EE/SE/CPRE 491 - Spring 2019

Student Suggested Project

Sheet Vision

Design Document

Team Number

sddec19-13

Faculty Advisor

Alexander Stoytchev

Team Members

Bryan Fung — Frontend/Backend, Meeting Facilitator
Garrett Greenfield — Front end, Team Scribe
Ricardo Faure — Frontend/Backend, Meeting Facilitator
Trevin Nance — Machine vision, Chief Engineer Power System
Walter Svenddal — Machine vision, Report Manager

Team Website

http://sddec19-13.sd.ece.iastate.edu/

Version: March 28/Version 1

Table of Contents

List of Abbreviations & Symbols

1.	AWS	Amazon Web Services
2.	API	Application programming interface
3.	MIDI	Musical Instrument Digital Interface

List of Definitions

1. Sheet Music - Music in its written or printed form. [1]
2. Musical Notes - A sign or character used to represent a tone, its position and form indicating the pitch and duration of the tone. [2]
3. Tabs - A form of written music, but instead of being represented in the traditional sense (what tone it makes), notes are represented by the specific position they are supposed to be played in.

1 Introduction

1.1 Acknowledgment

We would like to express our gratitude to our advisor Dr. Stoytchev for taking the time to help us map out our project, as well as providing technical assistance. We would also like to thank Dr. Daniels for providing us with course resources and guidelines to follow, to better ensure our project success.

1.2 Project Statement

Reading sheet music is no easy task. With the creation of alternate ways to learn how to play music, such as tabs and youtube tutorials, there has been a decline in the amount of people who can properly read sheet music. The problem with tabs and other kinds of methods of reading music is that they lack the complexity to be able to convey all of the specific nuances that a specific piece may have. The best option that captures all of the nuances that most musicians wish to convey when writing music, is sheet music. The problem with sheet music is that it can be very difficult at first, and since there is a decline in the amount of people that can read it, it can be difficult to find a proper way to read it.

Our solution for this is Sheet Vision, an application that can read and show a user how the sheet music is played, and how it is supposed to sound. This will lead to the user being able to draw parallels between what is on the sheet, and the music being played, supplementing the learning process of reading sheet music. Not only will our application play the music on the sheet, but it will also listen to the user playing it, and will give proper feedback to the user, to fix mistakes they may be making.

1.3 Operating Environment

Our product is expected to be used quiet indoor environment, with appropriate lighting. This is for our application to more accurately pick up sound from the user when they're playing along with our on-screen piano prompt, showing which keys should be played. The lighting is important to allow the camera to pick up an image clear enough for the computer vision algorithm to properly pick up the symbols needed to find which notes should be played, and when they should be played.

1.4 Intended Users

This product is intended for beginners and veteran music readers alike. This application should provide instructions simple and clear enough for even first-time musicians can keep up with using our product, yet powerful enough to ease some of the struggle of reading more complex songs for veteran musicians.

1.5 Assumptions and Limitations

Assumptions

- Our product will be able to read written sheet music documents.
- Algorithm will be run on the desktop app and/or mobile app, so no internet connection is needed.

Limitations

- Some features will be limited based on the users possession of sheet music and a musical instrument.
    - The user needs to provide their own sheet music to scan, since our application will not provide it for them.
    - The user will need an instrument to make use of the play-along feature.
- The quality of our output will rely on the quality of both a users camera and microphone, depending on which feature they are making use of.

1.6 Expected End Product and Deliverables

Sheet Vision Desktop and Mobile Application - Expected Delivery Date : December 2019

- User-friendly and responsive user interface.
- System to read in images of sheet music.

- Computer vision system used to decode sheet music into information useful for the application.
- System that uses information provided by the computer vision system to select what notes should be played and when.
- Audio processing system that can detect and decode audio into information useful for application.
- System that uses information provided by audio processing system to notify the user if they played the song correctly or not.

2 Specification and Analysis

2.1 Project goals, Deliverables

- ReactJS desktop application for windows.
- Application can read images from camera/file directory.
- Algorithm can recognize music notes in sheet music.
- Application can play the correct notes that come from the processed sheet music.
- Application can listen to user audio using the microphone.
- Algorithm can recognize what music notes are being played by the user based on audio input.
- Application can compare output from audio processing and image processing to determine if user is playing the correct notes.

2.2 Design Specifications

The mobile phone application will be a standard mobile application, the windows application will be .exe and the Mac application will be .app. The application will be able to accept both .png and .jpg images to analyze. Then the a front end application will be able to play MIDI files, and will be able to listen to the live audio feed from the microphone to detect errors in the music it hears.

2.3 Proposed Design/Method

The design calls for the front end code to be written in ReactJS, wrapping it in Electron and React Native to provide multi platform usability. The machine vision will written in Python using the OpenCV library along with Numpy on an Amazon Web Services (AWS) server to take the load off the users device, and to allow the machine vision code to be reused without having to be re written for each platform.

