Automatic exam correction

Remco Hendriks

June 11, 2012
Abstract

Educational institutions apply traditional examination methods, using pencil and paper. Correctors are to read through the answers manually, calculating a grade from the work, and optionally provide written feedback. Students are to receive only a grade from the corrector, usually they are urged to collect feedback individually from the corrector. This traditional approach of exam correcting, grading and feedback provision, has pitfalls in regard to correction efficiency and feedback distribution.

This paper discusses a single uniform software package which focuses on exam generation, (automatic) correction and feedback distribution on an online internet platform.

The document generation is provided by \LaTeX[1], extended with a stylefile which generates the document format needed with the required calibration points, needed for automatic computer vision analysis. The computer vision package OpenCV[2] offers methods to calibrate scanned exam documents, masking regions of interest such as barcode identifier, student number and answer areas. The internet web framework Python-Django[3] holds all functionality required to port the correction and feedback portions to an internet application.
CHAPTER 1

Introduction

The Universiteit van Amsterdam employs various methods for teachers and instructors to examine a student’s knowledge of a particular subject. Most popular is the written exam, where the instructor provides a question sheet and a separate answer sheet. The question sheet is generated in a manner the instructor suits them best, enumerating the individual open and multiple choice questions linearly or in sections. Other examinations are used as well, popular courses with high attendance often employ multiple-choice only exams, which use a standardized answer sheet that is able to be scanned, recognized and corrected by a software package. Alternative methods such as verbal or practical examinations are left out the scope of this paper.

Both the automated and manual examination methods have benefits and disadvantages.

Most importantly, the manual examination provides ability for the instructor to write feedback on the sheets of the student, where the automatic examination greatly reduces time required to calculate a grade. This paper discusses an (internet) software package which has the combined functionality of providing feedback to the student as well as automatically correct multiple choice questions.

Spearhead abilities of the package are to move the correction environment from pencil and paper to an online interface, displaying the answer regions of the scanned paper sheets; provide automatic recognition of ticked multiple choice question boxes, and thus automatically grade multiple choice questions; access for the student on the online platform so it may receive (general) personalized feedback.
The automatic exam correction project is inspired and based on several other works.

The most prominent related work is *Auto Multiple Choice (AMC)* [4], which functionality set overlaps with that of Automatic Exam Correction. Identical exam generation and configuration file generation is derived from the AMC papers. Notably the AMC package does not offer open question correction and feedback provisioning, and operates stand-alone as an installer desktop application.

Číp and Babinec [5] have made a multiple-choice concept where document calibration and identification is described in detail to recognize forms. Many elements from Automatic Exam Correction are derived from this paper relating the document generation concept.

Lira, Bronfman et al. wrote *Multitest* [6], a full digital environment dedicated to generate, correct and analyze (digital) multiple choice answer sheets. Although the materials and methods used are relatively outdated, it reveals insights on a functional model for a digital correction environment.

Olšák [7] made a TeX library to generate EAN-13 barcodes, which transforms an input string into a pure TeX barcode, which is used for digital detection and identification in the Automatic Exam Correction project.

The book *Learning OpenCV* [8] is a great reference with many examples written by Bradski and Kaehler, on which the computer vision part of the Automatic Exam Correction project is based.
CHAPTER 3

Problem setting

To present a detailed overview of the Automatic Exam Correction (AEC) project, this paper first discusses the elements required with their respective problem set. Elaboration on elements and problem solving is explained in further chapters.

The features of Automatic Exam Correction is explained best in a functional model.

Figure 3.1: Users functional model image

Figure 3.1 shows the end-user functional model for both instructors / correctors and students. The instructors / correctors are to perform diverse actions to make use of the whole package. It depends on using \LaTeX with a stylefile provided by the AEC package, a flatbed or sheet-feed scanner which is able to produce PDF files, and any computer with internet connection.

More interesting is the program loop, which uncovers more of the coding functionality required.

Figure 3.2 shows the three key (computer science) areas; \LaTeX document development, Computer vision and Web development.

\LaTeX has all the tools necessary for the instructor to generate an exam to his or her specification, where the stylefile provides the specific commands to make sure all required extra information is present on the document, such as calibration marks, barcode, student number field and answer fields. Moreover, the stylefile generates a descriptor file, apart from the PDF file, which contains all meta-information about the document, in a simplified manner, which is computer readable at the time when the PDF scan is inserted into the web application for processing.

The web application provides the shell in which the computer vision processing, correction and publication resides. Python-Django provides the web framework which has convenient tools to set up an application at the scale required. Furthermore, the computer vision elements are
handled by OpenCV, which has a Python package, presenting the possibility to create a tightly integrated application where web elements and document processing live closely together.

