Automated Instructional Aid for Reading Sheet Music

Albert Ng
Department of Electrical Engineering
Stanford University
Stanford, CA, USA

Asif Khan
Department of Electrical Engineering
Stanford University
Stanford, CA, USA

***Abstract*— The possibility of developing accessible optical music recognition presents an amazing array of opportunities to improve the musical education experience. While the topic has been the subject of research for decades, widespread application of this technology has not reached the general consumer. We have developed a sheet music image processing application targeted towards beginner music to provide an instructional aid for the novice musician. The application accepts an image file of the sheet music, annotates all of the notes, and generates and plays a MIDI file of the song. The application recognizes and classifies quarter, half, and whole notes with very high accuracy (97-100%) for digitized sheet music readily available on the internet. Other note types and symbols will be added in the future.**

Keywords-optical music recognition, sheet music, OMR, image processing

I. INTRODUCTION

Optical music recognition (OMR) has been the subject of research for decade. Many image processing algorithms and techniques have been developed to address this problem [1, 2, 3], and yet the problem still poses many challenges to scientists and researchers today [4, 5]. The traditional application of OMR has been the extraction of existing printed sheet music to digitized formats. The difficulty of the problem along with this limited application scope results in software that typically costs hundreds of dollars and requires proprietary file types [6, 7, 8]. An accessible and easy-to-use OMR application could provide an amazing tool for improving the musical education experience. For example, a novice musician could use such a tool to hear what a selected piece of music should sound like, giving the user a much better intuition during practice.

The goal of this project is to develop the algorithms necessary to parse digital sheet music images freely available on the internet, and to provide a playback mechanism for the parsed musical notes. The application is geared towards beginner music, the most common use case for such an educational tool. The application accepts an image file of the sheet music, annotates the note letters, octaves, and durations, and generates and plays a midi file of the song. The application is written in MATLAB with an added executable, and is currently capable of recognizing and classifying quarter, half, and whole notes with very high accuracy (see experimental results). We recognize that this is only a subset of the full feature set of musical notation, but we believe this is enough to extract significant educational value for the novice music enthusiast.

The processing algorithm can be divided into 5 steps: staff line detection and removal, segmentation, note head detection, note head classification, and midi synthesis.

II. ALGORITHM

The high-level overview of our image processing pipeline can be seen in Fig. 1. Here we explain each component in more detail.

A. *Staff Lines*

1) Detection: The first step in processing a given input image is to detect the individual staff lines of the piece of music. In our case, we used images which were already correct in skew and binarized, and thus were able to utilize the Y-projection of an image in order to determine where the staff

