = repmat(SoundFromRecording(:,end-2:end),2,1);
    % 각각의 음이 11개 이상 일치하는지 확인
        IsCorrect = (sum(TempScore == TempSound)>=22); .
        end
result1 = sum(IsCorrect); % 일치하는 개수 반환
result2 = (ComparisonNum(2)+1); % 음이 연주된 것으로 파악되면 카운트
else % 음이 연주되지 않으면 처음으로 초기화
result1 = 0;
result2 = 0;
end
```

## 4. 개발 결과

### 4-1. 개발 성과 및 활용도 (1) - 악보 인식

1. 오선줄 인식	2. 음표 인식
	
그림 3 - 오선줄 추출 결과	그림 4 - 음표 추출 결과
악보를 이미지 처리 tool 중 침식을 이용해 선만 우선적으로 추출한 뒤, 행 합을 통해 어느 열에 선이 존재하는지 파악하도록 실시하였다.	인식한 오선줄을 이용하여 박자표, 음자리표 등 다른 기호들을 삭제한 뒤, 음표만 남겨 좌표를 추출하였다. 정상적으로 음표만 추출한 결과를 얻을 수 있다.

### 3. 음의 위치를 통한 음의 높이 추정

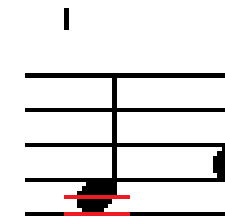


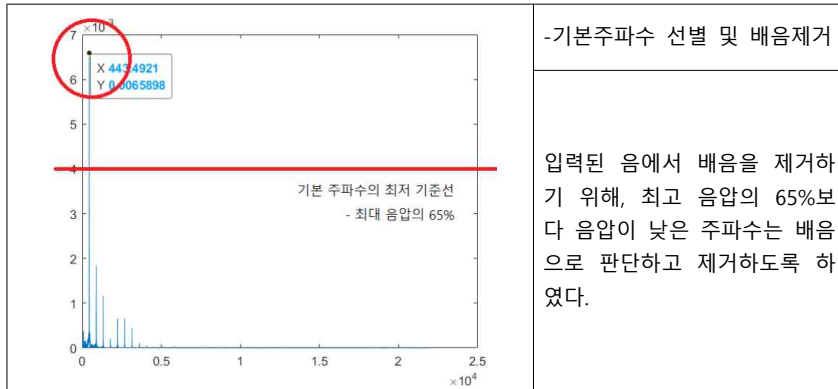
그림 5 음의 위치 추정

- 그림 5와 같이, 오선줄의 선으로부터 음표가 얼마나 멀리 떨어져 있는지를 통해 음의 높이를 추정하였다.
- 이 때, 마지막 줄과 4번째 줄의 간격을 2d로 추정, 음표의 y좌표가 해당 음의 위치로부터 d(그림 5의 두 개의 빨간줄의 간격)보다 가까이에 존재할 경우 그 음에 해당한다고 판정하였다.
- 악보를 넘어가는데 있어서 마지막 다섯 개의 줄만 필요하므로, 악보의 마지막 줄의 음만 추출하여 비교를 실시한다.
- 비교된 음은 하나의 열벡터로 저장되며, 행방향은 C4~G5까지 총 12개의 음을 나타낸다.



## 4-2. 개발 성과 및 활용도 (2) - 음향 인식

### 1. 외부 음에 대한 FFT 분석



### 2. 주파수 to 음 변환

- 주파수로 얻어진 음을 다음 표에 맞추어 음 벡터로 변환시킨다.

음계	C4	D4	E4	F4	G4	A4	B4	C5	D5	E5	F5	G5
중심 주파수 (Hz)	262	294	330	350	392	440	494	523	587	659	740	831

표 1 - 음계의 중심주파수

- 녹음된 음이 중심주파수로부터 bandwidth 내에 존재하면 음이 있다고 판단하도록 하였다.

- 이 때, bandwidth는 중심주파수로부터  $440 \times 2^{\pm 4}$  이다.
- 인식된 음은 열벡터로 기존 열벡터에 연속하여 저장된다.

### 2. 녹음 주기 제어

- 주기를 일정하게 하기 위해 다음과 같은 feedback을 적용시켰다.

$$\text{다음 녹음 시간} = \text{이번 녹음 시간} - 0.75 \times \text{초과된 녹음 시간}$$

- 이 때, 녹음시간이 줄어들 때 동시에 초과 녹음시간도 감소하므로 feedback된 녹음시간이 진동하지 않도록 가중치 0.75를 추가하였다.