Extra (third party) packages are required to provide intermediate processing, such as conversion from PDF to PNG files, barcode scanning and student number scanning. Some of these are readily available for Python, others run on different platforms.

An optional element of the project is to integrate additional (student) information by processing a Blackboard Learn[9] course students file. At the Universiteit van Amsterdam, all courses are mediated by the Blackboard Learn package, where all information regarding students, courses and grades are maintained. Blackboard offers exporting functionality to share information with other platforms. AEC utilizes this ability to enhance the publication functionality, which uses student’s email addresses to mail grades and feedback.

The pilot application does not look into possibilities to export grades and feedback into other systems, such as Blackboard Learn.
CHAPTER 4

Project targets

For the Automatic Exam Correction project, the following targets are defined.

The main target is to create a functional pilot application which implements the full functional model, from document generation to result publication. This main target is divided into separate (sub)targets, each ordered by their respective area.

The \LaTeX\ portion requires generation of the stylefile, which provides all elements required in the project further on. The stylefile should be able to produce a document which can be duplicated by an arbitrary factor, and be able to individually distinguish each copy, and associated pages, via a barcode. The stylefile must also be able to spatially detect and define the answer areas, to produce coordinates which are to be stored into the generated descriptor file.

The computer vision part requires the incoming PDF files to be converted to PNG image files, which then are ready to be used further on. Each PNG file produced has to represent one page. Every converted page is individually calibrated, barcode scanned and if necessary, student number scanned. The calibration of the documents is important, as it needs to be rotated and scaled into the position required to precisely pinpoint regions of interest (barcode, student number, answer areas).

The barcode represents the document identification. It should contain exam ID, copy ID, and page ID, to successfully order and link pages. This relieves the instructor to tell the web application which pages belong together. A barcode is generated by the \LaTeX\ stylefile, which is an adequate manner to transfer digital information from scanned sheets into the application’s internal model. Several other digital code options are available for this task, such as a QR-code. A barcode was chosen in preference of a QR-code, because there is a readily available pure \TeX\ package to generate barcodes [7]. The QR-code package has proven not usable to implement in conjunction to the stylefile.

The student number detection is a more delicate target. The students are to fill in their unique student number into the field on the exam which has separators for each digit. OpenCV is able to detect the student number region (aided by the descriptor file, generated by the stylefile), and isolate each digit to be processed. The \textit{MNIST}[10] website has several papers which document the routines to accurately predict which handwritten digit maps to a ‘real’ digit, which is viable in the AEC project to successfully link a scanned sheet to a student.

The descriptor file generated by the stylefile during document creation, precisely dictates which questions and answers are on the PDF file, including question number, points, abbreviation and coordinates. After document calibration, the answer coordinates exactly pinpoint the regions of interest needed by the web application to display the answer regions to the right question. When a sheet is successfully identified by barcode and student number, it can be inserted
into the application internal model along the information regarding the questions and answers.

Multiple choice questions are allowed in the exams to insert extra variety. The stylefile provides simple commands to create multiple choice questions, along with the information required to enable automated correction by the application, such as points per answer. Internally, the application thus must be able to detect the checkboxes, and figure out which is ticked (marked), ‘corrected’ (filled in entirely), and left blank.

To provide a correction environment in the web application, all elements need to be stored in a convenient manner such that exams, students and questions can be easily traversed. Each individual exam copy should be able to be graded based on question points or grade, feedback should be able to be given to each answer. The model should be prepared for intuitive and convenient usage of the application, such as in-order question correction.

The feedback loop from the instructor to the student is most important, and must be implemented in a fashion that it is easily accessible (over the web), and has a clear interface. This can be done by using the student’s email addresses provided by the optional Blackboard export file.
5.1 \LaTeX and stylefiles

\LaTeX\cite{1} is one of the most widely used software packages by professionals and academia, to generate adequate documents including special styling functions such as mathematical expressions or program code. It is widely extensible due to its programmable nature, and many \LaTeX\ packages are produced (by third parties) to add more functionality. A \LaTeX\ document produces a PostScript file, or PDF file, which are the uniform standard for document sharing, where the document always appears identical to the original, independent of the device used to view it.

An interesting package in relation to the AEC project, is the exam package\cite{11}. According to the documentation itself, it

"attempts to make it easy for even a \LaTeX\ novice to prepare exams."

Additional \LaTeX\ commands are made available to the user, such as \texttt{\textbackslash question}, \texttt{\textbackslash points} and \texttt{\textbackslash answer}. It works very convenient as it automatically enumerates the questions, and provides all necessary options to style the document to the desire of the instructor.