2.4 Design Analysis

Architecture:

The computer vision algorithms will mainly be run on a machine on an AWS web machine and be accessed with the use of API requests sent from the client application to this machine, which we will refer to as the server. The client application will be available in the form of both mobile and desktop application, both multiplatform, thanks to the frameworks that we are using to create them. A stack of ElectronJS and ReactJS are used on the desktop client to permit it to be available on MacOS, Windows and Linux, with little to no changes to our code, making it simple to maintain. Likewise, for the mobile client, we will be using React-Native, which allows our application to be available to both android and iOS devices. With so many clients, how is it that we are maintaining the main component of the application intact and consistent through all operating systems and devices?

The way we are maintaining this consistent is by only having this component stored in one place, in which any kind of client can access, therefore, we decided it would be best to keep the computer vision and processing all separate from the client application and have a clear API in which we simply send data to the AWS machine, the machine processes the data and simply returns a data structure with data we can use to play sounds and create UI updates that act on the data returned. The data will be consistent no matter what device it gets sent to, making it easy to make multiple versions of the same app, without risking functionality and avoiding data inconsistencies when processing data on different systems.

2.5 Process Details

When the user opens the application they will be greeted with the option to upload an image of sheet music or take a picture themselves. Once the image has been selected/taken, the user will then be able to upload the images. Once uploaded a POST request will be sent to the AWS machine containing the machine. The AWS machine will accept the POST which will contain the image. Upon receiving the image the main Python program will be ran. Inside of the main program the image will be groomed for the NoteFinder object, this grooming includes removing image corrections, such as removing noise or rotating the image and converting the video to binary. When it receives the black and white image the NoteFinder will find all of the regions of interest and send them to different methods to have the notes extracted from them and put into arrays. Then these arrays are combined in the NoteMapper, this output is sent to the MidiConverter. Once the MIDI converter has finished the MIDI file is returned to the front end. From here the MIDI file can be played while animating the notes which are being played on an animated piano. The user can also play the sheet music themselves while the application listens to their playing and alerts them when they make mistakes.

3 Testing and Implementation

3.1 Hardware/Software

AWS Machine:

- Software:
- OpenCV
- Python
- NumPY
- Apache

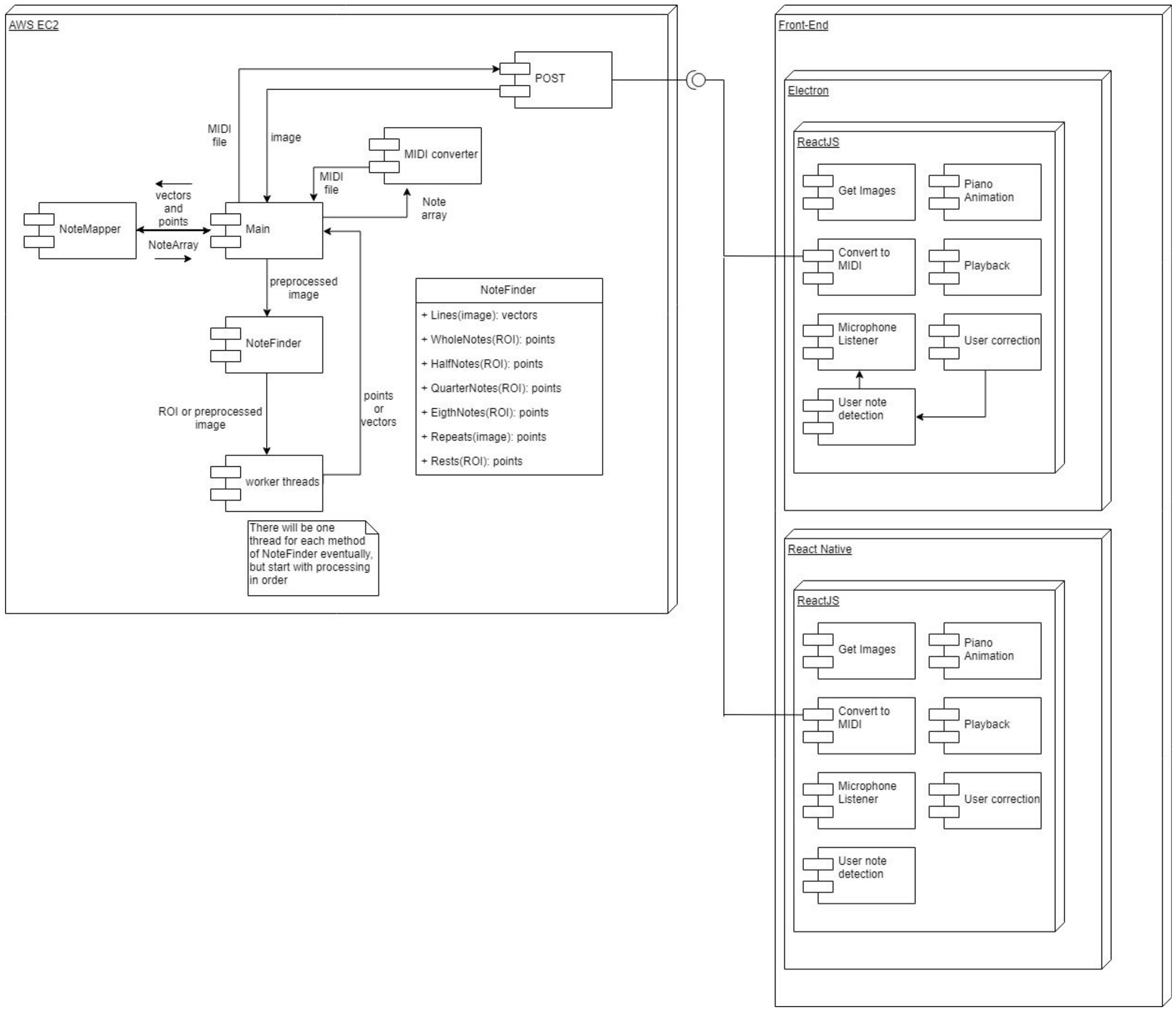
Client (Desktop)

