Bachelor Arbeit Abt. I FS20

Projekt: Korrektur-Unterstützung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Digitale Korrekturunterstützung

Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2020

Autor(en):	Tiziano Gemma, Lukas Christel
Betreuer:	Prof. Stefan Richter
Experte:	Dr. Ferranti Ettore
Gegenleser:	Prof. Markus Stolze

1. Inhalt

1. INHALT 2

2. AUFGABENSTELLUNG 4

3. ABSTRACT 6

3.1 ZIEL DER ARBEIT 6
3.2 VORGEHEN / TECHNOLOGIEN 6

4. MANAGEMENT SUMMARY 7

4.1 AUSGANGSLAGE 7
4.2 VORGEHEN 7
4.3 TECHNOLOGIE 7
4.3.1 Handschrifterkennung 7
4.3.2 Bildmanipulation 7
4.4 ERGEBNISSE 8

5. TECHNISCHER BERICHT 9

5.1 AUSGANGSLAGE 9
5.2 BEGRIFFE 10
5.2.1 Offline Handschrifterkennung 10
5.2.2 Online Handschrifterkennung 10
5.2.3 Codes und Marker 10
5.3 PROBLEMBESCHREIBUNG 11
5.3.1 Use Cases 11
5.3.1.1 Übersicht 11
5.3.1.2 UC1: Druck-PDF generieren 12
5.3.1.3 UC2: Korrektursession CRUD 12
5.3.1.4 UC3: Erkannte Punkte Anzeigen 13
5.3.1.5 UC4: Live Addition 14
5.3.1.6 UC5: Notenschlüssel erstellen 14
5.3.1.7 UC6: Notenblatt exportieren 15
5.3.1.8 UC7: Zu korrigierende Seiten anzeigen 15
5.3.1.9 UC8: Multiple-Choice-Aufgaben auswerten 16
5.3.1.10 UC9: Antworten als Text speichern 16
5.3.2 Non-Functional Requirements 16
5.3.3 Problembereiche 17
5.3.3.1 Prüfung filmen 17
5.3.3.2 Perspektive korrigieren 17
5.3.3.3 Punkte auf der Prüfung suchen 18
5.3.3.4 Handschrifterkennung 18
5.3.3.5 Punkte zuordnen 18
5.3.3.6 Punkte manuell korrigieren 18
5.3.3.7 Persistenz 18
5.3.3.8 User-Interface 18
5.4 LÖSUNGSKONZEPT 19
5.4.1 Prüfung filmen 19
5.4.2 Perspektive korrigieren 19
5.4.3 Punkte auf der Prüfung suchen 19
5.4.4 Handschrifterkennung 20
5.4.4.1 Cloud-Dienst 20
5.4.4.2 Lokale Handschrifterkennung 21
5.4.4.3 Online-Handschrifterkennung 21
5.4.4.4 Video Schrifterkennung 21

5.4.5 Punkte zuordnen ... *21*
5.4.5.1 Marker ... 21
5.4.5.2 Seitenzahl erkennen ... 21
5.4.6 Fehler korrigieren ... *21*
5.4.7 Persistenz ... *22*
5.4.8 User Interface ... *22*
5.4.8.1 Design ... 22
5.4.8.2 User Experience ... 22
5.4.9 Plattform & Infrastruktur ... *23*
5.5 UMSETZUNG ... 24
5.5.1 Handschrifterkennung ... *24*
5.5.2 Plattform & Infrastruktur ... *25*
5.5.3 Punkte zuordnen ... *28*
5.5.3.1 Marker ... 28
5.5.3.2 Seitenzahl mit Texterkennung auslesen ... 28
5.5.4 Marker Implementation ... *29*
5.5.4.1 ArUco & Qr ... 29
5.5.4.2 AruCo Optimierung ... 31
5.5.5 Perspektivenkorrektur ... *31*
5.5.6 Verhalten bei mehreren Seiten ... *32*
5.5.7 User Interface Implementation ... *33*
5.6 USABILITY-TESTS ... 35
5.6.1 Tests ... *35*
5.6.2 Erkenntnisse und Massnahmen ... *36*
5.7 ERGEBNISDISKUSSION ... 37
5.7.1 Eingetretene Risiken ... *37*
5.7.1.1 Verfügbare Webcams ... 37
5.7.1.2 QR Erkennung ... 37
5.7.2 Weitere Probleme ... *37*
5.7.2.1 Handschrifterkennung ... 37
5.7.2.2 Performance beim PDF generieren ... 37
5.8 ZUSAMMENFASSUNG UND AUSBLICK ... 39
5.8.1 Automatische Korrekturen ... *39*
5.8.2 Texterkennung der Antworten ... *39*
5.8.3 Prüfung drucken ... *39*
5.9 QUELLEN / LITERATURVERZEICHNIS ... 40

6. ANHÄNGE ... 41

6.1 SOFTWARE ENGINEERING ... 41
Anhang A: User Manual ... 41

2. Aufgabenstellung

Bachelorarbeit «Digitale Korrekturunterstützung»

Einführung

Das Zusammenzählen von Punkten in Prüfungen ist für Dozierende aufwändig und fehleranfällig. Bei 100 Prüfungen dauert das Zählen (und Summieren) allein schon etwa 2h. Eine erhebliche Vereinfachung ist die Verwendung von Excel, um das Summieren zu automatisieren. Dennoch besteht weiterhin die Gefahr, dass beim Übertrag vom Papier in das Excel-Sheet Fehler geschehen.

Aufgabe

In dieser Studienarbeit sollen die Studierenden eine App zum automatischen Erfassen und Zusammenzählen von Punkten entwickeln. Die App soll auf einem Mobilgerät oder Laptop laufen, das dauerhaft per Kamera die Korrektur beobachtet. Sobald Punkte (in roter Farbe) vergeben werden, sollen diese in das HSR-Noten-Excelsheet an die richtige Stelle übertragen werden.

Folgende zusätzliche Aufgabenstellungen können mit bearbeitet werden:

- Abgleich mit dem ausgedruckten Notenblatt
- Automatische Bewertung von einfachen Aufgabentypen (Multiple-Choice, Rechnungen)
- Digitalisierung der Antworten (gerade der nicht lesbaren)

Die Studierenden sollen dabei strukturiert und ingenieurmässig vorgehen und wissenschaftlich nachvollziehbare Untersuchungen durchführen und dokumentieren. In der Wahl der Technologien besteht keine Einschränkung.

Termine

Die Studienarbeit beginnt am 19.2.2020. Abgabetermin ist der 12.6.2020.