Packages for \LaTeX\ can be created by writing a stylefile, which defines all new commands with their own properties. Apart from ‘standard’ text formatting such as emphasizing, centering and sections; it is possible to create dynamic commands which make use of document variables, such as \texttt{\textbackslash thepage}, which returns the current page where the command is issued.

New packages can extend existing (included) packages, such as the position handler commands, which are used to return the cartesian pixel locations of where the handler commands are issued. The AEC stylefile uses this commands to locate the key regions of interest on the document (barcode, student number, answer boxes).

When generating a PDF file, additional files are created by \LaTeX, such as compile files which aid the PDF generation process. It is possible to create more external files, via the file handler commands, which are useful to write arbitrary data linearly to an external file. In the AEC project, it is used to write the document identification, question metadata and pixel locations of various regions of interest to an external file.

5.2 Computer vision and OpenCV

Computer vision is the term used for the processing of visual media by computers, like images, video and more. OpenCV\cite{2} is a well-known open-source package which has many applications in the field of computer vision. The library offers many functions to perform largely basal
modifications, which can be chained to implement more advanced operations.

For a computer, images and video are merely a large collection of individual pixels which do not bear any more information to the picture than position, color and luminosity. To discern structure from any image, OpenCV offers functionality to observe individual pixels and obtain picture information by scanning pixel neighborhoods. For example, OpenCV can detect a circle when adjacent pixel formations follow a certain curve in which it describes a circle with a center point and radius.

The AEC project uses the OpenCV package to calibrate image scans of exam sheets, preparing regions of interest for additional preprocessing by other packages, such as the barcode and student number. Furthermore, the detection of ticked multiple choice boxes, is handled by OpenCV as well.

OpenCV is primarily available as a C(++) library, with additional shell wrapper interfaces making the API available in various other programming languages. The OpenCV interface used in the AEC project is Python.

5.3 Python and the Django framework

Python[12] is a computer programming language. It has a wide reputation for its ease of use, clarity and brevity coding style and offers packages for almost any computer science area.

Django[3] is a high-level internet web framework written in Python which focuses on rapid development and deployment of well-structured applications. It more or less follows the model-view-controller architectural pattern, which separates data structure, state management and generation, and presentation. The Django framework emphasizes the Don’t repeat yourself principle, which focuses on reusability of components to create well designed complex websites.

A Django web application can be deployed easily on all common server platform and packages such as Apache[13] and MySQL[14], and includes scalable systems such as Lighttpd[15], nginx[16], Google App Engine[17] and Heroku[18].

The AEC project employs Django as the shell for all online activity, including the OpenCV package.
CHAPTER 6

Implementation details

6.1 \LaTeX{} stylefile

To create an exam, the instructor uses his or her own \LaTeX{} environment, where the \texttt{uvaexam} package is to be included on top of the file.

An example \LaTeX{} exam document can be found in the test files directory.

The \texttt{uvaexam} package is devised to make compatible exams as easy as possible, with simple \LaTeX{} commands. By importing the package, the document automatically places the calibration marks in the corners, and via extra commands standard information such as exam name, code, teacher and date are added.

The main environment is opened with the \texttt{\begin{exam}} command, where introductory text can be placed. Questions are defined in the \texttt{\begin{questions}} environment. Two types of questions are allowed, open or multiple choice questions. \texttt{\q} starts a new question, which accepts two arguments, the first for points possible, the second for a brief summary of the question. Content after the \texttt{\q} command is interpreted as question text, displayed on the final document. The summary in the second argument is used for the web application interface, and is thus not printed on the final document.

More elaborated questions with sub-questions is also included in the package. The \texttt{\{subquestions\}} environment can be defined after a \texttt{\q} command, wherein subquestions \texttt{\sq} are defined. The subquestion command accepts the same arguments as the question command.

To create a boxed answer-area, the command \texttt{\makeemptybox} is made available with the required size argument, which defines the height. The \texttt{\makeemptybox} command is required after every open question or -subquestion command, the package is not able to generate coordinates otherwise.

Multiple-choice questions are made via the \texttt{\begin{multiplechoices}} environment. Each \texttt{\choice} command within generates a checkbox, points and summary arguments are passed via the command arguments. Note that the multiple-choice environment must be preceded by a (sub)question command, to display the question text and maximum points available. The points assigned to the \texttt{\choice} command are not shown on the document, they are used in the web application to determine points scored.

Standard \LaTeX{} commands, and additional packages on the client system, can be used normally in conjunction with the \texttt{uvaexam} package, as long as the document template generated by \texttt{uvaexam} is not tampered with.
The more exotic functionality of the stylefile is the generation of an external descriptor file. The commands `\newwrite`, `\openout`, `\closeout` and `\write` are responsible for writing data to a (new) file in the same directory where the PDF document is generated.

