Little changes, big effects - optimizing Korect

Ioan-Alexandru Eftimie
Computer Science Department,
„Politehnica" University of Bucharest,
Bucharest, Romania
ioan.eftimie@cti.pub.ro

Abstract – **Student examination is an important part of the educational process. Although various electronic testing and grading solutions have been developed and gained interest in the last years, classic short answer paper examination is still a preferred method of examination, due to the simplicity of implementation and low technical requirements.**

Korect is a complete solution for automatic test generation, processing and grading, with inline answer marks and paper processing via auto feeder scanners and OCR detection. This document presents work for optimizing the preprocessing phase and folder pooling during automatic correction flow. Results of this work are better stability and performance of the Korect solution.

I. INTRODUCTION

The evaluation of students using multiple choice test papers is a very popular method of evaluation in the academic environment. While this method has some obvious advantages, correcting tests is a repetitive activity, taking a considerable amount of time for a large number of students.

Korect is designed to resolve most of the problems associated with these types of test papers. It automates the grading process, necessitating as little human intervention as possible. Korect handles all the phases of the process, starting with question management, test generation, test grading and ending with reports, statistics and archiving.

Although for simple use cases (an exam up to 200 students evaluated using a single scanner and computer), the current solution performs well, the solution doesn't scale to larger exam scenarios, such as multiple scanners and a distributed processing environment. The initial attempt was to port Korect to a Hadoop managed infrastructure, but due to the complexity of the task and after close investigation, it was dropped for simpler, yet significant improvements.

This paper describes work done improving the preprocessing module – initially an external tool, called from within high level main code; a new module that pools the scanner's output folder for files is also presented.

The rest of the paper is structured as follows: Section II presents the related work, Section III contains a detailed description of Korect, Section IV describes the implementation of proposed optimizations, Section V presents the testing results, in Section VI we discuss the potential problems and possible solutions to them, and also presents the conclusions of the paper.

II. RELATED WORK

N. Lozano et. al developed a Scoring Tool for Electronic Paper Exams [1]. Digital ink is used by the students and text is captured and transmitted to the Paper Architecture server, where is segmented and analyzed. They take a hybrid approach, as automatic scoring is used for selection questions and free text answers can be corrected by teachers interactively through an interface. The interface is able to reorganize questions and mark empty questions in order to optimize marking time. The application also supports statistics export and automatic result e-mailing to the students.

N. Nakagawa et. al. extended the scoring tool in order to run the same application with different pen and paper devices, without having to re-write the main application [2]. A client has to be developed for each pen and paper device they want to support. The ink is first obtained from the device in a proprietary format, it is segmented into pages, and then the pen-tip coordinates are mapped to a standard coordinate system and stored in an InkML format. This file is sent by the client to the server that is able to analyze it.

G. Cen et. al. developed an auto-generated paper management system based on lightweight J2EE tools [3]. It includes a set of modules for user, subject, classification, questions and paper management. An efficient algorithm is used by the design process to perform analysis and compose examination papers. The paper will be generated automatically based on the subject, question type and difficulty level.

L. Shushu et. al. designed a mathematical model for auto-generating examination papers based on a genetic algorithm [4]. The model uses the intelligent search of genetic arithmetic in order to satisfy a set of requirements. The theoretical analysis shows that the genetic algorithm has polynomial time and space complexity and is able to efficiently generate exam papers in an intelligent manner.

III. KORECT

The evaluation process can be divided into different stages, each one with their own requirements and needed features.

The first stage consists of question management. With normal, manual methods of evaluation question

management is practically inexistent as questions are usually stored in an unordered manner. Korect permits the tracking of individual questions, as well as attaching different quantifiers such as difficulty and chapter.

The next stage, the test paper generation, can be done with greater control thanks to the advanced question management. Test papers can be generated using large pools of questions, using different parameters (what chapters to use for example) and both the questions and the answers are randomly arranged to prevent student from copying the answers from each other.

The third stage is represented by test evaluation. One of the advantages of Korect is that it does not need any special hardware. The tests are scanned with a scanner at a normal resolution (300 DPI or more) and then they are processed by the application. After the tests are evaluated by the application, the teacher can overview the corrected tests and make minor corrections (for example answers changed by the student). This is also the stage in which the student's name is filled in.