Betreuung

Die Studienarbeit wird durch Prof. Stefan Richter betreut. Jeden Mittwoch von 15:00 bis 16:00 findet eine Besprechung statt, in der die Studierenden den Fortschritt der Arbeit präsentieren.

Fragen und Angelegenheiten können ausserhalb dieses Termins auch per E-Mail erörtert werden.

Bewertung

Die Arbeit wird anhand der folgenden fünf gleichgewichteten Punkte bewertet:

1. Organisation, Durchführung (Projektplanung u. Nachführung Arbeit gemäss Projektplan, Selbständigkeit, Einsatz, Zusammenarbeit mit Auftraggeber, Betreuer)
2. Bericht (Inhalt des Projektschlussberichts, Gliederung, Darstellung, Sprache der gesamten Dokumentation)
3. Problemanalyse (Vorstudie, Literaturstudium, Anforderungsspezifikation, Anforderungsanalyse, Domainanalyse)

4. **Lösungsentwurf (Lösungsvarianten und deren Beurteilung, Variantenentscheid, Konzept, Entwurf)**
5. **Realisierung und Test**
6. **Präsentation der Bachelorarbeit**

Hinweise

Die folgende Seite bietet zahlreiche nützliche Informationen zum Schreiben einer wissenschaftlichen oder technischen Arbeit:

https://www.ifs.hsr.ch/index.php?id=13194&L=4metadata%2Foai_dc_1.dc

3. Abstract

3.1 Ziel der Arbeit

Durch diese Arbeit soll den Dozenten an der HSR die Korrektur von Prüfungen erleichtert werden. Nach der Bewertung der einzelnen Aufgaben müssen die Punktzahlen aller Aufgaben zusammengezählt werden. Diese Arbeit soll den Dozenten von unserer Lösung abgenommen werden. Es geht dabei um Prüfungen, die auf Papier gelöst und korrigiert werden. Die Punktzahl, die ein Dozent für die jeweilige Aufgabe vergibt, wird in Echtzeit gelesen. Zum Speichern wird die Punktzahl automatisch einer Prüfungsseite und einem Studenten zugeordnet. Die Punktzahlen werden in Echtzeit angezeigt, damit der Dozent bei einer falsch gelesenen Zahl sofort eingreifen kann. Mithilfe des importierten Notenschlüssels wird aus der Gesamtpunktzahl eine Notenübersicht generiert, die so dem Sekretariat abgegeben werden kann.

3.2 Vorgehen / Technologien

Ein zentraler Bestandteil der Software ist die Handschrifterkennung. Wir haben in Prototypen verschiedene Technologien ausprobiert und uns wegen Zuverlässigkeit, Kosten und API für den Cloud-Service «Google Vision»[1] entschieden. Damit erkannt wird, welche Prüfung momentan korrigiert wird, drucken wir auf jede Seite ArUco-Codes[2]. ArUco Codes sind mit einem schiefen Kamerawinkel oder schlechter Auflösung deutlich besser lesbar als QR-Codes. Nebst der Identifikation der aktuellen Prüfung wird der ArUco-Code auch verwendet, um die Kameraperspektive zu korrigieren, damit ein unverzerrtes Bild an die Texterkennung geschickt werden kann. Als Plattform setzen wir .NET-Framework ein und programmieren in C#, weil für diese Technologie viele «3rd Party»-Libraries verfügbar sind. Zur Bildmanipulation setzen wir OpenCV[3] ein, die Webcam benutz AForge[4] und zur PDF-Bearbeitung verwenden wir PdfSharp[5].

---

[1] *https://cloud.google.com/vision/docs/ocr*

[2] *https://www.uco.es/investiga/grupos/ava/node/26*

[3] *https://opencv.org/*

[4] *http://www.aforgenet.com/*

[5] *http://www.pdfsharp.net/*

4. Management Summary

4.1 Ausgangslage

Mit dieser Arbeit soll eine Software entwickelt werden, die bei der Korrektur von Prüfungen unterstützt. Es ist nicht das Ziel, die Korrektur vollständig zu automatisieren. Die Aufgaben müssen immer noch von Hand bewertet werden.

Während der Korrektur sollen die Punktzahlen auf der Prüfung von einer Kamera aufgenommen und zusammengezählt werden. Nachdem alle Prüfungen von allen Studenten korrigiert wurden, soll eine Gesamtübersicht ausgedruckt werden können.

4.2 Vorgehen

Bei dieser Arbeit geht es prioritär darum, bestehende Technologien zu einer neuen Gesamtlösung zusammenzusetzen. Wir haben uns deshalb in einem ersten Schritt hauptsächlich auf die Evaluation von bestehenden Technologien konzentriert. Vor allem für die Handschrifterkennung gibt es viele verschiedene mögliche Ansätze.

Wo notwendig haben wir Ideen mit Prototypen ausprobiert, damit wir uns ein möglichst gutes Bild davon machen konnten.

Erst als wir uns grundsätzlich für eine Technologie entschieden haben, haben wir mit der Implementation unserer Lösung begonnen.

4.3 Technologie

4.3.1 Handschrifterkennung

Bei der Handschrifterkennung haben wir uns für den Cloud-Dienst Google Vision AI[6] entschieden. Hauptsächlich ausschlaggebend bei der Entscheidung für einen Cloud-Dienst waren die hohe Genauigkeit und der vergleichsweise geringe Entwicklungsaufwand für uns. Dagegengesprochen hat die lange Antwortzeit, und die Kosten.

4.3.2 Bildmanipulation

Das Kamerabild ist bei der Aufnahme oft verdreht und verzerrt. Das muss vor der Handschrifterkennung lokal auf dem Gerät korrigiert werden. Für Perspektivenkorrektur und Bildmanipulation haben wir uns für OpenCV[7] entschieden.

---

[6] *https://cloud.google.com/vision/docs/ocr*

[7] *https://opencv.org/*

4.4 Ergebnisse

Durch die Integration verschiedener bestehender Komponenten konnten wir eine funktionierende Softwarelösung erstellen.

Mit dem Einsatz eines Cloud-Dienstes gingen wir zwar einen Kompromiss bei der Wartezeit ein, können aber durch die so gewonnene bessere Zuverlässigkeit insgesamt trotzdem Korrekturzeit einsparen.

5. Technischer Bericht

5.1 Ausgangslage

Das Ziel dieser Arbeit ist es, eine Applikation zu entwickeln, welche Dozenten und anderen Lehrpersonen eine Unterstützung für das Korrigieren von Prüfungen bietet.