A barcode is inserted into the document via a plugin which contains all macro’s needed to iteratively generate an image by using only TeX commands [7]. For the end-user, only the string to be converted needs to be provided for generation.

To duplicate an exam, the command `\examCopies` copies the exam by the factor given in the argument, and adds it after the first exam, updating the barcode so that copy and page numbering remain in order.

At the time the PDF file is made, the stylefile loops over the commands to linearly generate the descriptor file in the format the web application accepts. The governing format of the descriptor file is JSON, which utilizes a simple, readable syntax, and is easily converted by Python to a dictionary-format.

### 6.2 Web application design

The web application is the centerpiece of the AEC project. It controls all aspects from uploading documents to distribution of grades and feedback. This paper will focus on the main principles of the web application, which should provide sufficient information to reproduce core elements.

#### 6.2.1 Model

To store all information in a database, a consistent model scheme is required. The AEC model system is designed in such fashion that as little as possible data is stored. The hierarchy is best shown in a graph.

Refer to the model code reference for the exact implementation of the pilot, Listing 9.2

![AEC model system diagram]

Figure 6.1: AEC model system

Figure 6.1 displays the connection between individual object classes. What can be noticed is that there are acyclic relations, and most relations point to either Question and Sheet, which
both do not share a relation with each other.

The information obtained from the generated descriptor file and scanned content is to be mapped onto the model. A linear description of the hierarchy is discussed below.

The main entity is the Exam class. It describes all information regarding one exam instance, such as the owner (system user), name, date, and coordinates to document elements, the barcode, calibration marks, student number, -name and -email.

A Sheet is one copy of an Exam. This holds information about the final grade and feedback for this copy, as well as the owner (Student). All page images of one copy are stored into SheetImage.

Questions of an Exam are only defined once, in a separate class. The Question class does not share a relation to Sheet, to prevent duplicates. The data stored into this entity are question number, part, subpart, points available, page number, description (text) and coordinates to the answer region.

The Answer class defines an answer to a question, but is also related to a Sheet. This dual relation exclusively describes the indirect relation between a Student, Sheet, Question and Answer. Via backwards relations the correct ownership of an Answer can be deducted, as both Sheet and Question share a common parent, Exam. This particular method of relation composition is to reduce the number of Answer objects in the database; as an Exam (always) has Questions, but do not need an Answer from a Sheet (thus a Student).

The answer class describes information about the score and feedback to a ‘physical’ answer.

To implement multiple-choice questions (and answers), an ‘extension’ to a question is made as a relation between Choice and Question. Choice does intentionally not inherit the Question class, as it describes a many-to-one relation, Question can have multiple Choices. A relation could be made from Answer to Choice, in case the question related indicates a multiple-choice question. The Answer coupled to Choice denotes the choice predicted by the system. Choice attributes are points and description (text).

Coordinates are of less importance to the model system, it describes pixel coordinates to other model classes.

6.2.2 Views

To perform actions on the model, views are built around the model classes. Most of the views are concentrated around the System user, as this is the instructor or corrector who logs in on the application and manages all exams.

Traversal of the views mostly follow the hierarchical model system, in an intuitive manner.

The Exam view is the view which is used as primary entry point, a system user gets an enumeration of their exam objects after login. It provides a function to add an exam, which inserts a multitude of new objects into the entire system.

Following one exam object results into the Sheet view. It enumerates all sheets, with information about the students, grade and status, which is derived from the descendant Student and Answer objects. For example, the Status can be calculated by checking if the number of Answer object connections equals the number of Questions connected to the parent Exam (thus Status is ‘Done’).

Sheet provides general functions to review all sheets and publication.
# Sheet object is supplied as sheet
for question in sheet.exam.question.all():
    print [question]

    # Retrieves all Answer objects which relates to this sheet and question:
    answer = [query] Answer.objects.filter(sheet=sheet, question=question)

    if not answer:
        [create new Answer object and show]
    else:
        print [answer]

Snippet 6.2: Enumeration of questions and answers by sheet, displaying internal relation between
question and answer.

# Question object is supplied as question
print [question]

# Backwards retrieval of Sheets
for sheet in question.exam.sheets.all():
    if sheet.answer:
        print [answer]
    else:
        [create new Answer object and show]

Snippet 6.3: Enumeration of students and answers by question.

From the Sheet view the individual Sheet view can be accessed. It displays information re-
garding that single sheet, with ability to only modify information and review that particular
sheet.

Introspection into the Question, Answer and Choice objects do not happen with their ‘own’
view. The Review functionalities at the Sheet and individual Sheet view provides access to modify
these objects. Reviewing sheets can be done in both ‘linear’ (per student) or ‘ordered’ (per question) method.
Snippet 6.2 displays (Pythonic) pseudocode how to retrieve question and answers together.