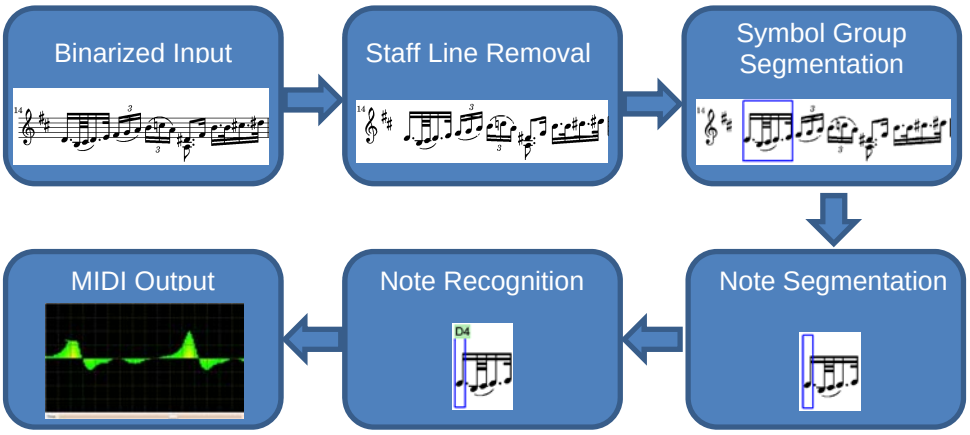

Figure 1. Sheet Music Reader Image Processing Pipeline

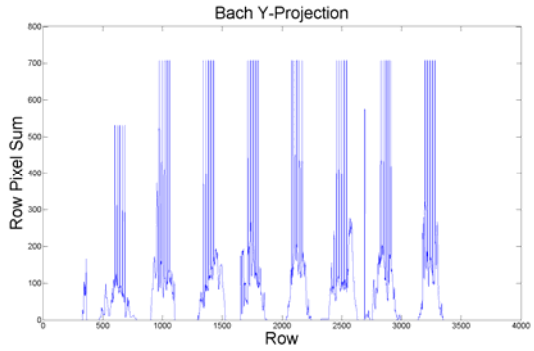

Figure 2. Bach Y-Projection for Staff Line Detection

lines lay. An example of this Y-projection is shown for a musical piece by Bach (see Appendix) in Fig. 2. In order to determine where the staff lines were given this projection, we made use of the following observations:

- Because staff lines are horizontal features in the image, they create large peaks in the Y-projection.
- These peaks are clustered in groups of 5 staff lines.
- Each cluster of staff lines has individual lines which are roughly spaced equally apart.

To this end, the algorithm we developed in order to extract staff lines initially selected local maxima from the Y-projection that were above a pixel count of *0.25*w*, where *w* was the width of the image. We further filtered this set of potential staff lines by noting that each staff line corresponded to very distinct peaks - to account for this, from this set of peaks, we only kept those which had a local minima following it that was at most 70% of the peak's value. This allowed for some noise suppression and in most cases ensured only one or a small number of peaks would remain for each staff line.

From this set of peaks, the next step was to make use of the clustering properties we knew to exist because of the nature of sheet music notation. To aid in this, we made a preliminary estimate of the pixel gap between staff lines, *g*, as being the median of the set of interval lengths between adjacent staff line candidates remaining from the previous steps. We then iterated through each of the remaining candidate peaks, forming clusters of 5. We excluded any peaks that were less than *g/2* or more than *2*g* pixels apart from the previous peak. Once we found a new peak that was more than *2*g* away from the previous peak, we assumed we had found a new cluster of staff lines. If the previous cluster was less than 5 staff lines, we would discard it as noise, and if the previous cluster was more than 5 staff lines, we would take the top 5 peaks and use those. In practice this worked quite well for detecting each cluster of 5 staff lines in the image. We then used the staff line locations to segment the image into individual staff clusters. An example of this for Bach is seen in Fig. 1.

2) Parameter Extraction: Once we had the final staff line locations from the previous step, we recalculated the gap between the staff lines, g, by once again computing the median of the set of interval lengths between adjacent staff lines. We also calculated the staff thickness, t, by compiling a list of all the black vertical run lengths in each column, and taking the median of this. We observed that the overwhelmingly most common run length in this case would correspond to the thickness of a staff line. At this point, we enlarge the image such that g is at least 20 pixels. This allows us to mitigate the effects of low pixel resolution in future processing steps. The parameters g and t are used extensively in future processing steps to set various image processing parameters during sheet music recognition.

3) Removal: We were then able to remove the staff lines by using the staff line locations and the parameter *t* found in the previous step to create a mask and use MATLAB's roifill() function to suppress staff lines. We did this to help make subsequent segmentation steps simpler. An example of this is seen in Fig. 1.

B. Segmentation

After the staff lines were identified and removed from the image, we segmented the image in multiple iterations in order to identify the individual notes. We performed 3 segmentations: staff segmentation, connected group segmentation, and note segmentation.

1) Staff Segmentation: In the staff segmentation, we divided the image into horizontal strips, one strip per staff. This was done from the staff line coordinates computed from the previous section. The center of the staff segment was the coordinate of the center of the 5 staff lines. The height of the staff segment was computed from the width of the 5 staff lines and the distance between the first two sets of staff lines. This was done so that notes that lie above and below the staff lines to be included in the staff segment.