Das Manuelle korrigieren soll dadurch effizienter und weniger fehleranfällig werden.

Diese Arbeit geht von einer bis anhin manuellen Prüfungskorrektur aus, in der die Prüfungsaufgaben einzeln bewertet werden und das Total aller Punkte auf jeder Seite zusammengezählt und auf der Prüfung notiert wird.

Die Aufgabenstellung gibt dabei folgenden Kontext und die erwarteten Funktionalitäten der App vor:

In dieser Studienarbeit sollen die Studierenden eine App zum automatischen Erfassen und Zusammenzählen von Punkten entwickeln. Die App soll auf einem Mobilgerät oder Laptop laufen, das dauerhaft per Kamera die Korrektur beobachtet. Sobald Punkte (in roter Farbe) vergeben werden, sollen diese in das HSR-Noten-Excelsheet an die richtige Stelle übertragen werden.

Ausschnitt aus 2. 2. Aufgabenstellung

Der Fokus dieser Arbeit liegt somit auf der Erstellung eines Lösungskonzeptes und dessen Umsetzungen. Um eine optimale Lösung zu erreichen, wird dabei Wert auf eine gründliche Recherche und Evaluation von vorhandenen Tools und Dienste gelegt.

5.2 Begriffe

5.2.1 Offline Handschrifterkennung

Bei der offline Handschrifterkennung besteht als Ausgangslage ein Bild des handschriftlichen Textes. Das kann ein einzelnes Zeichen, ein Wort oder ein ganzer Text sein. Die offline Handschrifterkennung versucht, aufgrund des Bildes, herauszufinden was darauf abgebildet ist.

Es spielt dabei keine Rolle, ob Remote-Dienste involviert sind. Offline Handschrifterkennung kann lokal auf dem Gerät oder von einem Cloud Service durchgeführt werden.

Die Methode zur Handschrifterkennung spielt ebenfalls keine Rolle. Offline Handschrifterkennung kann Pattern-Recognition, Feature-Extraction oder künstliche neuronale Netze verwenden und die Resultate mithilfe von Wörterbüchern verbessern.

5.2.2 Online Handschrifterkennung

Bei der online Handschrifterkennung besteht als Ausgangslage die Information, in welcher Reihenfolge und Richtung die Linien gezeichnet worden sind. Mit diesen zusätzlichen Informationen ist eine viel höhere Zuverlässigkeit möglich als bei offline Handschrifterkennung.

Es spielt keine Rolle, ob Remote-Dienste involviert sind. Online Handschrifterkennung kann lokal auf dem Gerät oder von einem Cloud Service durchgeführt werden.

Online Handschrifterkennung setzt voraus, dass während dem Schreiben, die zusätzlichen Informationen, wie die Reihenfolge der Striche, festgehalten werden können. Z.B. indem auf einem Touch-Display geschrieben wird.

5.2.3 Codes und Marker

QR-Codes sind zweidimensionale grafische Codes, die Informationen als Muster codiert darstellen.

QR-Code

ArUco-Maker sind eine Weiterentwicklung von QR-Codes. Sie funktionieren nach denselben Prinzipien, sind jedoch optimiert für Augmented Reality Applikationen.

ArUco-Marker

Die Begriffe Code und Marker werden in dieser Arbeit synonym verwendet und meinen (falls nicht ausdrücklich als QR-Code oder ArUco-Marker bezeichnet) eine beliebige, QR-ähnliche Implementation.

5.3 Problembeschreibung

Es gilt möglichst viele Tasks des manuellen Korrekturvorgangs zu automatisieren, um diesen zu beschleunigen. Dabei soll darauf geachtet werden, dass möglichst wenig zusätzlicher Aufwand durch den Einsatz der App für den User entsteht.

Der Haupt-Task, den es zu Automatisieren gilt, ist das Zusammenzählen der Punkte. Alle weiteren Funktionalitäten, wie beispielsweise der Export der Noten und die Überprüfung auf Vollständigkeit, sind von diesem Task abhängig.

5.3.1 Use Cases

5.3.1.1 Übersicht

Primary Actor	Phase	Use Cases
User (Dozent)	Vor Prüfungssession	UC1: Druck-PDF generieren
	In der Korrekturphase	UC2: Korrektursession CRUD UC3: Erkannte Punkte Anzeigen UC4: Live Addition UC5: Notenschlüssel erstellen UC6: Notenblatt exportieren UC7: Zu korrigierende Seiten anzeigen Optional (Nicht ausspezifiziert) UC8: Multiple-Choice-Aufgaben auswerten UC9: Antworten als Text speichern

5.3.1.2 UC1: Druck-PDF generieren

UC 1	
Name	Druck PDF einer Prüfung generieren lassen
Primary Actor	User
Main Success Scenario	1. User importiert Prüfung als PDF 2. User wählt die Anzahl benötigter Kopien 3. User wählt den Speicherort für das generierte, getagte PDF
Frequency of Occurrence	Einmal pro Prüfung
Miscellaneous	Das generierte PDF enthält QR tags, damit die Seiten später von der App identifiziert werden können.

5.3.1.3 UC2: Korrektursession CRUD

UC 2	
Name	Korrektursessionen erstellen, aktivieren, ändern und löschen
Primary Actor	User
Main Success Scenario	1. User öffnet eine bestehende Korrektursession aus der Liste
Extensions	Abweichung nach MSS 1: a. User erstellt eine neue Korrektursession b. User löscht eine Korrektursession c. User ändert eine Korrektursession in dem er sie öffnet und die entsprechenden Felder (Name, Semester, Jahr) anpasst.
Frequency of Occurrence	Mehrmals pro Session

5.3.1.4 UC3: Erkannte Punkte Anzeigen

UC 3	
Name	Alle erkannten Punkte auf einer Seite anzeigen
Primary Actor	User
Preconditions	1. User verwendet eine getagte Prüfung 2. User hat eine Korrektursession geöffnet
Postconditions	1. Erkannte Punkte wurden gespeichert
Main Success Scenario	1. User legt eine Prüfungsseite unter die Kamera 2. User korrigiert die Aufgaben und schreibt Punktzahlen auf die Prüfung 3. User sieht und überprüft die erkannten Punktzahlen in der App
Extensions	Abweichung nach MSS 3: a. Der User stellt fest, dass nicht alle Punktzahlen korrekt erkannt wurden. b. Der User kann manuell die fehlenden Punktzahlen hinzufügen c. Der User kann falsch erkannte Punktzahlen ändern
Frequency of Occurrence	Mehrmals pro Session / stetig

5.3.1.5 UC4: Live Addition

UC 4	
Name	Live Addition aller Punkte auf einer Seite anzeigen
Primary Actor	User
Preconditions	1. User verwendet eine getagte Prüfung 2. User hat eine Korrektursession geöffnet
Main Success Scenario	1. User legt eine korrigierte Prüfungsseite unter die Kamera 2. User kann Gesamtpunktzahl in der App ablesen und auf die Prüfung übertragen
Frequency of Occurrence	Mehrmals pro Session / stetig
Miscellaneous	Die erkannten Punkte gelten als korrekt sobald der User die Gesamtpunktzahl auf die Prüfung überträgt.