## 4-3. 개발 성과 및 활용도 (3) - 비교 분석 / 코드 실행 결과

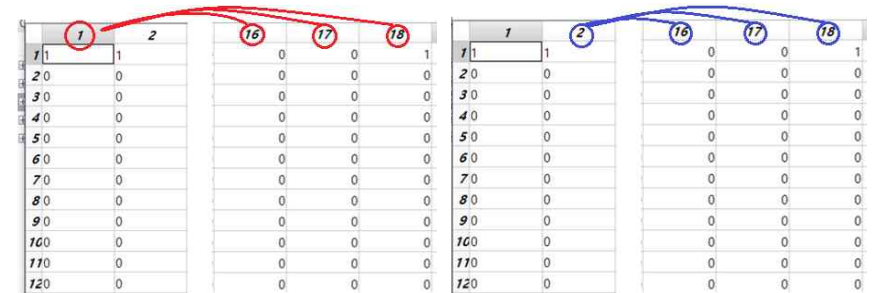


그림 6 1번째 악보의 벡터와 녹음 비교

그림 7 2번째 악보의 벡터와 녹음 비교

- 악보, 녹음 일치 확인은 악보에서 2개의 열벡터, 녹음에서 1개의 열벡터를 비교한다.

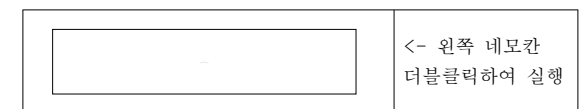
- 악보에서 1개의 벡터를 선택하게 되면 새로 들어온 녹음에 대해 일치여부를 비교한 후, 다음 악보를 비교하게 된다. 하지만, 다음 음이 1회라도 불일치하게 되면 바로 불일치 판정되어 카운트가 초기화 되게 된다.

- 이를 해결하기 위해 악보에서 2개의 벡터를 선택하여 기존 음이 일치한다면 다음 음이 틀리더라도 한 번의 tolerance를 줄 수 있도록 설정하였다.

- 녹음의 경우 3개의 열벡터를 선택하여 새로운 음이 들어오더라도, 기존의 음과 악보의 벡터를 비교하여 일치하면 다음 음을 한 번 더 검사할 수 있도록 tolerance를 주었다.

- 또한 녹음 간격이 정확하지 않아, 앞의 음이 같이 녹음될 수 있으므로 12개의 벡터 성분 중 11개가 일치하면 일치하는 것으로 판정하도록 설정하였다.

위의 설정을 통해, 연주할 때 프로그램상 오류로 인한 악보 인지에 실패하지 않도록 설정하여 성공한 시연을 보일 수 있었다. 다음 영상을 통해 코드를 실행한 시연영상을 확인할 수 있다.

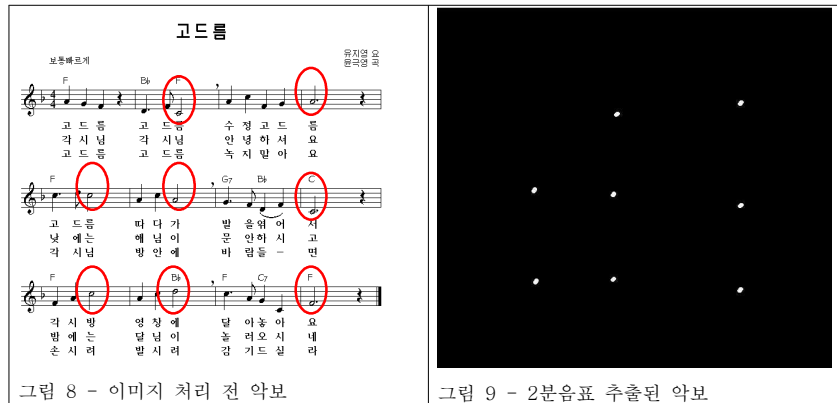


## 5. 결론

### 5-1. 잘 수행된 부분에 대한 분석

#### 1. 악보 인식

- 오선줄을 명확하게 구분하여 찾아내는데 성공하였다.