2) Connected Group Segmentation: In the group segmentation, vertical strips of the staff segment were identified as group segments. This segmentation narrowed down our image search to only regions with interesting features. To tolerate some noise resulting from staff line removal, we first performed a sum filter with a neighborhood window of 1 row x 4 columns, and then took the X-projection of this sum filter as in Fig. 3. We thresholded the results of this X-projection and merged together small regions for further noise tolerance. The islands of this thresholded result gave us the group segments.

3) Note Segmentation: In the note segmentation, we identified vertical windows of the group segment that contain individual notes. To do this, we assumed that the note heads are taller than the other components of the notes, without including the note stems. Therefore, for each column, we

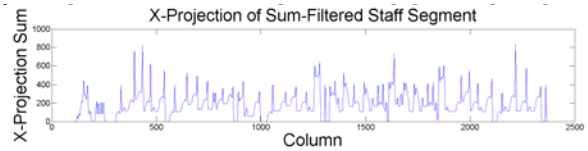

Figure 3. Staff Segment X-Projection for Group Segmentation

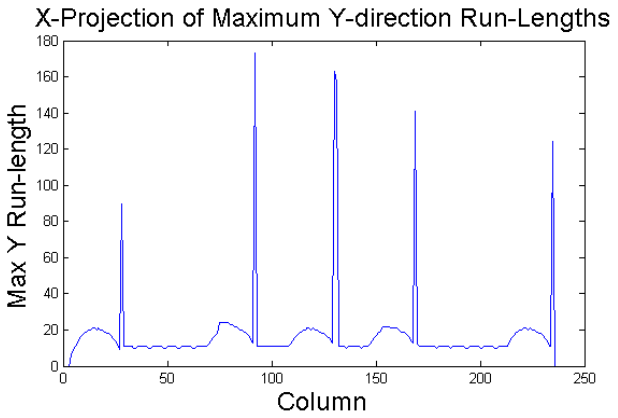

Figure 4. Group Segment Max Y Run-Lengths for Note Segmentation

several characteristics: dull mounds corresponding to note heads, sharp peaks corresponding to note stems, and an offset due to note beams. Because we expected note heads to be the height of a staff line gap, we removed max run-lengths greater than *1.5*g* and smaller than *0.75*g*. The islands of this result gave us the individual note segments.

C. Note Head Detection

Once the note segments had been identified, we were able to identify the coordinates of the note heads. We would like to perform a simple erosion to accomplish this task, but doing so would not detect any half or whole notes. We assumed that half and whole notes are individual notes that are not grouped together with any other features. Barring printing mistakes, this is typically the case, since half and whole notes are not connected to any other notes. Therefore, we filled holes with radius smaller than *g/2* for group segments of width less than 1.5 times a note width, which we defined to be *2*g*.

At this point, we could detect quarter, half, and whole note heads by simply performing erosion with a disk structuring element. Since we expected the note heads to have a diameter of approximately the staff gap line *g*, we chose a radius of *0.8*g/2* for the structuring element. An example erosion result is in Fig. 5.

D. Note Identification

Given the eroded regions from the previous portion of the processing pipeline, we next classified the note type, octave, and pitch of the note. To determine the octave and pitch, we used the centroid of the region found previously and cross-referenced it with the staff line locations of the original image in order to round the position to the nearest half step of g, which directly was converted into an octave and pitch. Our current implementation assumed the notes were written in treble clef.

We currently support only quarter, half, and whole notes. To determine the note type, we first determined whether the region around the note was filled or empty. To do this, we counted the number of filled pixels in a small circular region surrounding the centroid in the original image before erosion or small-region filling and thresholded to determine if a note head was full or empty. If the note head was full, we classify the note as a quarter note. If the note head was empty, we looked at

Figure 5. Note Erosion Results (Black) Superimposed on Note Segment (Gray)

the X-projection of the note to look for the presence of a note stem, which we observed to be at least *2.5*g* in length in most musical fonts. If we found a peak of this height or greater in the X-projection we assumed the presence of a stem, and classified the note as a half note. Otherwise we classified the note as a whole note.