5.3.1.6 UC5: Notenschlüssel erstellen

UC 5	
Name	Notenschlüssel erstellen
Primary Actor	User
Preconditions	1. User hat eine Korrektursession geöffnet
Postconditions	1. Notenschlüssel ist persistiert und kann angezeigt werden
Main Success Scenario	1. User klickt auf Notenschlüssel erstellen 2. User weist einzelnen Punktzahlen Noten zu 3. User definiert auf welchen Bruch einer ganzen Note gerundet werden soll 4. User speichert den Notenschlüssel
Extensions	Abweichung nach MSS 4: a. Zuweisung ist nicht eindeutig b. User wird durch Meldung über ungültige Werte informiert c. Speicher Button ist deaktiviert
Frequency of Occurrence	Einmal pro Session

5.3.1.7 UC6: Notenblatt exportieren

UC 6	
Name	Notenblatt exportieren
Primary Actor	User
Preconditions	1. User hat eine Korrektursession geöffnet 2. User hat in der aktuellen Korrektursession einen Notenschlüssel definiert
Postconditions	1. Notenblatt PDF ist am gewählten Speicherort gespeichert
Main Success Scenario	1. User klickt auf Notenblatt exportieren 2. User wählt den Speicherort 3. User erhält Erfolgsmeldung
Extensions	Abweichung nach MSS 2: a. Es wurden nicht alle Punktzahlen bestätigt, es wurden nicht alle Seiten bewertet oder es fehlen Namen der Studenten b. User wird durch Meldung über fehlende Werte informiert c. Warnung kann ignoriert werden
Frequency of Occurrence	Einmal pro Session

5.3.1.8 UC7: Zu korrigierende Seiten anzeigen

UC 7	
Name	Noch zu korrigierende Seiten anzeigen
Primary Actor	User
Preconditions	1. User hat eine Korrektursession geöffnet
Main Success Scenario	1. User sieht den aktuellen Stand der Session: a. Anzahl erkannte Studenten b. Anzahl erkannte Seiten c. Anzahl bestätigte Seiten d. Noch zu bestätigende Punktzahlen e. Noch zu korrigierende Seiten
Frequency of Occurrence	Kontinuierlich während der Session

5.3.1.9 UC8: Multiple-Choice-Aufgaben auswerten

Optional – Nicht ausspezifiziert

Der User hat die Möglichkeit Multiple-Choice-Aufgaben zu erstellen, die Lösung zu definieren und die Bewertung festzulegen.

Bei der Korrektursession werden bei diesen Aufgaben die Antworten der Studenten erkannt und gemäss definierten Bewertungsschema bewertet.

5.3.1.10 UC9: Antworten als Text speichern

Optional – Nicht ausspezifiziert

Der User kann die Antworten der Studenten digital als Text anzeigen und speichern.

5.3.2 Non-Functional Requirements

Kategorie	Anforderung	Zielwerte		
		M1 minimal	M2 target	M3 outstanding
Performance	Dauer für die live Erkennung und Addition von Punkten	< 4s	< 2s	< 1s

Um einen Mehrwert durch die Nutzung der App zu erhalten, muss die Verarbeitung des Video-Streams performant sein. Es gilt, die Dauer des manuellen Punkte-Addierens zu unterbieten.

Kategorie	Anforderung	M1	M2	M3
Usability	Möglichst wenig Interaktionen während der Prüfungskorrektur	Eine Bestätigung pro Seite und in Ausnahme-fällen	Nur in Ausnahme-fällen	Keine UI Interaktionen

User Interaktionen brauchen Zeit und sollen deshalb so weit wie möglich reduziert werden. Jede zusätzlich nötige Interaktion, welche durch den Einsatz der App entsteht, verringert den Mehrwert.

Kategorie	Anforderung	M1	M2	M3
Reliability	Erkennungsrate der Punktzahlen sollte hoch sein	> 75% korrekt	> 90% korrekt	> 98% korrekt
	Datenschutz	Verschlüsselte Datenüber-tragung an Drittanbieter	Nur anonyme Datenüber-tragung an Drittanbieter	Keine Daten-übertragung an Drittanbieter

Eine hohe Erkennungsrate ist ein weiterer Faktor, der den Mehrwert der App beeinflusst. Werden zu wenig Punkte erkannt, lohnt sich der Einsatz der App nicht.

Der Datenschutz muss gewährleistet sein. Personenbezogene Daten sollten im Idealfall nur lokal verarbeitet werden. Wo dies nicht ausreicht, muss die Übertragung verschlüsselt stattfinden und die Daten sollten möglichst anonymisiert werden.

Kategorie	Anforderung	M1	M2	M3
Supportability	Sollte mit möglichst vielen Prüfungsformen kompatibel sein	Kompatibel zu spezifischen Prüfungen im LaTeX-Format	Kompatibel zu allen Prüfungen im PDF-Format mit Vorschriften ans Format	Kompatibel zu allen Prüfungen im PDF-Format ohne Vorschriften ans Format
	Sollte mit möglichst vielen Webcams kompatibel sein	Kompatibel zu mindestens einer getesteten Webcam	Kompatibel zu allen getesteten Webcams	

Ziel ist, möglichst wenig Anforderungen an das Format der Prüfungen zu stellen, um den durch die App generierten Zusatzaufwand für den Dozenten zu minimieren.

Kategorie	Anforderung	M1	M2	M3
Implementation Requirement	Plattform	Windows / Android		
	Programmiersprachen	C# / Java		

Die Plattformanforderungen ergeben sich durch die uns zur Verfügung stehenden Geräte.
Als Programmiersprachen kommen jene infrage, bei denen alle Gruppenmitglieder genügend Erfahrung mitbringen, um einen effizienten Projektablauf zu ermöglichen.

5.3.3 Problembereiche

Für folgende Themenbereiche muss eine Lösung gefunden werden.

