FINAL REPORT

Image Processing. Team Project 2

TEAM 02 16 **(PERFECT PITCH)**

20164542 김선재 (ELECTRONIC AND ELECTRICAL ENGINEERING, TEAM LEADER)

20141001 이지호 (COMPUTER SCIENCE ENGINEERING)

20141261 송제웅 (COMPUTER SCIENCE ENGINNERING)

20145315 심지선 (BUSINESS ADMINISTRATION)

2018 November 13

FINAL REPORT

Image Processing. Team Project 2

Contents

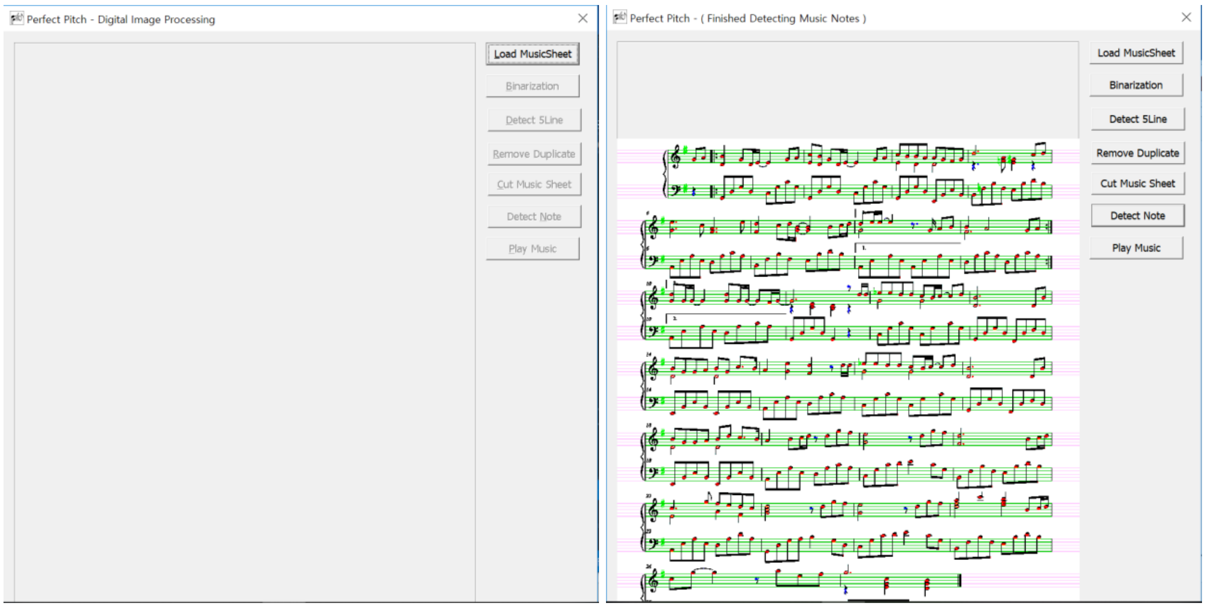
Outline

Overview

Project description & Screenshots …

The goal of the program 'perfect pitch' is to allow the computer to recognize the image of a 'score' and to play the music containing the score itself.

The input will be an image of music score, and the output is music played by a computer.



