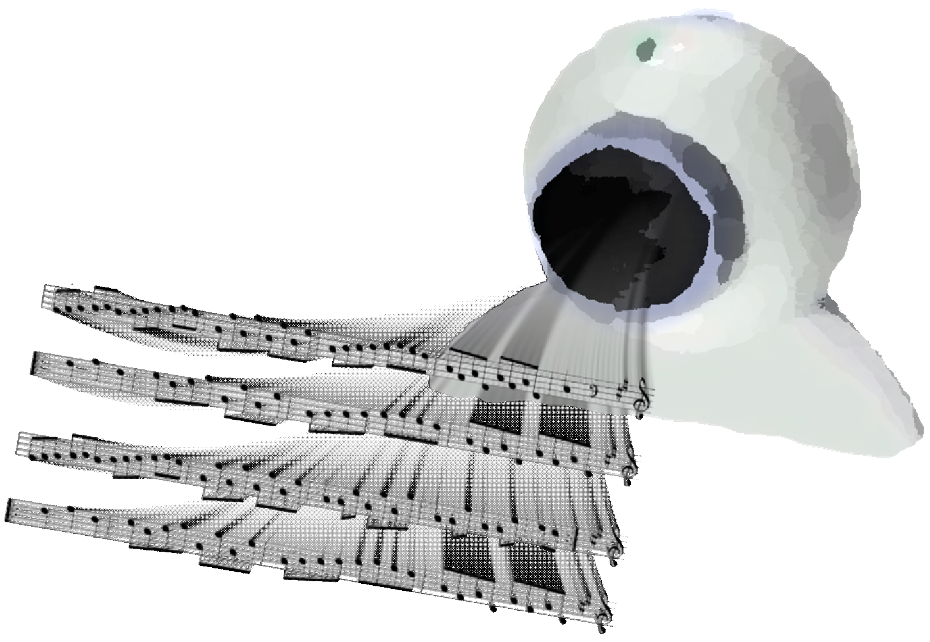
Camera Optical Music Recognition



Frank de Boer

Vrije Universiteit, Computer Science
June 2008

1 Introduction 3
2 Design 3
  2.1 Flex 3 SDK and ActionScript 3
  2.2 Problem 3
  2.3 Steps 4
3 Implementation 4
  3.1 Applying the filters 4
  3.2 Finding the staff 5
  3.3 Finding the notes 5
4 Music 6
  4.1 Storage 6
  4.2 Playback 6
5 Post Mortem 7

1 Introduction

Optical Music Recognition, referred to from now on as OMR, is the application of optical character recognition to interpret sheet music or printed scores into editable and, often, playable form. Once captured digitally, the music can be saved in commonly used file formats like MIDI and Music XML. OMR has various applications, such as scanning and digitally archiving large physical music libraries, scanning sheet music so musicians can play along to a digital playback of their score, or importing the score into their digital audio workstation.

This project focuses on fast computer vision OMR. Computer vision is the science and technology of machines that see. In this case, it means that a webcam was used for input. Using the Flex framework, the input is visualised and played back in a web browser. The benefits of low-latency recognition of sheet music are manifold: It can potentially allow computers to 'sight-read' music; it allows people unable to read music to quickly skim over melodies and harmonies or quickly enable composers to test the waters for their sketches.

2 Design

2.1 Flex 3 SDK and ActionScript

This project will use Adobe's Flex for easy input/output and its extensive imaging facilities. Flex is a highly productive, free open source framework for building and maintaining expressive web applications that deploy consistently on all major browsers, desktops, and operating systems.[1] The essential components of Flex are the ActionScript 3 programing language and the MXML markup language. ActionScript is widely known as the language that drives Adobe's Flash technology. Flex provides a complete enviroment for developing rich applications.

2.2 Problem

The classical problem in computer vision is determining whether or not the image data contains some specific object or feature. Humans can normally solve this task robustly and without effort, but is still not satisfactorily solved in computer vision for the general case: arbitrary objects in arbitrary situations[2]. The existing methods for dealing with this problem can at best solve it only for specific objects, such as simple geometric objects, human faces, printed or hand-written characters, or vehicles, and in specific situations, typically described in terms of well-defined illumination, background, and pose of the object relative to the camera. For the reasons above, the objects in this case are restricted to musically relevant characters on a sheet, directly in front of the camera, with an angle that is close to perpendicular to the axis of the camera.

For extra simplicity, rhythm and harmony are not taken into account when interpreting the music. Pitch and relative placement are the only parameters for a note.

2.3 Steps

Several steps are required to produce an interpretation. In chronological order, they are: image-acqusition, pre-processing, feature detection, extraction and finally, processing.

The first step is the image acquisition process. Image acquisition can be done in many ways, but for this project, several entry-level webcams were used. In the following pre-processing stage, the image is filtered and altered. Commen examples of filters are noise reduction and contrast enhancement. In this case, a noise filter was used to reduce the sensor noise caused by webcams. The image was also made ready for easier feature detection by applying a convolution filter. A convolution filter averages a grid of pixels into the center pixel, which can be used to create an edge detection effect. After pre-processing, feature detection comes into play. This step extracts features of various levels of complexity from the image. In the scope of this project, this step will detect horizontal lines to determine if there is a musical staff in view. This staff is then isolated in the extraction step. The staff is extracted from the image and prepared for final processing. In the last stage, the musical information is acquired through a reiteration of the feature detection process with a pixel density algorithm.