Snippet 6.3 displays the backwards retrieval of Answers, where Sheet dictates the availability
of an Answer (‘Ordered’ review).

Adding grades and feedback happens in de Review mode, where the associated forms are
located per question. Figure 9.1 is a screen-capture from the pilot web application.

The publication functionality is built into the Sheet view, where the system user has the
ability to assemble the grades and feedback. It is possible to directly list all students with their
grade, and flag all or individual Sheets to be published. Flagging for publication enables students
to access the web application through a separate login, where login credentials are supplied to
the students in advance by e-mail, which is available in the system.
At the time a system user inserts a new exam into the system, information incompleteness may occur. Data such as student e-mail addresses is to be provided separately because it is not digitally obtained from the exam sheets. An extra view is introduced at the Sheet view which gives the system user the ability to add extra information to Student objects. A visual complement option is available to display an image of the student name, number and e-mail address field from the scanned sheets, with according forms such that the instructor is able to manually insert missing information.

6.3 Document insertion, computer vision

The most delicate functionality in the web application is the insertion of new exam sheets. A number of operations are performed onto the scanned sheets, in order to extract relevant information to insert into the application model.

Refer to the program loop in Figure 3.2 for the steps described below.

6.3.1 Conversion

First, the PDF document is uploaded into the application memory. To prepare it for OpenCV, the individual pages from the document need to be saved separately in the PNG image format, which is readable by OpenCV.

An easy and versatile package to perform this conversion is convert[19], which can be called from unix command-line. The web application performs this command-line action by executing

convert -density 150x150 [source] [destination]

The density 150x150 argument denotes the destination resolution of the source image, in dots per inch. The destination files are named equally to the source, appended with a hyphen and number. The Python file system browsing functionality is then able to find all the output PNG files, by a filename search.

Increasing the scanning density resolution leads to larger output files, and lengthens the conversion duration. The Test setup and results section advises a resolution of 300x300 DPI, twice of that used in the internal conversion. The reason for this decision is to keep the image filesize small, which increases the web application speed as it needs to transfer (download) every image file to the client when reviewing. Moreover, the original document detail at 300x300 DPI is sufficiently interpolated when scaled down to 150x150 DPI, for usage of the digital barcode scanning.

6.3.2 Calibration

Second, the individual PNG files need to be calibrated, as they may not be in the desired orientation and size for the web application interface. Different scanning methods, such as automated sheet feeders and manual flatbed scanners may produce different results, requiring document calibration.

The \LaTeX stylefile produces two calibration marks on the output document, two solid circles in both the bottom-left and the top-right corner. OpenCV finds these circles, with center point pixel coordinates, by extracting all contours from the document, with FindContours. A (OpenCV) contour is a collection of pixels which are spatially connected, where the edges (black-white) are extracted. Practically every visible object in the image has a contour rendition, requiring filtering.

The OpenCV package includes several tools to extract properties from a contour, such as contour area, which returns the area size in pixels of the enclosed contour. For detection of the circles, the following properties are tested.
\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} =
\begin{bmatrix}
  \cos \theta & -\sin \theta & t_x \\
  \sin \theta & \cos \theta & t_y
\end{bmatrix}
\]

Matrix 6.4: Affine Transformation Matrix, with rotation and translation factors.

MinEnclosingCircle interprets a contour as a (minimum enclosing) circle, returning the centerpoint and radius. If it returns zero, no circle could be fitted, thus those are discarded. The circle area is calculated from the radius by \( \pi \cdot \text{radius}^2 \). Areas smaller than two pixels are discarded, removing a lot of contour noise.

ContourArea is used to compare the OpenCV calculated area by the MinEnclosingCircle calculated area, where non-circle contours are compared to (interpreted) circle contours. If the contour is indeed a circle, these should be close together. The ContourArea is divided by the calculated circle area from MinEnclosingCircle, discarding all results \( x < 0.5 \) or \( x > 1.5 \), with \( x = \frac{\text{ContourArea}}{\text{circle area}} \).

ArcLength gives the length of the edge of the contour. Calculating the circumference from the radius of MinEnclosingCircle \( 2 \cdot \pi \cdot \text{radius} \), these can be compared with a division as well. Because of the great inherent property of circle roundness, the ArcLength and calculated circumference are very accurate, keeping only contours accepting \( 0.98 < x < 1.02 \).

The calibration circles are left in the remainder of contours, passing all above filters. Still, other circle figures present in the document may have passed as well. To remove these, the equality of the properties of calibration marks is used as a filter. It is known that the calibration marks are structurally equal, with identical area and circumference. Therefore all remaining contours are compared to each other, ranking which two contours matches best on their properties. These ‘top’ two contours then are elected as the two true calibration marks.