< Picture 1. Detection Result >

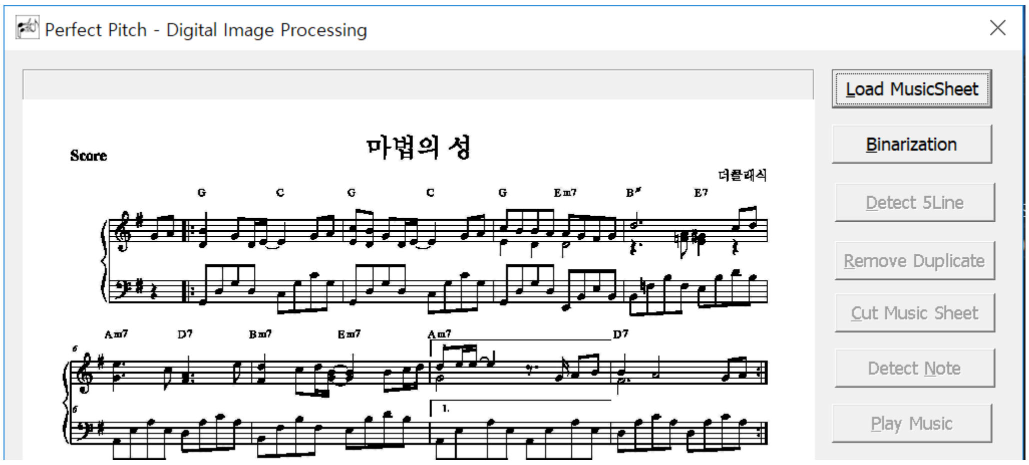
Program Definition

With this program, we don't have to spend a lot of time reading the score when practicing the instrument. At the same time, we will be able to spot the information that the composer intended without missing. If you want, you will be able to instantly appreciate music you want to hear only with the score.

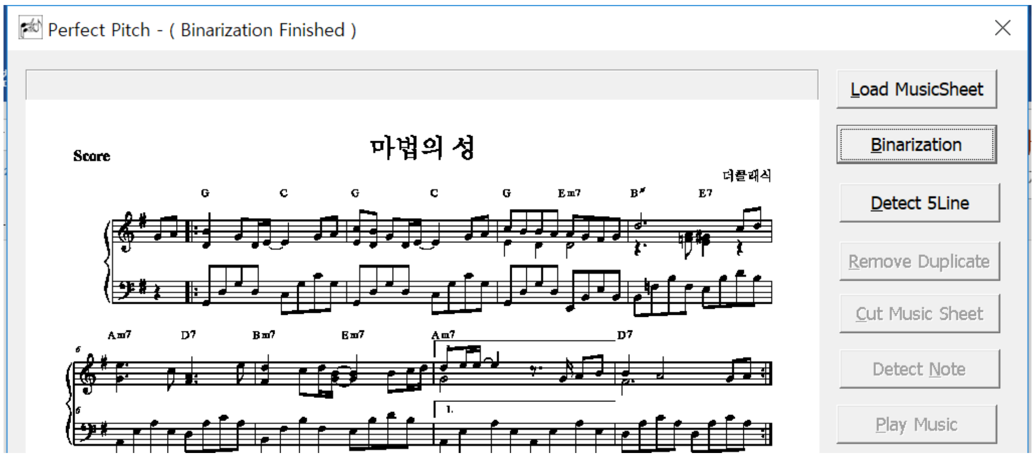
Main functions

1. Binarization

Load music sheet image file and make them into binary, black and white pixel under threshold conditions.



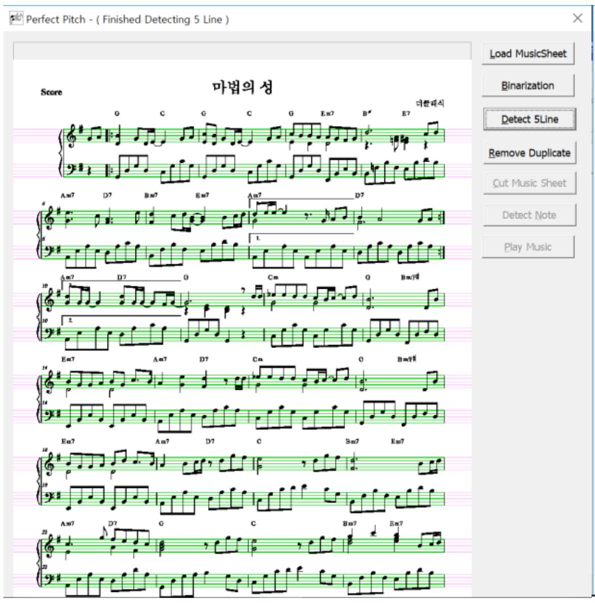
< Picture 2. Picture Before Binarization >