- 음표 객체를 찾아내어 무게중심을 구하여 음표의 위치를 구하는데 성공하였다. 또한 음표가 2분음표인 경우, 이미지에 빈 구멍이 있는 것을 이용하여 2분음표의 길이만큼 음의 벡터를 추가하였다. 그로 인해 2분음표와 4분음표를 컴퓨터가 구별하여 인식할 수 있도록 설정하였다. (그림8, 그림9 참조)



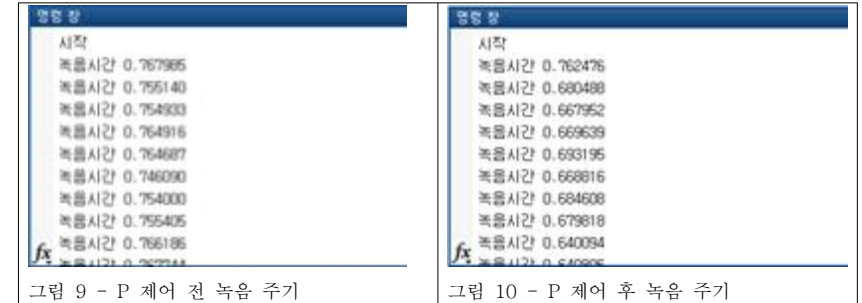
- 악보에서 원으로 인식될 수 있는 글씨, 박자표, 높은음자리표를 위치와 형태를 이용하여 제거함으로써 음표 추출의 정확도가 비교적 높게 나타났다.

- 높은음자리표, 박자, 음표 중에서 음표만 골라내는 작업은 음표 기둥의 기하학적 특징을 이용했다. 음표 선을 세선화 후 행 합을 하면 기둥을 포함한 행은 1로 나타나므로 1의 개수가 일정 수 이하일 때 객체를 지우도록 하였다.

- 8분음표의 경우 객체마다 열 합을 1회 하고 최대값 위치의 오른쪽 값을 0으로 만들어 꼬리를 지우고 행 합을 실행했다. 그리고 팔각형의 마스크를 이용하여 음표 머리를 추출했다.

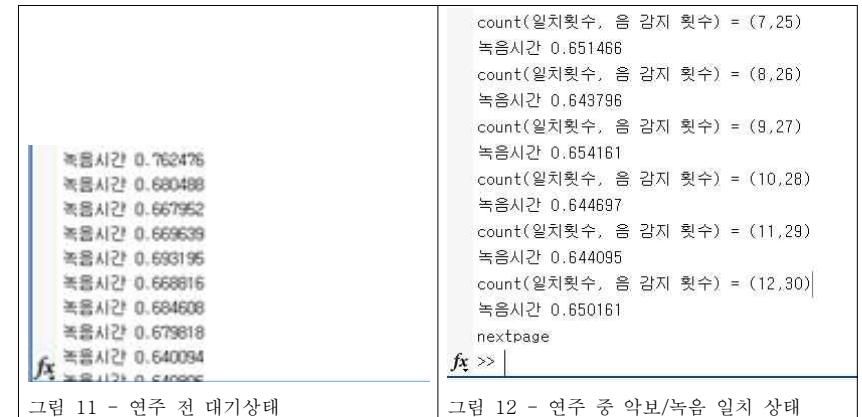
#### 2. 음향 인식

- 직접 악기와 컴퓨터를 연결하여 소리를 받아 녹음하였고, 노이즈가 거의 발생하지 않은 상태로 프로그램을 실행시켜 오작동의 가능성을 줄였다.



- 녹음 시간 외 기타 연산처리 시간으로 인해 녹음 주기에 변동이 오는 것을 막기 위해 코드에 P 제어를 추가함으로써 상대적으로 일정한 녹음시간을 가지도록 설계하였고, 위 그림 9와 비교하여 그림 10과 같이 원하는 녹음주기인 6.66초에 가깝게 수렴하여 잘 작동한 것을 확인할 수 있다.

#### 3. 음 비교 분석



- 연주 대기상태에서 그림 11과 같이 count되지 않고 정상적으로 녹음 대기상태 중에 있는 것을 확인할 수 있다.
- 악기가 연주되기 시작하고, 그림 12와 같이 악보의 마지막줄의 0.8배만큼 일치할 때 정상적으로 nextpage가 출력되는 것을 확인할 수 있다.

## 5-2. 미흡한 부분에 대한 분석

### 1. 악보 인식

- 악보 내 가사, 박자표, 음자리표 등 다양한 악상 기호를 인식하지 못한다. 따라서 현재 프로그래밍 단계에서는 수동적으로 박자를 입력해야하며 반드시 높은 음자리표에 해당하는 음계만 인식하여 연주할 수 있다.