3 Implementation

3.1 Applying the filters

The first step, image acquisition, is trivial with Flex. Asking the Flex framework for a Camera object and accessing its data is a very simple procedure. With the acquired image stored into a Bitmap object, the pre-processing stage can commence. In this stage, three filters are applied to the image.

First, a noise filter is automatically applied to the image, either by the Flex framework or the camera drivers; this removes most of the sensory noise produced by the hardware. Second, a convolution filter is applied to the image. By applying a negative weight on one edge of the convolution filter, and a positive weight on the other, the values will trend towards zero if they are alike, and upwards if a contrast exists. This creates a bitmap with high values for strong edges, and low values for areas with little or no contrast, effectively providing an edge detection effect. In **figure 1** below, you can see the results of the convolution filter. In the top left of the image, a pen flashlight was used to illuminate part of the sheet for higher contrast.

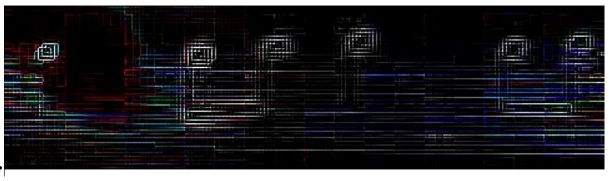


Figure 1 - Edge detection with a convolution filter

As a third step, the image is converted to a binary bitmap with a threshold filter. This filter calculates if the values for each pixel in the bitmap exceed a certain level, after which they are considered ‘on’. If they do not exceed the level, they are considered ‘off’. This binary bitmap is then used for feature detection. The threshold is a user parameter, because the level required for a pixel to be ‘on’ or ‘off’ based on values after edge detection depends on lighting conditions, camera placement and image contrast. Detecting and automating these paramaters would exponentially increase the complexity of the project.

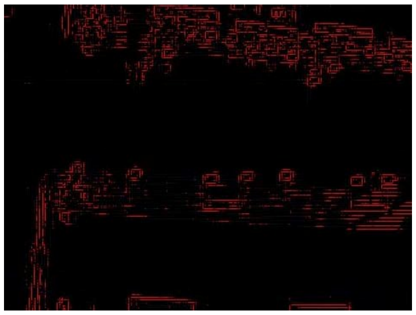


Figure 2 - A binary image (red is 'on')

3.2 Finding the staff

The image from the webcam is sampled into a variable matrix of Flex Sprite objects. For a 640x480 image, the default image sample size is a 64 by 48 matrix. The exact sampling size can be set, however, allowing the user to calibrate the system. The bitmaps in the Sprite objects are converted to binary images, so a pixel is either on or off as described earlier. The program iterates through each row of the matrix, applying a line-search algorithm to each Sprite object therein. This line-search algorithm does a per-pixel comparison, counting the number of pixels that are 'on', returning a positive result if the minimum amount of consectuive pixels for a line has been detected. If, for a given row R in Sprite matrix M, the amount of positive results x from the line algorithm exceeds a set threshold y, the row is flagged as being a potential element of a staff.

If a row from the matrix is flagged as a potential staff element, all subsequent rows that are also flagged will be considered part of a single staff. The number of subsequent rows deemed part of a staff will determine staff height. After this process, the staff -or staffs-, as a feature, has been detected and can be extracted for further interpretation

3.3 Finding the notes

A new bitmap is created by taking the detected staff and cropping the image to three staff heights, with the middle of the staff as the center. This bitmap is then sampled into another matrix of Sprite objects, with a sample size that is twice -to accomodate half steps- the amount of notes to fit in the vertical space. The amount of notes can be determined with a simple formula: A staff of five lines has 4 open spaces as seen in **figure 3**, so an open space is staff height divided by four. The lines themselves are conveniently ignored. The size of a single note 'dot' is generally an ellipse with the height of the open space as diameter. The minimum sample size to capture all the notes is

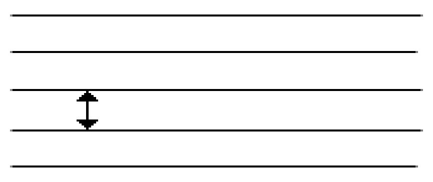


Figure 3 - The open space on a staff

therefore the height of the new bitmap divided by two times the size of an open space.

The resulting matrix of sprites is then run through another feature detection process. During this process, the sprites are sampled and run through an algorithm that detects rough circular patterns. Standard practices in pattern recognition weren't available in Flex, so a custom hybrid between curvature detection and blob detection (measuring pixel density) was designed and implemented. The curvature detection consists of drawing an imaginary circle over each sprite in the matrix. If the pixel density (the amount of pixels that are 'on' after all the filters have been applied) inside the circle exceed the equivalent number outside the circle, the sprite is flagged as a note, as seen in **figure 4** below.