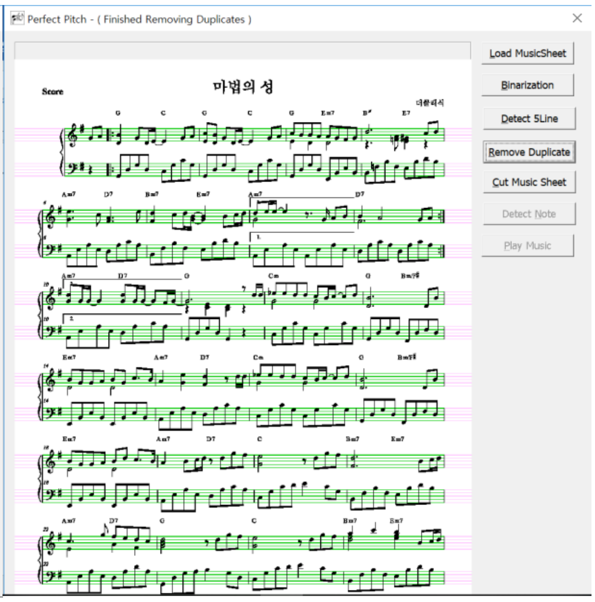
< Picture 3. Picture After Binarization >

2. Detecting five lines of a music score & Removing duplicated five lines

Make a histogram of the black pixel from the y-axis. 5 lines are recognized under the threshold condition. The height is then calculated to clear the overlaid lines.