5.3.3.1 Prüfung filmen

Während der Dozent die Prüfung korrigiert, muss die Applikation in Echtzeit den Korrekturvorgang filmen, um die Punktzahlen auf der Prüfung zu lesen. Es darf kein zusätzlicher Arbeitsschritt für den Dozenten entstehen, wie z.B. die Prüfung nach der Korrektur zu scannen.

5.3.3.2 Perspektive korrigieren

Wenn die Kamera nicht exakt über dem Mittelpunkt der Prüfung positioniert wird, ist eine Korrektur der Kameraperspektive notwendig, weil das Prüfungsblatt sonst auf dem aufgenommenen Bild nicht rechteckig ist. Das würde die Orientierung auf der Prüfung und das zuordnen von Punktzahlen zu Aufgaben erschweren. Handschrifterkennung mit verzogenen Zahlen wäre ebenfalls weniger zuverlässig.

5.3.3.3 Punkte auf der Prüfung suchen

Auf der aktuellen Prüfungsseite müssen alle Punktzahlen, die der Dozent notiert, gefunden werden, um sie anschliessend zu verarbeiten. Nur der Dozent schreibt mit roter Farbe auf die Prüfung. Er schreibt aber Punktzahlen und Kommentare mit roter Farbe.

5.3.3.4 Handschrifterkennung

Wenn die Punktezahl auf der Prüfung gefunden wurde, muss dieser Bildausschnitt mithilfe einer Handschrifterkennung ausgelesen werden.

5.3.3.5 Punkte zuordnen

Wenn eine Prüfungsseite zum zweiten Mal unter der Kamera liegt, dürfen die Punkte auf dieser Seite nicht zusätzlich gutgeschrieben werden. Deshalb müssen die Punkte, die auf der aktuellen Prüfungsseite erkannt werden, einem Studenten und einer Prüfungsseite zugeordnet werden können.

5.3.3.6 Punkte manuell korrigieren

Wenn die Handschrifterkennung einen Fehler macht, muss der Dozent das mit einem möglichst kleinen Aufwand korrigieren können.

5.3.3.7 Persistenz

Eine Korrektursession muss unterbrochen werden können. Alle Daten müssen bei der Wiederaufnahme der Korrektur genau gleich vorhanden sein.

5.3.3.8 User-Interface

Das User-Interface muss dem Dozenten als Hilfsmittel dienen und darf den Korrekturprozess nicht durch unnötige zusätzliche Interaktionen verlangsamen. Da der Dozent bei jeder einzelnen Seite Informationen von der App benötigt, sollten diese immer prominent ersichtlich sein.

5.4 Lösungskonzept

5.4.1 Prüfung filmen

Wir haben uns zwei verschiedene Varianten zum Filmen der Prüfung genauer angesehen. Einerseits eine USB-Webcam direkt am Computer angeschlossen und andererseits die Handykamera, die das Video über ein Netzwerkprotokoll an den Computer überträgt.

Die Smartphone-Kamera hätte den Vorteil, dass die meisten User bereits ein solches Gerät besitzen. Für die Texterkennung ist ein qualitativ hochwertiger und hochauflösender Videostream entscheidend. Die persönliche Erfahrung hat gezeigt, dass Live-Video-Streams von Smartphones aus (per WLAN-Verbindung) oft problematisch sind und vor allem bei hohen Bitraten schnell Engpässe entstehen.

Der Einsatz von Webcams ist zu bevorzugen, da wir uns so eine bessere Bildqualität erhoffen und nicht auf performante WLAN-Netzwerke angewiesen sind.

5.4.2 Perspektive korrigieren

Für eine zuverlässige Punkte-Erfassung und zur Vermeidung von Mehrfacherfassungen, erachten wir es als essenziell, die präzisen Positionsdaten der gefilmten Seiten zu erlangen.

Mit Hilfe dieser Positionsdaten ist es uns auch möglich, die Kameraperspektive zu korrigieren und ein Prüfungsblatt ohne Verzerrung im UI anzuzeigen.

Für die Positionserkennung bieten sich QR-Codes oder QR-Code ähnliche Marker an. Diese Marker sind meist schwarz-weisse Grafiken mit speziellen Merkmalen, die eine einfache Identifizierung trotz perspektivischer Verzerrung in einem Bild ermöglichen. Die meisten dieser Marker erlauben auch das Hinterlegen von Binärdaten, die dann als Muster im Marker codiert werden.

Wir entschieden uns, die Positionsdaten der einzelnen Seiten mithilfe solcher Marker auszulesen. In der Theorie genügt bereits ein solcher Marker auf der Seite, um die Perspektive der Seite zu berechnen.

Unser Konzept sieht vor, in der App die Möglichkeit zu bieten, eine Prüfung mit diesen Markern zu versehen und als PDF-Datei abzuspeichern.

Gegenüber einer Variante ganz ohne Muster auf den Seiten, besteht der Vorteil einer präziseren und zuverlässigeren Datenerfassung.

5.4.3 Punkte auf der Prüfung suchen

Wir untersuchten zwei verschiedene Konzepte, welche uns die Ortung der Punktzahlen ermöglichen.

Gemäss Aufgabenstellung und Vorgaben besteht die Möglichkeit, die Voraussetzung zu definieren, dass alle Punktzahlen in roter Farbe geschrieben werden müssen. Dies eröffnet die Möglichkeit, im Kamerabild durch Farbfilter die Punktzahlen zu orten.

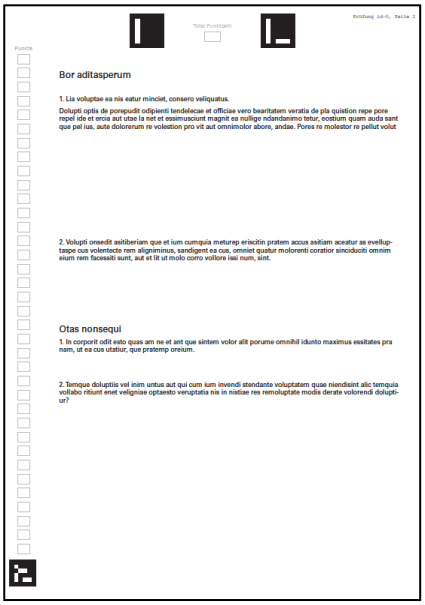
Ebenfalls besteht die Möglichkeit, auf der Prüfung gewisse Bereiche für die Punktebewertung zu definieren.