A. Architecture

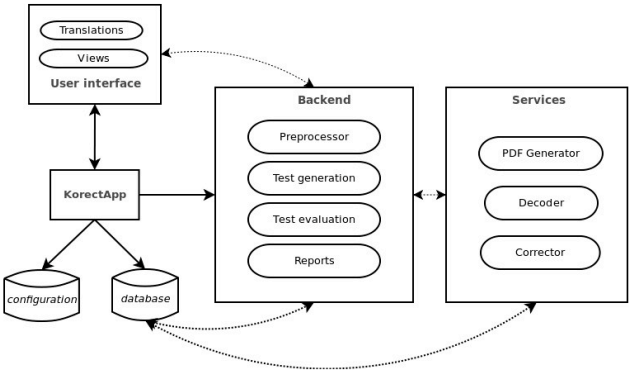

Figure 1: Application Architecture

Korect consists of a backend module containing specific modules for each stage, various services, independent user interface code, and the main application tying everything together, and also managing the configuration and database.

B. Functionality

The application's functionality can be divided into the following modules:

- Question management. This component is actually divided in 2 subcomponents, a question importer that reads a text file written using a simple format and the question management interface. Using these interfaces questions can be added, edited.
- Test generation. The test generation component receives any number of questions and answers and is able to create a PDF file containing the necessary markers, the bar code needed for the identification of the test paper and the respective questions. This PDF will be then printed, filled out by the test takers and then scanned and evaluated. This component also saves the geometry of the page, which is needed later in the data extraction process.

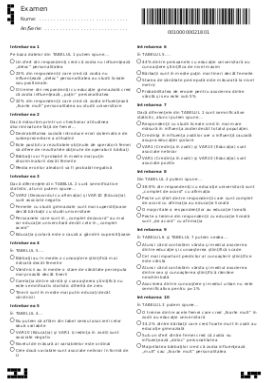

Figure 2: Generated test paper

- Test paper preprocessing. As the test papers are scanned the image is not always straight so test papers need to be prepared for data extraction. This is done by the preprocessing component, a program written in C using an optimized image processing library that takes the input image, detects special markers (Fig. 3) that are present on the paper, rotates it so that it is straight and removes the margins.

Figure 3: Markers used for alignment detection

- Test evaluation. The program detects the answers from each scanned paper and computes the grade of each student. It uses the geometry information that was stored in the first phase of the process to detect if the question was answered and which answer was selected if it was. If more answers were selected than they should have been, then the program marks it for manual inspection in the next stage.
- Test paper examination. After reading the answers from the test paper Korect provides an easy to use interface to examine each corrected test paper for evaluation errors. The test paper is shown along with an overlay with additional information such as selected answer, whether the answer was correct or not and total grade. In this interface the person who is correcting the test papers can make adjustments to the test paper.

C. Implementation

The "Preprocessing" component is one of the most important elements of the checking process. Since all the checks that are done in the later stages of the application are based on the geometry of the page, the proper alignment of the page is crucial to the correct operation of the application. The "Preprocessing" component uses the special markers that are created by the PDF generator to align and crop the

scanned image. The output image will be later used to detect and extract the needed information from the corrected test papers.

The "Preprocessing" component is implemented as a standalone executable and is built using cross-platform code. It uses functions from the OpenCV library, a high performance open source imaging library that implements a number of useful algorithms.

To properly detect the markers the program looks at a region in each corner of the image, and identifies all the contours in that region. It then checks the size of that contour and if it is big enough it compares the contour with the markers using a function in the OpenCV library, `cvMatchShapes`, which uses Hu moments.

After detecting the contours the program computes the bounding boxes of each marker. The markers are shapes composed of straight lines that can be inscribed in a rectangle. This solution was chosen because in this way the markers are different enough to be properly detected by the contour matching algorithm. By being able to inscribe the marker in a rectangle using the library functions the application can compute the corners (limits) of the image. This further allows to easily compute the center of the image (will be used for rotation) and the difference in angle by using the leftmost contours. The application then rotates the image and repeats the contour detection, this time extracting the extremities of the contours to compute the crop zone.

IV. Optimizations

A. Preprocessing module

Although the existing solution confers a good performance (preprocessing phase takes 250 ms per image), due to the OpenCV usage and compiled bytecode speed, the solution, external tool called via `subprocess.Popen`, isn't flexible enough.