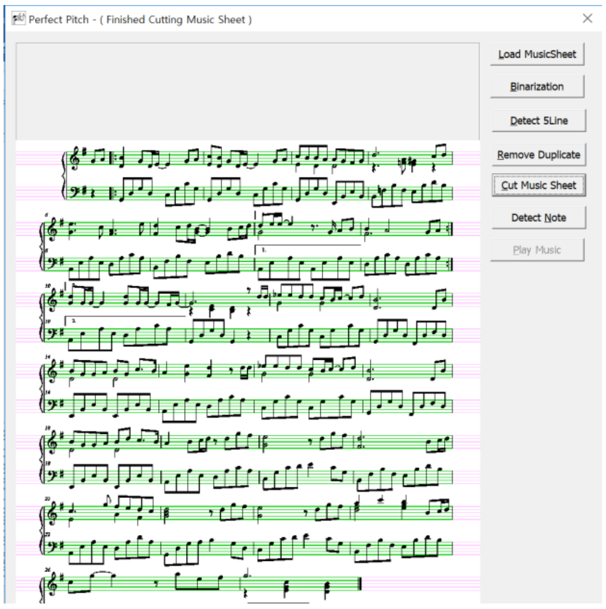


< Picture 4. Picture After Detecting 5 lines >

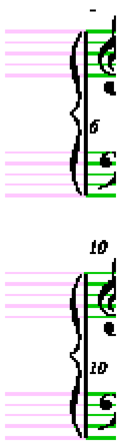


< Picture 5. Picture After Removing Duplicate pixel >

3. Cutting staves



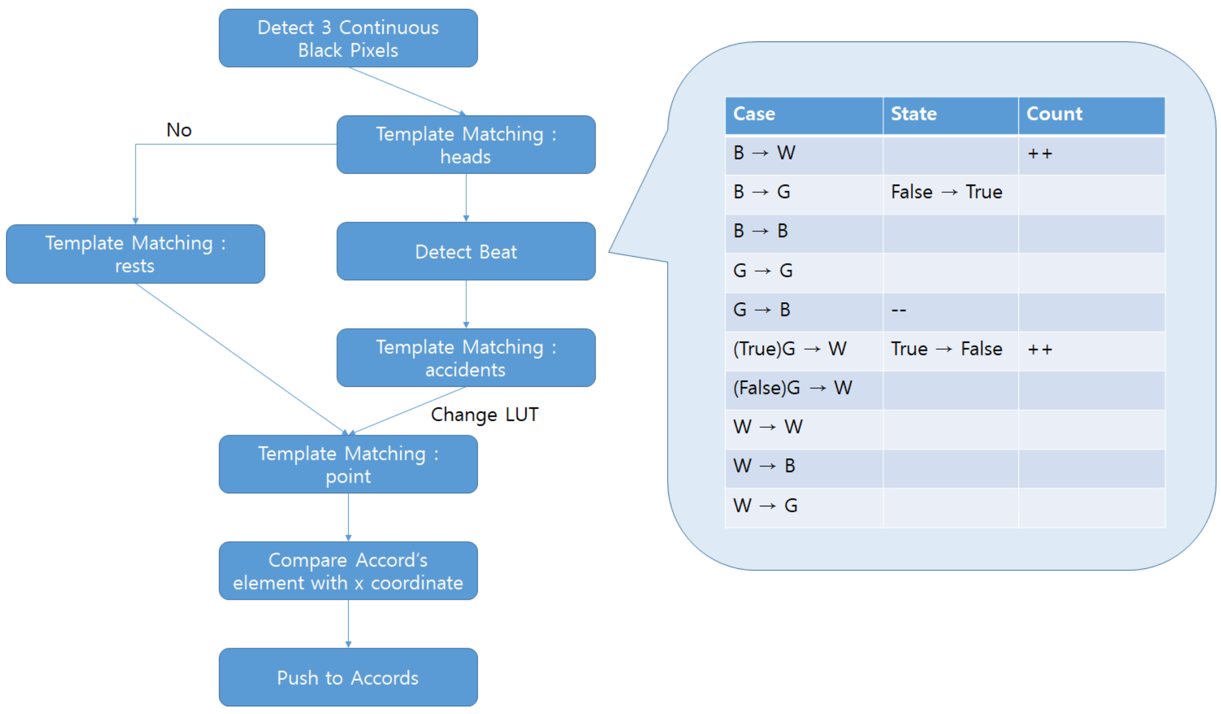
< Picture 6. Picture After Cutting the Staves >



< Picture 7. Picture Cutting Stave by '{' symbol >

The program's algorithm of cutting the music sheet's each stave is to detect the { symbol on the left side. Every stave have those symbol, so program detects the symbol and and finds the middle part of it and cut into each line, distinguishing left hand and right hand part.

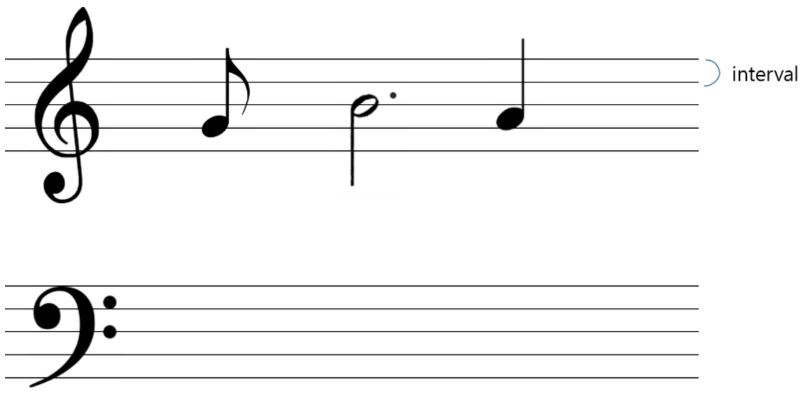
4. Detecting notes of each staves


Detect 3 Continuous Black Pixels
Template Matching : heads
No
Template Matching : rests
Detect Beat
Template Matching : accidents
Change LUT
Template Matching : point
Compare Accord's element with x coordinate
Push to Accords

Case	State	Count
B → W		++
B → G	False → True	
B → B		
G → G		
G → B	--	
(True)G → W	True → False	++
(False)G → W		
W → W		
W → B		
W → G		

< Picture 8. The Flow of Detecting algorithm >

A search of musical note is made as shown in the following flowchart. Simply detecting through template matching makes program operation very slow, so exploration of the notes is carried out at the section where the three black dots are connected (for detecting all objects except for 5 lines, since the thickness of the 5 lines is 2 pixels). Furthermore, all of the ranges used for navigation were made available for a wide variety of music sheets, using interval (= distance between two lines close to each other during the wrong time).



< Picture 9. The Composition of Five lines >

1) Detecting Heads

The matching channel is set to green (channel that has processed 5 line with 125). In the template matching process, we counted 'black and gray' and 'gray and white' pixel value as well as exact black pixel value with the template. This is to prevent the low matching rate caused by the misalignment of the part that overlaps with the 5 line of the sheet.