Versuche zeigten, dass das Isolieren von mit roter Farbe geschriebenen Zahlen problematisch sein kann. Um eine genaue Isolation zu erhalten, muss der Farbunterschied genügend gross sein. Hier gibt

es viele Faktoren, welche dies beeinflussen. Zu diesen zählen die Farbechtheit der Kamera, die Lichtverhältnisse bei der Aufnahme, die Weissabgleicheinstellungen der Kamera, die Farbe des Papiers und nicht zuletzt die Farbtöne der verwendeten Stifte.

Gewisse Faktoren liessen sich durch Anwenden von Bildmanipulationen optimieren. So könnte zum Beispiel ein automatischer Weissabgleich (eine Farbkorrektur) in der App durchgeführt werden.

Aus Sicht der User-Experience wäre die Lösung mit der roten Farbe zu bevorzugen, da dem Dozenten so weniger Vorgaben an sein Prüfungslayout gemacht werden. Nach Abwägung der beiden Varianten entschieden wir uns für das Konzept der vordefinierten Bereiche und gegen die Verwendung der roten Farbe. Entscheidend war, neben dem zusätzlichen Aufwand für die Implementation der Farbkorrekturen und Ortung, auch die geschätzte Zuverlässigkeit der Lösung. Im Gegensatz zu vordefinierten Bereichen bleibt bei der Farbfilter-Lösung immer eine gewisse Fehleranfälligkeit durch die vielen Faktoren bestehen.



Prüfungsseite mit Punktebereich am linken Rand

Unser Konzept sieht vor, einen Bereich am linken Seitenrand für die Punktzahlen zu reservieren. Dieser Bereich, dargestellt durch einzelne Felder, streckt sich über die gesamte Höhe der Seite. Hinzugefügt werden diese Felder im selben Schritt wie die Marker.

5.4.4 Handschrifterkennung

Zur Handschrifterkennung gibt es verschiedene Möglichkeiten. Vier Konzepte sind nachfolgend genauer erläutert.

5.4.4.1 Cloud-Dienst

Verschiedene Cloud Dienstleister bieten einen Service zur Handschrifterkennung an. Auffallend ist, dass AWS kein Produkt enthält, das speziell auf Handschrifterkennung ausgerichtet ist. Bei Azure und GCloud sind schon Produkte in der Vollversion vorhanden, beide entwickeln aber momentan aktiv an einer neuen Version mit neuer API.

Diese Vorabversionen kommen für uns nicht infrage, weil die Preview API jederzeit abgestellt werden könnte, wodurch unser Programm unbrauchbar wird.

5.4.4.2 Lokale Handschrifterkennung

Für Handschrifterkennung auf dem lokalen Gerät bietet sich Tesseract an. Tesseract ist ein Framework für Handschrifterkennung mit Machine-Learning. Damit Tesseract gut funktioniert, muss das Bild vorverarbeitet werden. Text darf nicht schräg stehen und muss genügend gross sein. Weitere Korrekturen bezüglich Helligkeit und schwarzen Flächen sind ebenfalls notwendig.

5.4.4.3 Online-Handschrifterkennung

Online Handschrifterkennung würde bedeuten, dass wir nicht nur die fertige Zahl erkennen lassen, sondern zur Erkennung auch die Information in welche Richtung die Striche gezeichnet wurden benutzen. Diese Information mit einer Kamera aufzunehmen ist sehr anspruchsvoll. Für Online-Handschrifterkennung müsste eher ein spezieller Stift oder eine spezielle Unterlage verwendet werden.

5.4.4.4 Video Schrifterkennung

Es ist auch möglich, ein ganzes Video an Cloud-Dienste zu schicken. Als Resultat bekommt man dann den Text zurück, der in diesem Video erkannt wurde. Diese Variante löst auch das Problem, dass die Zahlen während einiger Zeit verdeckt sind. Für unser Projekt kommt diese Variante allerdings nicht infrage, weil die Antwort bei den evaluierten Anbietern nicht in Echtzeit zurückkommt.

5.4.5 Punkte zuordnen

5.4.5.1 Marker

Um die Prüfung einem Studenten zuzuordnen, werden auf jeder Seite für Computer einfach lesbare Marker angebracht. Hierfür bieten sich QR-Codes oder ähnliche Marker an.

Um die Punktzahl einer genauen Seite zuzuordnen, können ebenfalls extra Marker für die Seitenzahl angebracht werden oder es können die schon bestehenden Marker verwendet werden.

5.4.5.2 Seitenzahl erkennen

Um die Punkte einer Seitenzahl zuzuordnen, kann die Seitenzahl, die für Menschen sowieso schon aufgedruckt wird, verwendet werden. Diese Seitenzahl müsste mithilfe einer Texterkennung erkannt werden. Zuverlässiger ist es, auch hierfür Marker zu verwenden.

5.4.6 Fehler korrigieren

Um dem Dozenten die Möglichkeit zu geben, falsch erkannte Punkte zu korrigieren, werden ihm im UI alle erkannten Punkte der aktuellen Seite angezeigt und er hat die Möglichkeit diese zu editieren, löschen und zu ergänzen.

Um zu verhindern, dass vom Dozenten als korrekt befundene Ergebnisse zu einem späteren Zeitpunkt von der App verfälscht werden (durch neue, fehlerhafte Texterkennungsresultate), können einzelne Seiten in der App bestätigt werden. Ab diesem Zeitpunkt findet keine Texterkennung auf dieser Seite mehr statt.

Um für diese Bestätigung nicht jedes Mal mit dem UI interagieren zu müssen, kann der Dozent auch das Total der Punktzahlen in einem speziell dafür gedruckten Feld oben auf der Prüfung notieren.

Dieses Feld läuft, wie auch die Felder für die einzelnen Punkte, durch die Texterkennung. Erkennt die App das Total in diesem Feld, gilt die Seite als bestätigt.

5.4.7 Persistenz

Als Datenspeicher haben wir uns für eine lokale Datenbank entschieden. Mit SqLite steht uns eine einfache Lösung zur Verfügung, ein Datenbanksystem in unsere Software integrieren. So sind keine externen Abhängigkeiten zu einem Datenbanksystem in der IT-Abteilung oder einer Cloud Datenbank notwendig. Das macht die Installation für einen Anwender deutlich einfacher.

Das Risiko eines Datenverlustes durch Verwenden einer lokalen Datenbank, ist in unserem Falle vernachlässigbar. Mit unserer Software sparen wir dem Anwender maximal 2 Stunden Arbeit. Das ist die Zeit, die notwendig wäre um die Punkte von Hand zusammenzuzählen. Der mögliche Schaden ist also sehr gering.