- Hardware:
- High-Precision Microphone
- High-Resolution Camera/Document Scanner
- Software:
- ReactJS
- ElectronJS
- HTML, JS, CSS]

Client (Mobile)

- Hardware:
- Mobile Camera
- Mobile Microphone
- Software:
- React Native
- JS

3.2 Graphics



3.3 Functional Testing

The following list includes testing for functional requirements

I. The application shall be able to access the devices camera
II. The application shall be able to access the devices saved images
III. The application shall accurately display piano animations which correctly for a given MIDI file

IV. The application shall be able to accurately find and play simple music such as *Mary Had a Little Lamb*.
V. The application shall be able to listen to sound and notify the user if the notes are incorrect.

3.4 Non-functional Testing

The following list includes testing for non-functional requirements

I. Performance: Test that the machine vision algorithm should be able to analyze the music sheet within several seconds of the initial query.
II. Scalability: At least 50 devices should be able to make post requests simultaneously without affecting performance.
III. Extensibility: Test that the algorithm should be able to compile and run on AWS/APP.

3.5 Modeling and Simulation

Our application after creation is very easily mass produced to the public for the final step of revising our functionality through public testing. Because our product can be easily sent to the public we can have people who are excited about our product test the functionality and simulate any edge cases we may run into while creating new use cases for the product.

3.6 Implementation Issues and Challenges

The biggest challenge of the project will be the implementation of the computer vision to accurately read sheet music. There are many problems which can cause errors in the note recognition and detection. These problems range from issues with image quality to the specific stylings of the symbols on the page.

3.7 Technical Approach

Since our end goal is to have an application which can be used on a computer as well as a mobile device we are limiting ourselves to using a portable front end framework for our application. As well as implementing our machine vision in a language which will be easily interfaced with through our front end application, and is portable to different devices. These restrictions greatly changed our approach to the application we wanted to create by having to choose frameworks that can communicate and distribute the data throughout the project while still being multiplatform with an open option of adding apple accessibility in future updates.

3.8 Design Testing / Implementation

To test our sheet-reading algorithm, we have selected 5 different pieces of sheet music which must be translated correctly to a well-formatted MIDI file to determine success. We'll take pictures of the sheet music using our test phones, then supply them to the algorithm either through the application or directly. So long as the algorithm produces a correct MIDI file for the input of these pictures of sheet music, we will not need to do integration testing with the application to determine the correctness of the algorithm.

To test that our mobile application functions correctly, we will run through the process of receiving a picture by all accessible means and testing the communication to and from the backend with the MIDI files. The piano animation must also be tested, and for that we will create a MIDI file which contains a large range of different notes, time signatures, and so on to attempt to catch any errors with our piano. Finally, we will also test as many use cases for our app as possible, and ask acquaintance test subjects to attempt to use the app to try to find any errors.

3 Closing Material

4.1 Conclusion

Sheet Vision is an application for reading and playing sheet music, built on a stack of electron, react, react-native, openCV and the AWS cloud to allow support for multi platform compatibility, as well as keeping the application easy to maintain and quick to prototype in. It will follow a user friendly design scheme to allow all users, no matter their skill level in reading, to be able to take advantage of our application with no complicated learning curve, just drop the image and go. Our application is built to cater as large an audience as possible, going from any mobile or desktop operating system, and any skill level. There will be proper and thorough testing to make sure that our application is up to par with our, and our client's standards, and test cases will be used to verify code functionality as the application goes through its lifecycle.

4.2 References

OpenCV - https://opencv.org/

ReactJS - https://reactjs.org/

React Native - http://www.reactnative.com/

MIDI - https://www.midi.org/specifications

AWS EC2 - https://aws.amazon.com/ec2/getting-started/