The first priority was to detect the notes(head) which are expected to have a highest number, and then to detect rest symbols. When an object has detected, we changed the color of only one channel to avoid redundant navigation. By using this technic, we could reduce the time consumption on navigation without using labeling (without using dynamic discounts).

Like the following table, we changed the detected objects' color to eliminate Template Matching errors in the occasion of overlapped searches, exception processing, and navigating of object on the lines.

Objects	R	G	B
Head	255	0	0
Point	255	0	0
Accident	0	255	0
Score Lines	0	125	0
Rest	0	0	255

< Picture 10. Color table of detected objects >

2) Detecting Location of UP/Down Notes and Stems

We figured out if there is a straight line at the lower left and upper right of the note and remember its Up/down information and the coordinates of the note's stem. If the stem does not exist, the note will be a whole note.

3) Flag Navigation (Beat Detection)

After a note or rest symbol detection is done, the beat detection will begin. Based on the Up/Down note and note's stem coordinate information obtained in the previous step, the program navigates left/right in a descending process starting from the end of the stem. Count the changes of black and white, but the program does not count the 5 line. For the notes, identify whether the notes are empty or black and check if there is a tail upright or down-left. The case charts up above were made because the 5 lines were changed to green in the pre-treatment process.

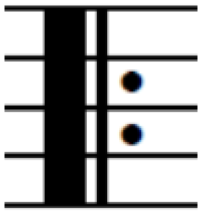
And finally, check the number of branches derived from the tail to identify the type of the notes. For both the notes and the rest symbols, we check if there is a point at a certain distance to the right and give it 1.5 times the beat. In the case of a note, we detect the possible accidents that may have happened additionally. We checked only the certain distance on the left side of the note.

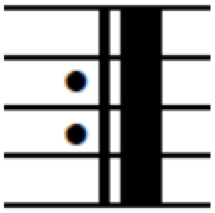
4) Detecting and Processing Sharp, Flat and Natural

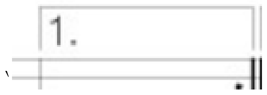
First, we've made a LUT that replaces the y-coordinate of the basic head with a pitch of music, and then we've copied and used the default LUT in linearScore. Using the template matching, set the scale and correct the copied LUT. Program set the scale of the initial score and correct the copied LUT by template matching. While searching for the notes, search for the sharp, flat and natural symbols, then continue to modify the LUT. In the case of the natural symbol, use the original LUT to make it as initial pitch.

This function is processed simultaneously in the Accident function.

5) Repeat Mark Navigation Algorithms


Repeat Mark [1]


Repeat Mark [2]



Jump Mark [3]

< Picture 11. The Repeat Marks >

If the above symbols are found through the initial template matching, the pitch shall be recognized as a note with an unusual number, such as -100,-101,-102, and placed in the temporary sheet. When all the work is done and the temporary sheet will be copied to the final score, and perform the following actions.

- **Normal musical note found**: push to new final core

- **Repeat Mark (1) found**: Store the index, do not push in final core

-**Jump Mark (3) found (odd number case)** : Set jumpstate to TRUE and save its index (jumpIndex). (Program saves and doesn't push.

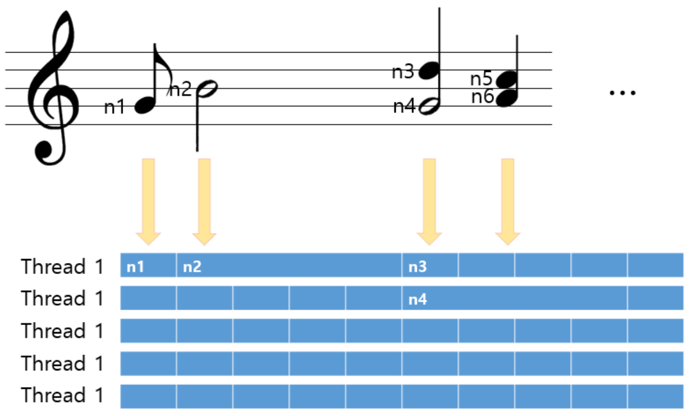
- **Repeat Mark (2) found**: By setting the first RightRecurIndex to -1, if you meet 2 without detecting Repeat Mark 1, program repeats it from the beginning.