Next, the calibration marks are used to transform the image into the desired size and orientation.

WarpAffine is available to map a source image onto a destination image by an Affine Transformation Matrix, as in Matrix 6.4

The rotation factor \( \theta \) is calculated by deriving the angle between the calibration points, with \( \arctan((a_y - b_y)/(a_x - b_x)) \), with \( a \) and \( b \) the two calibration points.

Shifting the document into a compact viewport is done by factor \( t \), which is obtained through the center pixel locations of the calibration coordinates, added by a static slack factor. The document needs to be translated correctly to make sure that the coordinates of document elements (from the descriptor file) are still valid.

When calibration is completed, the resulting image is saved onto the system.

6.3.3 Barcode scan

\[\text{BIN8B6-2012/05/01-5-2}\]

Figure 6.5: Barcode example scan

Thirdly, the barcode from the pages are scanned and digitally read. The Python package
ZBar\cite{20} includes functions to insert an image, from which a barcode is detected and read. ZBar automatically scans the image for available barcodes in upright position, no additional instructions are required for digital recognition. Each sheet image file is fed into the ZBar scanner, which returns the code embedded. The barcode is in CODE-128 format, which provides high density of ASCII encoding. An example barcode from a sheet is displayed in Figure 6.5

The detection and information of the barcode is critical, as it validates document type, exam id, copy and page number. All scanned pages are checked if they have the same exam id, and if they are complete, i.e. if page two of a copy is found, a page one must exist as well.

The calibration of the page images may produce ‘flipped’ images, which are unintentionally rotated 180 degrees. When a barcode is not found in the region of interest, the image will be flipped, and checked again. Ultimately the upload process is canceled when no barcode could be found, returning a message to the user to retry and remove the pages without barcode.

6.3.4 Student number scan

![Figure 6.6: Student number example scan](image)

Fourth, the student number is digitally read. This is a more complex operation as it involves isolation of digits, and predicting what it is.

The students are to fill in their student number into one field on the document, which has small visible digit separators, dividing the field into twelve equal ‘parts’. The system is already able to detect the student number region (via the document descriptor), thus it only has to split each digit box apart for prediction. An example image of the student number field is displayed in Figure 6.6

Prediction of hand-written digits is a complex computer science area, from which only the main components are described in this paper.

The MNIST\cite{10} website provides references to diverse papers which implements algorithms to accurately predict digits from images. The MNIST website also provides an extensive set of learning and training digits to use. For the AEC project, a simple Support Vector Machine (SVM) method is implemented for detection and prediction.

LibSVM\cite{21} is a Python package which allows the programming of a SVM, with scaling, training and prediction functions. The SVM needs an one-time scaling and training of a learning set to be able to predict digits from a test-set. For the AEC project, the SVM has been calibrated to perceive a 98% accuracy on the MNIST test set. The resulting kernel is already supplied with the AEC web application.

The individual digits are fed into the SVM predictor, results concatenated to return a student number.

6.3.5 Insertion into the model

When all information is gathered, it is inserted into the relational model. The barcode dictates most of the relations, as it has sorted out document sets and ownerships. Information of the descriptor file is inserted simultaneously, providing the online environment all data required for
the corrector to insert grades and feedback.

6.3.6 Multiple-choice correction

The AEC project application supports automatic correction of multiple choice questions. The document descriptor file provides the pixel locations of the checkboxes, which then can be processed. OpenCV is used to check whether or not a checkbox is ticked. CountNonZero calculates the number of pixels which have a non-zero brightness, even the slightest aberration is not included. Before this function is applied, the checkbox image is scaled up to 400%, thresholded such that all pixels are converted to either black or white, then smoothed with a 9x9 Gaussian kernel to enhance the presence of lines and scribbling found.

The CountNonZero figures are used to determine whether a checkbox is ticked, filled in entirely or left blank. Simple tests revealed blank boxes to have less than 2,000 nonzero pixels, corrected more than 6,000, and checked within those margins.

In the case more boxes are ticked (or corrected), an equilibrium algorithm is implemented to check which one appears to be a better prediction. The equilibrium in the pilot implementation is set in the middle of the ‘checked’ margin, 4,000 nonzero pixels. The checkbox who is nearest to this center, gets elected as ticked.

When no boxes are selected (thus all have less than 2,000 nonzero pixels), no answer is elected and thus no points awarded.

Internally, a multiple choice question can have multiple ‘answers’, it is possible for the instructor to assign any points to any answer choice.