Our segmentation approach also allowed us to naturally recognize chords if there existed more than one note in a given note column segment.

E. MIDI Synthesis

Finally, once given the list of notes, their durations, and pitches, we were able to generate a MIDI file for the decoded sheet music. We wrote a C++ program to generate MIDI files based on a command-line specification of musical notes and rests. This was essentially a string parser to handle a wide variety of input melodies, which then would strictly follow the MIDI file format standard to create a playable audio file. As part of our implementation of this application, we also allowed the user to select an instrument and tempo for the generated MIDI file. We called this executable from within MATLAB in order to generate and play the processed sheet music.

III. Experimental Results

We performed this processing algorithm on three sample sheet music images with different resolutions (see Appendix): *Jingle Bells* (471x580), *Twinkle Twinkle Little Star* (1564x626), and *Bach* (2479x3508). The first two samples are meant to be representative of beginner music for the most common use case. The third sample is a stress test on a complicated piece of sheet music. The results of the note detection and classification are described in Table 1.

TABLE I. Note Detection Results

Sheet Music	False Positives Detected	True Positives Detected	Correctly Classified
Jingle Bells	0	49/49 (100%)	49/49 (100%)
Twinkle Twinkle Little Star	3	42/42 (100%)	34/42 (81%)
Bach	9	323/334 (97%)	323/323 (100%)

IV. Discussion

Each of the results helps to illustrate a different aspect of the performance of our sheet reading application. For Jingle Bells, we see that we were able to detect the presence of each note correctly without false positives, as well as classify each

note's pitch and type correctly. This is one of many in a class of simple sheet music for which our application is able to generate playable MIDI files. These files help illustrate the robustness of our algorithms in the reduced feature set of musical notation that we currently support.

In the Bach piece, we were able to see how some of our algorithms performed on more complicated sheet music that we are not yet able to fully process. Here we see that we still detect the presence of almost all the notes, and of the notes we do detect we classify their pitch correctly each time. In this case, since we did not support every note type there, the note duration was usually mis-classified as quarter notes, but that was expected. Thus, in this case, we generate the right chords and notes, but the tempo of the synthesized MIDI melody was incorrect. These results are extremely encouraging for when we expand the feature set of our current application, as its performance on a relatively complicated piece such as Bach indicates that the algorithms will be quite robust moving forward.

Finally, we illustrate some of the challenges of sheet music recognition even in seemingly simple pieces such as the *Twinkle Twinkle Little Star* piece (see Appendix). Though we detect and classify the note duration correctly for every note, we get the pitch incorrect in 19% of the cases. This is because the musical font used exhibits some drift in the vertical placement of individual notes. This is most readily seen in the last half note of the piece. Even though the note is supposed to be a "C", when looked at individually and without any cognitive biases from the surrounding local context, it can easily be interpreted as a "B" as well. Results such as these suggest the importance of using intelligent context-aware methods to control against variable musical font renderings, and help to highlight the challenges required in creating robust detection algorithms, even in surprisingly simple-seeming musical pieces. This image, like all of the test images used in this project, was found through a Google image search and thus represent true modern renderings of musical symbols.

V. Conclusions and Future Work

In conclusion, over the limited timeframe given to conduct this project, we were able to successfully develop an application to read sheet music comprised of quarter notes, half notes, and whole notes, and to correctly synthesize a playable MIDI file from such an image. We evaluated our algorithm on several test pieces, both within and outside of our current processing capability, and were able to obtain extremely encouraging and motivational results. We hope to extend this work further to include more types of musical notes, rests, and other musical symbols. Finally, we hope to eventually improve on our algorithms further so that we can ultimately perform all of this processing on sheet music images taken with mobile phone and tablet cameras, which would allow us to revolutionize the way people are able to learn and play music.