5. Play the processed score

- Usage of Midi in mmsystem.h

The idea was to deliver pitch levels to the SendShortMsg() method to make the sound of the music. One thread has a pitch vector as a member variable, and then moves it one space from the very beginning of the vector to the value. It was implemented in multi-threads so that it could make a sound of multiple notes at the same time.

Therefore, the following data structures are required:



< Picture 12. Picture Data structure to make a sound of multiple notes >

However, because this kind of musical note reading procedure,, the red arrow part is made into a model called 'Accord' and is pushed to a vector type Accord each time the program reads an accord. When a new note is found, compare to the most recent x coordinate of Accord. If it is similar, then add to the last element in Accord, otherwise will treated as a new Accord element.

Then, when the music sheet processing has completed, convert the values of the vector into a certain data structure that matches the thread.

Design

Class

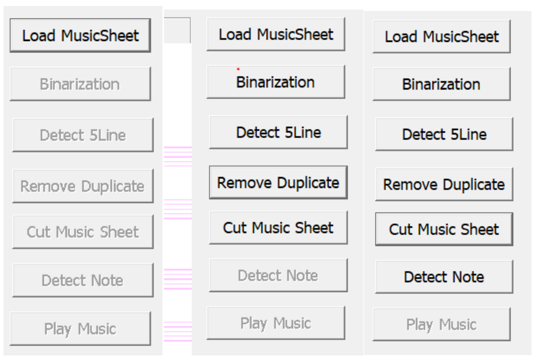
Description of C++ Class Hierarchy

Feature	File Name	Description
UI & Main Function	PerfectPitch-UI.cpp	It is the main function of the program and makes the GUI of it by using MFC (C++ Library
	PerfectPitch-UIDlg.cpp	
Processing Music Sheet	Pretreatment.cpp	Binarization, detect 5 lines and remove duplicate pixels of the line
	LinearScore.cpp	It cuts the music sheets into staves by right hand score and left hand score.
Detect the Notes, Calculate and Push	ScoreProcessor.cpp	Detect all the music notes and beats by template matching and calculating pixels
	Accord.cpp	Find the exact pitch of music notes by calculating the distance of pixel with the 5 lines
	Score.cpp	Push all the scores in to the threads
Playing Music	Thread.cpp	Play music.
	Midi.cpp	C++ library to produce the musical sound

UI

Used MFC library to visualize the program

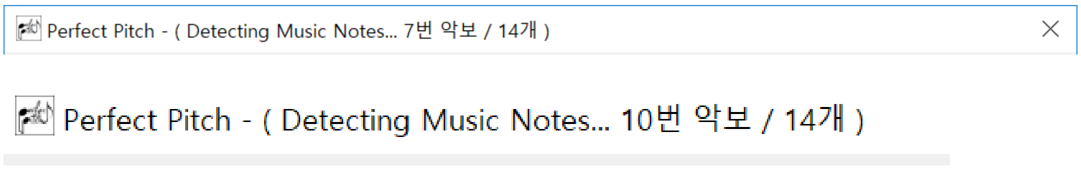
- Activate and Deactivate Buttons



< Picture 13. Activation of Buttons >

To avoid Error, the program first deactivates all the buttons except loading file button. Then we activate one by one in each step. We made program like this to avoid users to click detecting note button before binarization gets done.

- Visualize the Progress of the program



Perfect Pitch - (Detecting Music Notes... 7번 악보 / 14개)

Perfect Pitch - (Detecting Music Notes... 10번 악보 / 14개)

< Picture 14. Visualization of work progression >

We visualized the progress of the step which takes a pretty long time on the status bar. It shows during the process of cutting staves and detecting musical notes (the part that takes long time to be done)