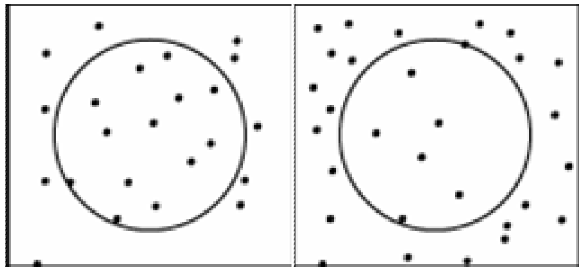


Figure 4- A note (left) and *not* a note (right)

4 Music

4.1 Storage

Once the music has been detected and extracted from the image, it is time to store the gathered data into a logical structure. Because this project ignores rhythm to keep it managable, common music storage methods like Music XML[3] are inapplicable. By giving each note a set length, one might be able to cheat the syntax, but doing so would defeat the purpose of a standard protocol. The data was therefore stored into a hierarchy of Flex Actionscript Classes, with semantics similar to Music XML.

4.2 Playback

Unfortunately, Flex, as of June 2008, does not have a MIDI library. MIDI, the defacto standard digital interface for music, would have been the ideal method of playing back the notes. Most MIDI libraries allow programmers to feed musical information into an interpreter, which uses virtual instruments or samples to play back the music.

The lack of MIDI gave rise to a new challenge. In Flex, the sound system is a black box that makes it very easy to play back files, but very hard to access the guts of the user's audio system. Regardless, two methods were devised to enable playback.

The first method of playback that was implemented for this project made use of an expirimental sound library that may or may not be featured in the next version of Flex. Using this library, a sine wave generator was built, to which the stored notes are passed as a value in Hertz. The resulting playback device can play back the frequencies of the notes in an obnoxious beep that is very unpleasant to the ears. Another problem with this method is that Flex timing is based on socalled ‘frames’ and that it is a relatively unreliable timing system. Sometimes, playback gets stuck, or a jittered beep is heard.

The second method makes better use of Flex’ feature set and is easier on the ears. I sat down at the piano and recorded three octaves of notes into a file format supported by Flex and wrote a class that routes the notes to their corresponding media file.

During playback, the lack of rhymic interpretation is circumvented by giving each note a length of roughly one second.

5 Post Mortem

The goal of this project, to bring computer vision to –simplified- music recognition, was partially reached. The simplification of interpreted music by omitting rhythm, key signatures, clefs and other elements not directly related to note pitch and placement were kept in place and none were added as extra features. None of this ‘could have’ functionality was added because of the complex nature of pattern recognition. While Flex was a great choice for this project in some ways, it hurt the accuracy and scope in others. Neural networks, an image library based on input rather than output and a MIDI library would have been valuable assets, but on the other hand, the built-in filters and the smooth, portable image capturing possibilities were really helpful.

The end result, as of now, is an application that is more of a demonstration of possibilities than a useful tool. The interpreted music is not always accurate and quite dependent on light conditions and camera quality. In some tests, even the reflective nature of the paper used had a significant influence on the interpreted results.

Despite the shortcomings, notes have been transferred succesfully from a piece paper held in front of a camera and played back to the user, which can safely be considered a success.

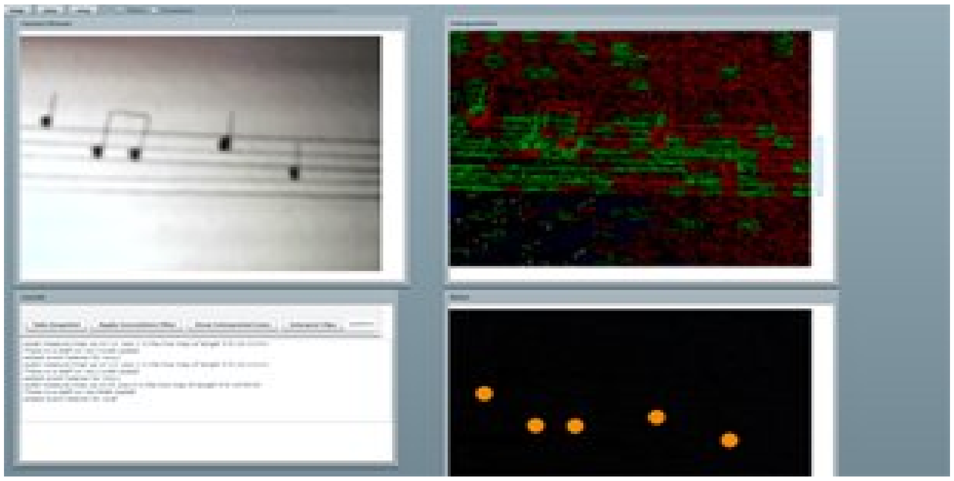


Figure 5 - The interface, clockwise from top left: The camera feed, the staff selection, the interpreted notes and the console.

[1] http://www.adobe.com/products/flex/

[2] R. Fisher, K Dawson-Howe, A. Fitzgibbon, C. Robertson, E. Trucco (2005). Dictionary of Computer Vision and Image Processing. John Wiley. ISBN 0-470-01526-8.

[3] http://www.musicxml.org/xml.html