Acknowledgment

We would like to thank Dr. Bernd Girod, David Chen, Vijay Chandrasekhar, and Derek Pang for their help and suggestions throughout the course of this class. Their teaching and mentorship were invaluable for the successful completion of this project.

References

[1] P. Bellini, I. Bruno, and P. Nesi, "Optical Music Sheet Segmentation," Proceedings of the First International Conference on WEB Delivering of Music, 2001.

[2] C. Dalitz, M. Droettboom, B. Pranzas, and I. Fujinaga, "A Comparative Study of Staff Removal Algorithms," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 5, pp. 753-766, May 2008.

[3] W. Homenda, and M. Luckner, "Automatic Knowledge Acquisition: Recognizing Music Notation with Methods of Centroids and Classifications Trees," International Joint Conference on Neural Networks, pp. 3382-3388, July 2006.

[4] A. Rebelo, I. Fujinaga, F. Paszkiewicz, A. R. S. Marcal, C. Guedes, and J. Cardoso, "Optical music recognition: state-of-the-art and open issues," Int J Multimed Info Retr, March 2012.

[5] D. Bainbridge, and T. Bell, "The Challenge of Optical Music Recognition," Computers and the Humanities, vol. 35, pp. 95-121, 2001.

[6] Capella-scan, http://www.capella-software.com/capella-scan.cfm, June 2012.

[7] Photoscore, http://www.neuratron.com/photoscore.htm, June 2012.

[8] SharpEye Music Scanning, http://www.visiv.co.uk/, June 2012.

APPENDIX

Here we show the three test sample images used for our experimental results.