Performance

Spending time of the Program

Bin $$O_{bin}(n) = (row\ * col)$$

Detect Line $$O_{D.L}(n) = (row\ * col)$$

Linear Score $$O_{L.S}(n) = (row\ * col)$$

Template matching $$O_{T.M(i)}(n) = (Template(i).row\ * Template(i).col)$$

Detect Object $$O_{D.O}(n) = \left(num(Object) * \sum O_{T.M(i)} \right)$$

Control Flow $$O_{C.F}(n) = \left(num(Object) \right)$$

Total $$O(n) = O_{T.M(i)} + O_{D.L}(n) + O_{L.S}(n) + O_{T.M}(n) + O_{D.O}(n) + O_{C.F}(n)$$

< Picture 15. Time Complexity >

Spending real time

1. Binarization : 4.613000 sec

   Perfect Pitch - (Binarization Finished, 4.613000 sec)

2. Detecting five liens of a music score : 1.026 sec

   Perfect Pitch - (Finished Detecting 5 Line, 1.026000 sec)

3. Removing duplicated five lines : 0.000 sec

   Perfect Pitch - (Finished Removing Duplicates, 0.000000 sec)

4. Cutting staves : 21.828 sec

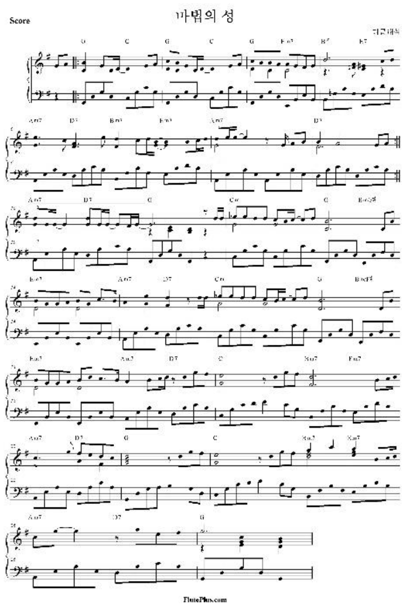
   Perfect Pitch - (Finished Cutting Music Sheet, 21.828000 sec)

5. Detecting notes of each staves : 61.264 sec

   Perfect Pitch - (Finished Detecting Music Notes, 61.264000 sec)

Time spent : Detecting notes > Cutting staves > Binarization > Detecting five lines > Removing duplicated line

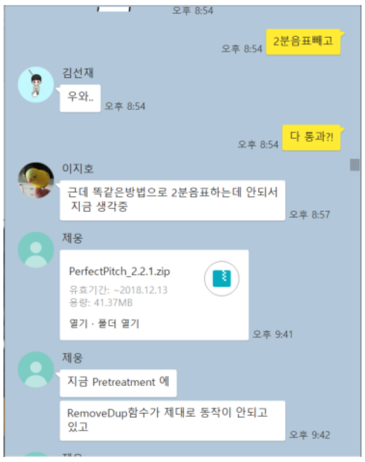
By Processing on the following score image



< Picture 16. The sample score image, The Castle Of Magic >

Meeting

Schedules and Pictures (Kakaotalk & real picture)



오후 8:54

2분음표빼고 오후 8:54

김선재
우와.. 오후 8:54

다 통과?! 오후 8:54

이지호
근데 똑같은방법으로 2분음표하는데 안되서
지금 생각중 오후 8:57

제웅
PerfectPitch_2.2.1.zip
유효기간: ~2018.12.13
용량: 41.37MB
열기 · 폴더 열기 오후 9:41

제웅
지금 Pretreatment 에
RemoveDup함수가 제대로 동작이 안되고
있고 오후 9:42

< Picture 17. Meeting pictures >

Team's thoughts

At first when we chose a topic, we thought that processing a music sheet into the piano sound was a very fun and challenging topic. But when we tried to code the program, we realized it was a very complicated and difficult subject. It was very difficult for us to search for notes one by one, without using the openCV function, but we were very happy when we finally got able to detect musical notes and symbols one by one by one by one.

We heard that there was a team last year that did a similar project as we did, but our goal was to release a program that was 10 times more advanced than what they did. The result turned out great. We can detect almost all kind of musical symbols including resting note, sharps, flats. And it is fancier since our program can even produce chords(accords). So we are very proud that we've done a good job on this project, and this project helped us to build up and advance our image processing knowledge.