Figure 9.1 displays a multiple-choice question, with images of the marked boxes. The prediction system has elected the third option as ‘checked’, above the first option who is clearly ‘corrected’.
Due to the size of the AEC project and time available, the application code is not written with testing principles like Test Driven Development. The development is focused on delivering functionality fast, with in-place tests and checks. Therefore it is not possible to run automated tests to verify all functionality is working properly. Nevertheless key elements of the project are described in this section with according results.

Enclosed with the AEC application is a directory with various files for testing purposes. The results in this section are derived from tests with these files.

The test environment is a Virtual Machine instance of Ubuntu 11.04 LTS running on a host machine equipped with a Core2Duo@2.0GHz and 4GB of RAM. The `convert` tool requires ImageMagick to be installed on the machine. The Python-Django distribution must be at least version 1.3 or higher. The tests and results are obtained via the built-in Django development web service.

Primary concern which needs adequate testing are the computer vision elements. Because the application depends on external factors such as scanning machines, some critical points are discussed below.

The calibration mechanism used depends on the assumption that the paper sheets are scanned onto a flatbed scanner or into a sheet-feeder. Therefore misplaced scans which might warp or obscure the paper could lead into failure of recognition. Overly bright or tampered calibration marks result into failure as well. Scanning the documents is advised at a resolution of 300x300 DPI, to preserve most of the document detail. Determining the size of the calibration marks is done by experimentation. Too small marks may cause less roundedness in conversion, too large marks are not visually pleasing on the documents. The system is particularly resilient to attempts to reproduce the marks (elsewhere) on the document, as it is nearly impossible to manually draw a perfect circle by hand.

The conversion from PDF to PNG images using `convert` requires the most processing power, because it has to pixel scan each PDF sheet at a resolution of 150x150 DPI. The test PDF, which includes ten pages, requires 14.4 seconds to convert to PNG files at a 1210x1728 px resolution. The performance of the other conversion and insertion elements are negligible. The upload of the files from client to server depends on the speed of the internet connection, this might add up when uploading PDF files with a large number of pages, up to tens of megabytes.

The test PDF has remarkable calibration accuracy, resulting PNG image files show rotation error up to one pixel vertically to on average a thousand pixels horizontally, posing no problems or errors of further processing, like barcode scanning which requires upright images. This also results into very precise extraction of regions of interest further on in the application. There are
Student number prediction has resulted into poor performance. Although the MNIST test set has proven 98% accuracy, the converted digit images do not approach such figures. Even when writing very clear digits, the SVM classifier does not return reasonable answers, such as predicting a hand-written one into a five. Therefore it is not possible to express accuracy in percentage, as it seems entirely random and thus a 10% prediction chance.

The multiple choice answer and prediction system is implemented based on manual testing. Several test documents were made with a large number of multiple choice questions, which are filled in randomly to generate a test-set. The decision margin is implemented based on the result of the nonzero pixel count tallies. The results of the nonzero pixel counts are enhanced by enlarging the checkbox image patch and additional smoothing, giving more easily recognizable transitions between the different states of checkboxes. Additional tweaking of the prediction mechanism may be needed when using the system in live production mode.
CHAPTER 8

Conclusion

A number of conclusions can be extracted from the (performance) results of the AEC pilot test. The major part of the application has adequate performance, scanning and conversion of documents are working properly as it displays correctly in the web application environment. The modification of data, such as grading, feedback and publication, behaves as expected and do not need further tweaking for their current function.

The poor performance of the student number prediction has led to the decision to not activate it for the pilot version. Because the SVM classifier always makes a prediction and does not return any warning about performance, it is unable for use. The source of the weak performance may be found in the incorrect conversion from individual digits to the format the test-set MNIST uses. Elements as noise, warped and translated digits may affect performance, although manually addressing these points with graphics editing software did not give any improvements. As conclusion, the SVM classifier needs to be revised, or replaced with other digit recognition packages such as Neural Nets or Convolutional Nets, which show similar or better test-set results on the MNIST website[10].

The program code of the SVM classifier is located in a separate directory in the AEC project package.

The prediction of multiple choice answers is reasonably good, unless carelessness scribbling is introduced. Because the prediction is done based on pixel count, it is not possible to use thick pencils to answer. Best performance is perceived with a standard 0.5mm pen. To ensure best prediction performance, students are advised to carefully and adequately fill the checkbox square when meant to be corrected.

When a student unintentionally corrects a checkbox instead of ticking, it is not possible to re-elect a faulty checkbox unless the student is able to erase the scribbling.
Recommendations for future work

The current implementation of the AEC project is only a pilot version. It offers the full functionality as described in the introduction, but there are elements which could be enhanced even more.