5.4.8 User Interface

5.4.8.1 Design

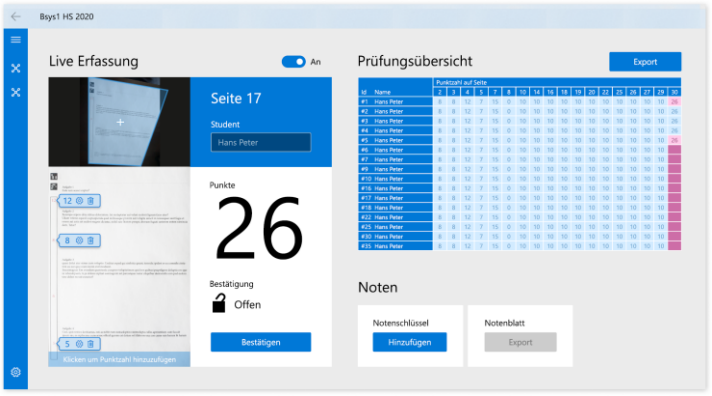
Da wir für die Windows-Plattform entwickeln, verwenden wird ein eng an das offizielle Windows Metro Design[8] angelehntes Designkonzept. Wir erhalten dadurch eine moderne Oberfläche mit den für die User bereits bekannten Windows UI-Elementen.

Designkonzept - Homescreen

5.4.8.2 User Experience

Einfache Bedienung und keine unnötigen Klicks sind die wichtigsten Ziele unserer Benutzeroberfläche. Um dies zu erreichen, haben wir bewusst die Screens auf das Wesentliche beschränkt und die Funktionalitäten so gebündelt, dass kein Wechsel zwischen Screens während dem Korrigieren notwendig ist.

[8] *https://docs.microsoft.com/en-us/windows/uwp/design/*



Designkonzept - Dashboard

Besonderes Augenmerk soll auf die UI Elemente rund um die Live-Punkte-Anzeige gelegt werden. Um eine möglichst rasche Überprüfung der Punkte zu ermöglichen, soll im UI ein Livestream der aktuellen Seite angezeigt und dieser mit einem interaktiven Overlay ergänzt werden. Im Overlay werden alle erkannten Punkte der aktuellen Seite angezeigt. Dies ermöglicht eine einfache und schnelle visuelle Kontrolle der Punkte. Die Werte sollen direkt in diesem Overlay geändert, gelöscht oder ergänzt werden können.

5.4.9 Plattform & Infrastruktur

Als Programmierplattform hatten wir folgende Ideen:

- Python
- C
- C++
- C#
- Web-App

Auf die Auswahl einer geeigneten Plattform wird im nachfolgenden Kapitel «Umsetzung» genauer eingegangen.

Folgende Infrastrukturen könnten verwendet werden:

- PC mit USB Webcam
- PC mit Handykamera über USB
- PC mit Handykamera über Netzwerk

5.5 Umsetzung

5.5.1 Handschrifterkennung

Bei der Lösung zur Handschrifterkennung haben wir uns für den «Vision AI»-Dienst der Google-Cloud entschieden.

Zuerst mussten wir uns grundsätzlich für eine Technologie entscheiden.

Bei der Online-Handschrifterkennung spricht stark dagegen, dass für die Korrektur zusätzliche Hardware notwendig wäre. Sonst könnte nicht erfasst werden, in welche Richtung und in welcher Reihenfolge Striche gezeichnet wurden.

Somit blieb uns noch die Wahl zwischen Cloud-Diensten und Tesseract. Die Zuverlässigkeit der Ergebnisse haben wir sehr hoch gewichtet. Recherchen zeigten, dass es unwahrscheinlich ist, mit einer eigenen auf Tesseract basierender Implementation und begrenzten Ressourcen an die Zuverlässigkeit eines Cloud-Dienstes heranzukommen. In einer anderen Arbeit hat Kenneth Soo 92.2 % Genauigkeit erreicht (SooK, 2016). Für unsere eigenen Tests haben wir folgende Trainingsdaten verwendet: (Bentham Collection), (IAM Handwriting Database), (Washington Database) und (Saint Gall Database). Wir haben diesen Ansatz aber nicht lange verfolgt und nur eine sehr tiefe Erkennungsrate erreicht.

Für GCloud gegenüber Azure haben wir uns entschieden, weil Google etwas günstiger ist als Microsoft *(Google, 2020) (Microsoft, 2020)*.

Folgend, die Bewertungsraster, nach denen wir uns richteten:

Azure	Gewichtung	Bewertung	Gewichtete Bewertung
Entwicklungsaufwand	-3	1	-3
Zuverlässigkeit	5	3	15
Geschwindigkeit	2	2	4
Zukunftssicherheit	2	2	4
Kosten im Betrieb	-3	4	-12
Korrekturaufwand	-4	2	-8
		Total	0

GCloud	Gewichtung	Bewertung	Gewichtete Bewertung
Entwicklungsaufwand	-3	1	-3
Zuverlässigkeit	5	3	15
Geschwindigkeit	2	2	4
Zukunftssicherheit	2	2	4
Kosten im Betrieb	-3	3	-9
Korrekturaufwand	-4	2	-8
		Total	3

Tesseract	Gewichtung	Bewertung	Gewichtete Bewertung
Entwicklungsaufwand	-3	5	-15
Zuverlässigkeit	5	1	5
Geschwindigkeit	2	4	8
Zukunftssicherheit	2	5	10
Kosten im Betrieb	-3	0	0
Korrekturaufwand	-4	2	-8
		Total	0

Online Handschrifterkennung	Gewichtung	Bewertung	Gewichtete Bewertung
Entwicklungsaufwand	-3	4	-12
Zuverlässigkeit	5	5	25
Geschwindigkeit	2	2	4
Zukunftssicherheit	2	2	4
Kosten im Betrieb	-3	4	-12
Korrekturaufwand	-4	4	-16
		Total	-7

5.5.2 Plattform & Infrastruktur

Als Programmierplattform haben wir .NET gewählt. Diese Entscheidung steht auch im Zusammenhang mit anderen Entscheidungen. Insbesondere der zur Infrastruktur. Bei der Infrastruktur haben wir uns für eine USB-Webcam an einem Windows Computer entschieden.