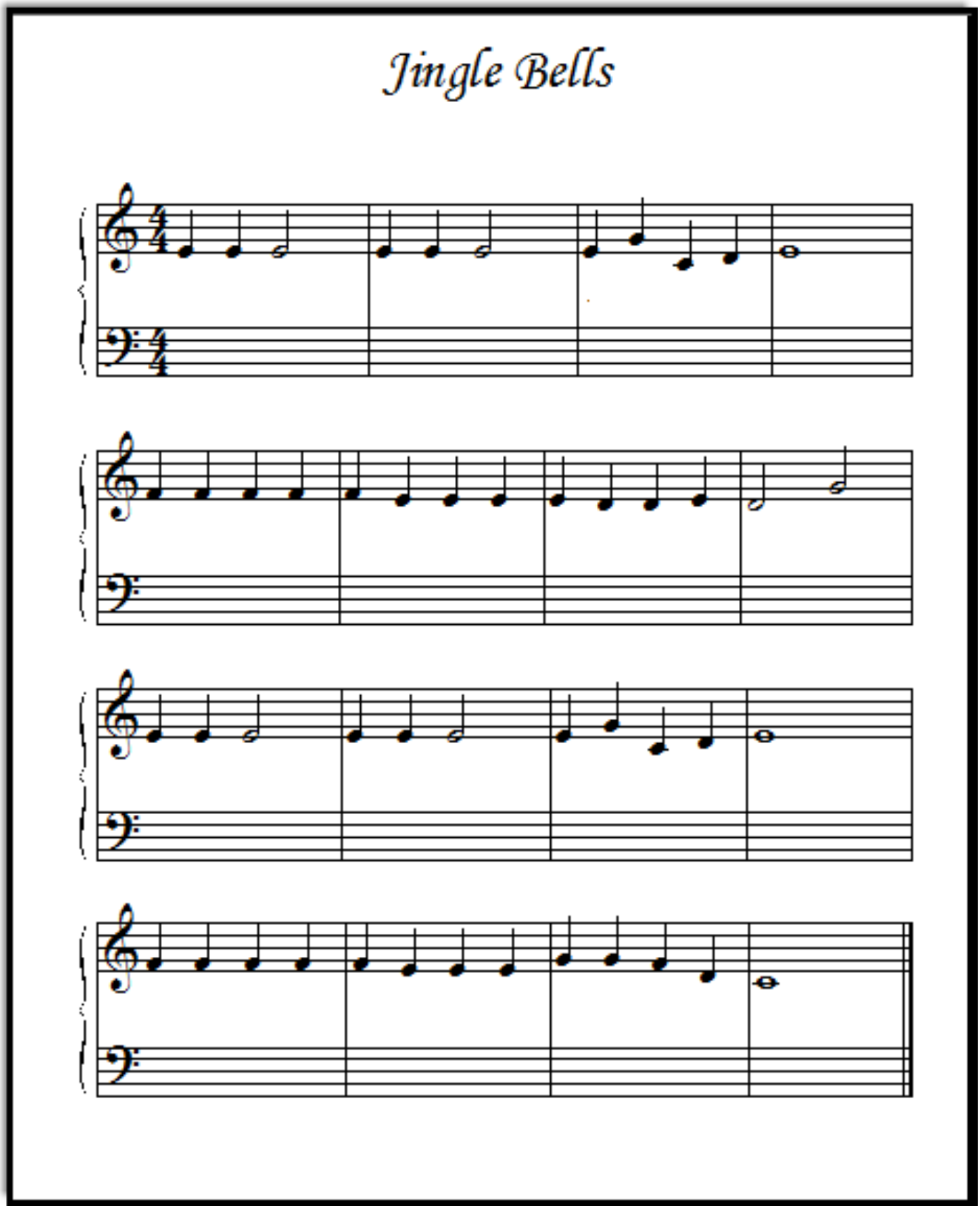

Figure 6. *Jingle Bells* Sheet Music Test Image

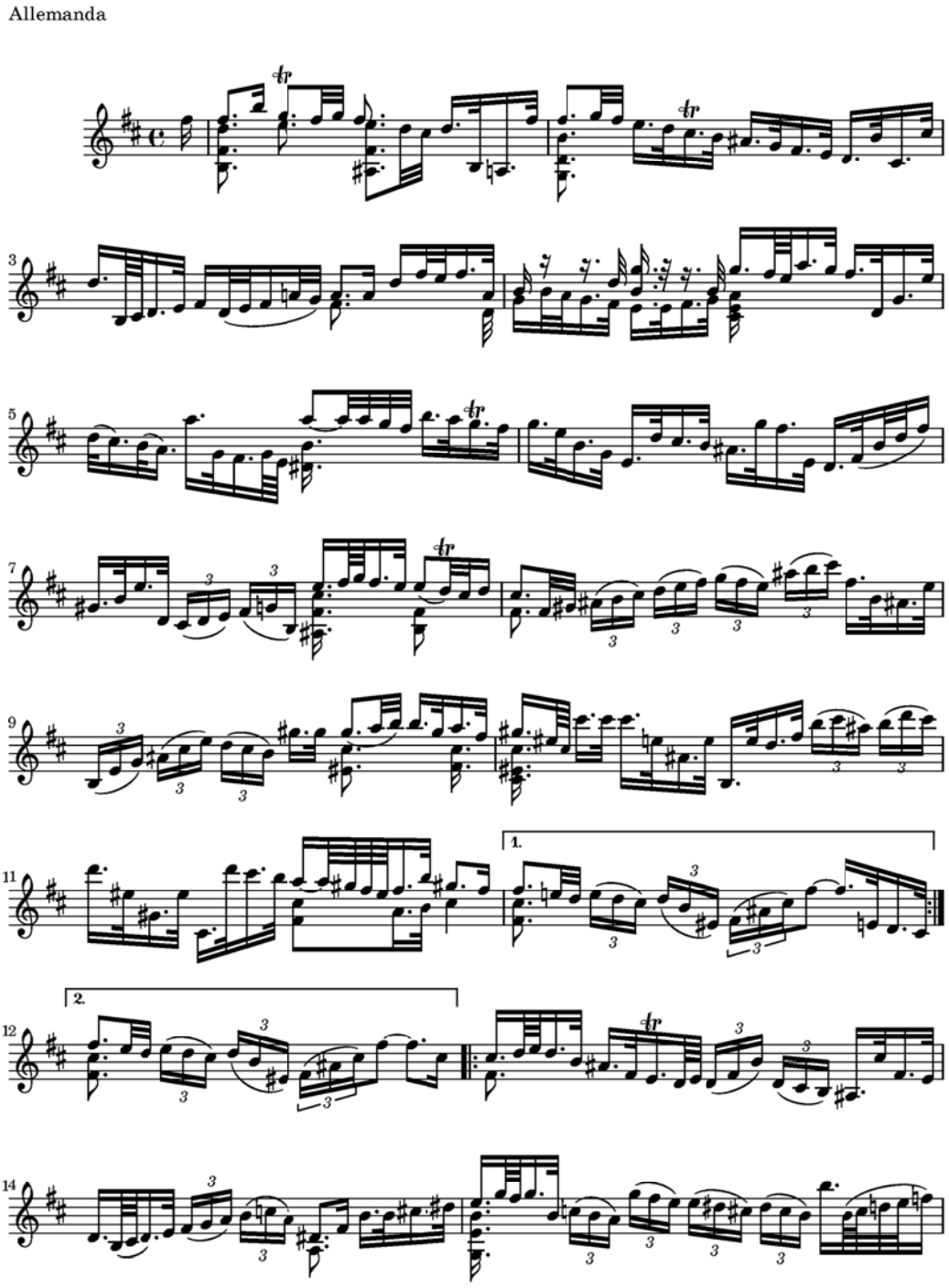