Additional flexibility could be introduced into the LaTeX document generation, the student name, number, email, barcode and calibration marks coordinates are currently hardcoded into the stylefile. This does not pose a problem in the pilot version, because no handlers are available to relocate these elements, but additional personalisation of documents could be welcomed which needs variable coordinate tracking. The web application is already fully compatible with variable coordinates for these elements.

During the development of the LaTeX stylefile, a QR-code was opted as digital identifier. Unfortunately, the QR-code plugin for \TeX{} did not work correctly when processing the \LaTeX{} document to PDF file and descriptor file, as it is not written in pure \TeX{} code. Therefore a barcode is introduced as replacement, but future change to QR code is recommended as it occupies less area on the document, could hold more information and has better validation. The digital scan of a QR-code could also prove to be more robust, as the individual pixel blocks are larger than single-pixel wide bars, and thus better recognizable.

As shown at the results and conclusion section, the student number detection needs improvement. The current SVM mechanism does not perform and needs revising until it reliably predicts student numbers. Various mechanisms are introduced at the MNIST website[10] with great performance ( > 99% accuracy).

The current pilot web application is not very resilient to faulty conversion and recognition of the documents. For example, when one sheet is found unrecognizable, it stops the entire uploading and conversion process, leaving the system user with the message that one of the sheets is faulty. Additional views could be introduced to the upload function, prompting the user to fix or skip any errors in-place, without discarding other (correct) information.

Converting PDF to PNG is the most resource intensive task when uploading new exams. Unless a moderately powerful server machine is dedicated for operating the service, this conversion could be nominated for improvement, as the other conversion operations are nearly instantly.

The current web application interfaces allows uploading a Blackboard student information file, which internally links Sheets to Students, which is viable when distributing grades and feedback to the students. A better solution would be to connect directly to a centralized database with student information, but this could cause security issues. A possible interesting option could be connecting to a federated network of identification domains, such as SURFnet[22] or Kennisnet[23], which discretely and securely allows students to log in to the AEC web application through a gateway operated by the federated network. This would cause to retrieve student
information after publication, because primarily only the student number is (digitally) available from the exam sheets. Furthermore, the scanned sheets would be anonymous unless the corrector peeks at the handwritten student name.
Bibliography


[5] Číp, Babinec, Optical processing of questionnaires, Doctoral Degree Programme, Faculty of Electrical Engineering and Communication BUT, Czech Republic, (Date of publication unknown).


[8] Bradski, Kaehler, Learning OpenCV, Computer Vision with the OpenCV Library, O'Reilly Media, September 2008


[19] ImageMagick, Software suite to create, edit, compose, or convert bitmap images, ImageMagick Studio LLC, Website: http://www.imagemagick.org/


[22] SurfNET, National research and educational network management of The Netherlands, Website: http://www.surfnet.nl

Appendix
Figure 9.1: Example view of the web application’s review functionality. Note the multiple choice question, which was automatically answered by the system, indicated by the green background of the correct answer.
class Student(Model):  # Student information
    username = CharField
    studentnumber = CharField
    email = CharField
    firstname = CharField
    lastname = CharField

class Coordinates(Model):  # Pixel coordinates of items on document
    bl_x = IntegerField
    bl_y = IntegerField
    tr_x = IntegerField
    tr_y = IntegerField

class Exam(Model):  # One exam set
    user = ForeignKey(User)
    name = CharField
    date = DateField
    barcodeCoordinates = ForeignKey(Coordinates)
    calibrationCoordinates = ForeignKey(Coordinates)
    studentnumberCoordinates = ForeignKey(Coordinates)
    nameCoordinates = ForeignKey(Coordinates)
    emailCoordinates = ForeignKey(Coordinates)

class Sheet(Model):  # One copy of an exam
    exam = ForeignKey(Exam)
    student = ForeignKey(Student)
    manualGrade = DecimalField
    feedback = TextField
    published = BooleanField

class SheetImage(Model):  # Image file of sheet, per page
    sheet = ForeignKey(Sheet)
    page = IntegerField
    file = CharField

class Question(Model):  # One question of an exam
    exam = ForeignKey(Exam)
    number = CharField
    part = CharField
    subpart = CharField
    points = IntegerField
    text = TextField
    pageNumber = IntegerField
    coordinates = ForeignKey(Coordinates)

class Choice(Model):  # One multiple choice answer of a question
    question = ForeignKey(Question)
    points = IntegerField
    text = TextField
    coordinates = ForeignKey(Coordinates)

class Answer(Model):  # Answer of a single question, from an user (sheet)
    question = ForeignKey(Question)
    sheet = ForeignKey(Sheet)
    score = IntegerField
    feedback = TextField
    choice = ForeignKey(Choice)  # MC only

Listing 9.2: (Pseudo)model of the web application.