- 또한 정해진 음계 (C4~G5)의 음만 인식할 수 있으며 sharp이나 flat이 생기는 경우 제대로 인식하지 못한다는 한계점을 가진다. 반음이 포함된 음은 구별하여 인식할 수 없다.
- 음표의 머리부분을 가지고 음의 길이를 구별하였으므로 4분음표보다 짧은 음을 제대로 구분할 수 없다. 더 자세히 구분하기 위해서는 8분음표의 꼬리를 인식할 수 있는 방법이 필요하다. 이에 대해서는 꼬리가 서로 연결되지 않은 이미지에 대해 무게중심을 기준으로 '복소 푸리에 해석을 이용한 이미지 인식'을 접목시켜 해석하면 음을 인식하도록 설계할 수 있을 것으로 예측된다.
- 피아노 악보는 대다수 양손 악보로 구성되어 있으나, 멜로디 라인을 기록한 부분만 인식할 수 있다. 이 부분은 아래부터 5개의 라인을 확인하도록 하는 것이 아니라, 10개의 라인에 대해 음을 분석하고 저장하도록 설계하고 음계의 범위를 낮은음자리표의 범위로 넓힌다면 가능할 것으로 예상된다.
- 그 외 다수의 악상기호를 인식하는데 있어 어려움이 많다.

### 2. 음향 인식

- 연주된 음을 지속적으로 연주되었는지 새로 연주된 음인지 판단할 수 없기 때문에 2분음표 하나로 연주된 음과 4분음표 두 개로 연주된 음을 구분할 수 없다. 즉, 측정 시기에 연주되고 있는지 여부만 확인할 수 있다.
- 따라서, 먼저 연주된 음이 다음에 연주된 음 데이터에 상호영향을 줘, 여러개의 음이 빠르게 한번에 연주될 경우 일치하지 않는다고 판단하여 악보가 넘어가지 않는 현상이 발생할 수 있다.

- P-제어를 통해 녹음 주기를 지속적으로 제어하고 있으나, 컴퓨터의 작동 상태에 따라 녹음주기가 급증하거나 급감하여 녹음에 지장을 줄 수 있다. 특히 급격하게 느려지는 경우 기록된 녹음시간이 현재 코딩에서는 0 이하로 적용되어 녹음이 중단되게 된다. 따라서 녹음시간이 0으로 수렴하려는 경향을 보이거나 녹음시간이 0으로 변하는 경우, 다시 초기 녹음시간부터 녹음을 시작하도록 하는 코딩이 필요할 것을 보인다.

- 악보와 마찬가지로, 음 데이터를 저장할 시 반음계의 데이터를 저장하지 않았으므로 반음은 인식할 수 없다는 문제가 발생한다. 이는 음계에 대한 데이터를 처음부터 넓은 spectrum으로 제작하여 (피아노 건반 기준 88개)를 사전에 저장한다면 해결할 수 있을 것으로 판단된다.

### 3. 기타 문제점

- 악보를 보여주고 닫는 명령어 imshow와 close에서 1s에 가까운 시간을 소요한다. 따라서 이를 해결하기 위해 악보 넘어가는 시기를 조절하기 위해 마지막줄에서 어느 지점까지 악보가 일치하도록 대기할 것인지 연주자에 맞춰 조절할 필요가 있다.
- 기반 프로그램이 매트랩이기 때문에 FFT 분석에 있어서 유리한 점이 있으나, 실제 현장에서 사용하려면 노트북과 연결할 라인 등 다양한 장비들이 번잡하게 필요하다. 또한 프로그램의 가격이 매우 비싸기 때문에 다른 프로그램에 이식하여 실행해야하는 문제점을 갖고 있다.
- 마지막으로 손으로 쓴 악보나 악보 표기법을 지키지 않은 악보, scan시 회전된 악보는 인식 불가능하다. 위의 문제점을 고려한 악보 turning system을 제작해야 상용화가 가능할 것으로 보인다.

## 5-2. 참고 문헌 및 참고 자료

1. (Matlab을 이용한) 디지털 영상처리 2판 / Gonzalez, Rafael C 외 2명/ 한국맥그로힐 / 2012 / 9장 형태학적 영상처리
2. (최신) 소음·진동:이론과 실무 / 정일록 저/ 신광문화사 / 2009 / 옥타브 밴드 분석 참조