Figure 7. *Bach* Sheet Music Test Image

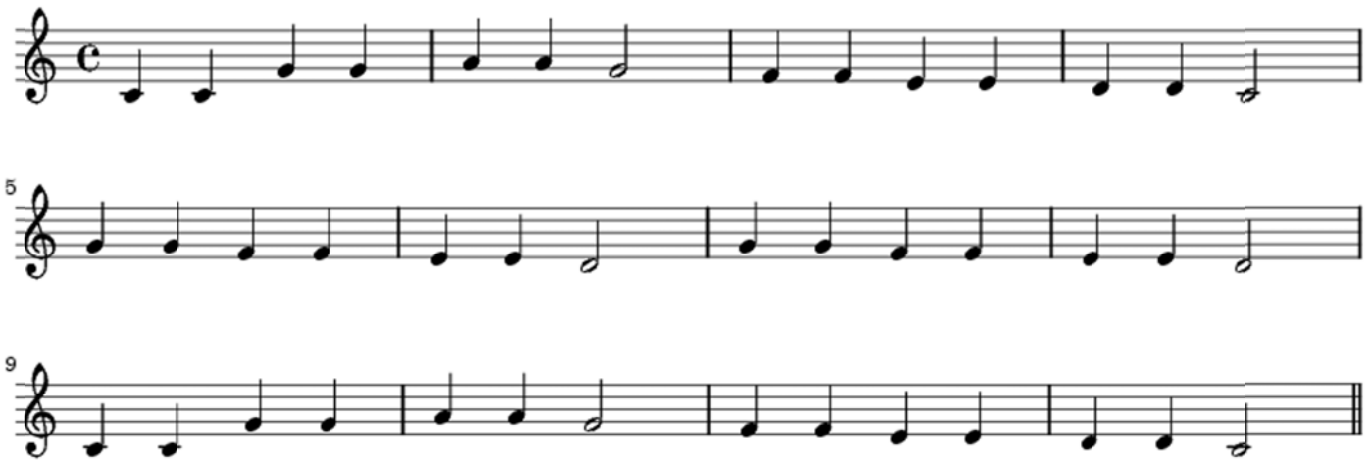

Figure 8. *Twinkle Twinkle Little Star* Sheet Music Test Image