A better solution will use C code for processing, but wrap it up inside a Python module – so that high level language features, such as exceptions, are available.

Simplified Wrapper and Interface Generator, or SWIG[5] for short, is a tool that provides a way to interface C/C++ with a variety of high level programming languages (notably Python, R but not Matlab – which has its own way of linking to C). It generates wrapper methods that allow the two languages to talk to each other.

Two modules were created using SWIG: `preprocessing` and `imageprocessing`. While the first does what the name says, the second one implements checks detection (student's answers) – a feature formerly implemented in Python, with poor results. Initially checks were detected by calculating the coverage percent; scanned image was loaded into memory, converted to a bitmap, then black dots were counted. A percent of black covering is calculated, and if this surpasses a given limit (constant), the check is considered valid. The new implementation makes use of specific computer vision API, called from within the C code; image agnostic results (Boolean answer values) are passed to higher level Python code.

Due to the fact that SWIG makes use of Cmake, a new build systems has been developed – using both Cmake and make for seamless and crossplatform configuration and building.

B. Folder pooling

Another part of Korect worth improving is the correcting process. Current limitations include the need to keep the GUI window open, after the process has started, the impossibility to stop/pause/resume it, lack of sensitivity to file system changes.

A new system, GUI independent, which can watch the folder for new files, stateful and fail safe was needed. The approach we have taken in this work was to develop a service (running as a different process), which manages the correcting process. Synchronization and communication with the main thread is done manually, using methods like an `@on_idle` decorator – assuring exclusive access – on the methods.

inotify is an event-driven notifier, its notifications are exported from kernel space to user space through three system calls. *Pyinotify* relies on inotify for monitoring filesystems changes. It binds these system calls and provides an implementation on top of them offering a generic and abstract way to manipulate those functionalities. We used *pyinotify* to watch the user selected folder for new files.

V. Testing

The newly developed preprocessing module has been tested using a real exam of 200 students, each test spanning over 2 pages.

The results show both speed and stability improvements:

	Original	*New*
Average preprocessing speed	250 ms	220 ms
Average correcting speed	2 sec/test	0.9 sec/test
Failed tests (on a 200 set)	8	3

Table 1: Testing results.

Average correcting speed improvement relies on moving the black coverage to from Python to C.

Failed tests number decrease is justified by better error handling – less papers show up in the manual review dialog, being automatically handled (discarded or moved to queue).

VI. Conclusions

Shape recognition and computer vision represent interesting fields in computer science. Automated test generation, correction and grading, may offer the subject for complex applications, such as Korect; fault tolerant flows, accuracy and speed, responsive graphical user interfaces are

aspects raising challenges and permitting various optimizations.

This paper's work reflects how little changes confer significant performance improvements. Using the right tool in the right place – or the right programming paradigm in the right scenario, assures best results.

By improving two key parts of correcting process – the preprocessing phase and checks detection, a significant gain of performance has been obtained.

Future work may involve applying the same approach for freeform extraction (i.e. student's signature) and report generation.

REFERENCES

[1] N. Lozano, K. Hirosawa, and M. Nakagawa, "A Scoring Tool for Electronic Paper Exams," *Seventh IEEE International Conference on Advanced Learning Technologies (ICALT 2007)*, Jul. 2007, pp. 120-121.

[2] M. Nakagawa, N. Lozano, and H. Oda, "Paper Architecture and an Exam Scoring Application," *First International Workshop on Pen-Based Learning Technologies (PLT 2007)*, May. 2007, pp. 1-6.

[3] G. Cen, Y. Dong, W. Gao, L. Yu, S. See, Q. Wang, Y. Yang, and H. Jiang, "A implementation of an automatic examination paper generation system," *Mathematical and Computer Modelling*, vol. 51, Jun. 2010, pp. 1339-1342.

[4] L. Shushu and W. Fengying, "Strategy and Realization of Auto-generating Exam Paper Based on Genetic Algorithm," *2010 International Conference on Artificial Intelligence and Computational Intelligence*, Oct. 2010, pp. 478-482.

[5] P. Ramachandran. "Wrapping with SWIG and Boost.Python: a comparison" in *SciPy'03 - Python for Scientific Computing Workshop CalTech, Pasadena, CA*