Python als Programmierplattform hätte sich besonders angeboten, in Kombination mit einer Tesseract Implementation zur Handschrifterkennung, da für Python viele Tesseract Bibliotheken existieren. C/C++ sind zwar der Hardware nahe, was die Webcam Ansteuerung vereinfacht, hätten bei uns allerdings viel Lern- und Programmieraufwand verursacht, da wir die Sprachen nicht gut kennen. GUI Entwicklung in C/C++ ist ebenfalls aufwendiger als in C# mit WPF. Da wir uns unabhängig von der Plattform für eine Handschrifterkennung in der Cloud entschieden haben, wären manuelle Web-Requests notwendig, da für C/C++ keine Bibliotheken zur Vision API von Google vorhanden sind. Wir haben uns gegen eine Web-App entschieden, weil nicht alle Autoren dieser Arbeit mit der Webentwicklung so vertraut waren, dass ein zeiteffizientes Programmieren möglich gewesen wäre. Eine zusätzliche, aber lösbare, Herausforderung bei der Web-App wäre die Persistenz von Daten. Nur wegen der Datenpersistenz eine Client-Server Architektur zu machen, würde viel Aufwand bedeuten. Eine weitere Idee wäre es, den Browser-Storage mit einer Import- und Exportfunktion zu verwenden.

Anstatt eine USB-Webcam zu verwenden, könnten wir auch eine Handykamera benutzen. Dann muss aber das Bild vom Handy an den PC übermittelt werden, was zusätzlichen Aufwand bedeutet. Die gesamte Applikation auf dem Handy auszuführen kommt für uns nicht infrage, weil Benutzerinteraktionen während dem Korrekturvorgang notwendig sind. Wenn das Handy so aufgestellt ist, dass die Kamera zum Korrigieren verwendet werden kann, ist der Bildschirm nicht mehr gut zugänglich für den Benutzer.

Bevor wir uns für C# entschieden haben, haben wir kontrolliert, dass es für alle von uns zu lösenden Aufgaben eine geeignete Bibliothek gibt und die Bibliothek unsere Erwartungen erfüllt oder der Programmieraufwand, um die Funktion selbst zu entwickeln, überschaubar bleibt.

Ausgewählte Plattform und Infrastruktur:

- USB Webcam: AForge[9]
- GUI: WPF
- PDF generieren / bearbeiten: PdfSharp[10]
- Datenbank: OR Mapper + SQLite

Folgend, die Bewertungsraster, nach denen wir uns richteten:

C# & USB Webcam	Gewichtung	Zeitaufwand	Gewichtete Bewertung
Lernaufwand	-10	2	-20
GUI	-5	2	-10
Kamera	-5	2	-10
Bildmanipulation	-5	4	-20
Handschrifterkennung	-5	2	-10
PDF bearbeiten	-2	2	-4
Persistenz	-5	1	-5
		Total	-79

[9] «vision and artificial intelligence library» *http://www.aforgenet.com/*

[10] «open source .NET library for processing PDF» *http://www.pdfsharp.net/*

C# & Android Kamera	Gewichtung	Zeitaufwand	Gewichtete Bewertung
Lernaufwand	-10	3	-30
GUI	-5	3	-15
Kamera	-5	3	-15
Bildmanipulation	-5	4	-20
Handschrifterkennung	-5	2	-10
PDF bearbeiten	-2	2	-4
Persistenz	-5	1	-5
		Total	-99

C/C++ & USB Webcam	Gewichtung	Zeitaufwand	Gewichtete Bewertung
Lernaufwand	-10	5	-50
GUI	-5	5	-25
Kamera	-5	2	-10
Bildmanipulation	-5	2	-10
Handschrifterkennung	-5	4	-20
PDF bearbeiten	-2	3	-6
Persistenz	-5	3	-15
		Total	-136

Python & USB Webcam	Gewichtung	Zeitaufwand	Gewichtete Bewertung
Lernaufwand	-10	4	-40
GUI	-5	4	-20
Kamera	-5	4	-20
Bildmanipulation	-5	2	-10
Handschrifterkennung	-5	2	-10
PDF bearbeiten	-2	3	-6
Persistenz	-5	2	-10
		Total	-116

Webapp & USB Webcam	Gewichtung	Zeitaufwand	Gewichtete Bewertung
Lernaufwand	-10	4	-40
GUI	-5	1	-5
Kamera	-5	1	-5
Bildmanipulation	-5	2	-10
Handschrifterkennung	-5	2	-10
PDF bearbeiten	-2	3	-6
Persistenz	-5	3	-15
		Total	-91

5.5.3 Punkte zuordnen

Um Punkte zuzuordnen, haben wir uns dafür entschieden, auf jeder Seite Marker anzubringen. Damit können wir die Seite einem Studenten zuzuordnen und die Seitenzahl auslesen. Die Marker sind dafür eine sehr zuverlässige Lösung und wir können sie gleich noch für die Perspektivenkorrektur verwenden.

5.5.3.1 Marker

Mit speziellen Markern kann eine sehr hohe Zuverlässigkeit erreicht werden, weil die genau dafür gemacht sind. ArUco-Marker haben in unserem Test eine noch deutlich höhere Zuverlässigkeit erreicht als QR-Codes. Mehr dazu im Kapitel «*5.5.4* Marker Implementation*»*.

Der Entwicklungsaufwand ist hier erhöht, weil wir die Marker vor dem Druck der Prüfung generieren müssen. PDF-Bearbeitung brauchen wir aber auch schon bei der Lösung, die verwendet wird, um die Punktzahlen zu suchen, deshalb ist das hier kein grosser Mehraufwand.

Marker	Gewichtung	Bewertung	Gewichtete Bewertung
Entwicklungsaufwand	-3	2	-6
Zuverlässigkeit	5	5	25
		Total	19

5.5.3.2 Seitenzahl mit Texterkennung auslesen

Beim Auslesen der Seitenzahl per Texterkennung besteht das Problem, dass per Texterkennung die Seitenzahl zugeordnet werden muss. Die Seitenzahl finden und die Erkennung sind beides Vorgänge, die nicht sehr zuverlässig funktionieren.

Als Entwicklungsaufwand fällt hier nur die Texterkennung der Seitenzahl an. Das ist kein grosser Mehraufwand, weil wir sowieso schon eine Handschrifterkennung in der Applikation haben.

Seitenzahl per Texterkennung	Gewichtung	Bewertung	Gewichtete Bewertung
Entwicklungsaufwand	-3	1	-3
Zuverlässigkeit	5	3	15
		Total	**12**

5.5.4 Marker Implementation

5.5.4.1 ArUco & Qr

Wir starteten mit dem Plan auf QR-Codes zu setzen, da uns diese bereits bekannt waren und gut auf unseren Anforderungen zu passen schienen. Um verschiedene QR-Recognition-Bibliotheken und Dienste zu evaluieren, bauten wir für alle Bibliotheken einen simplen Prototypen. Uns interessierte dabei die Performance, also wie schnell ein QR-Code erkannt wird, und die Robustheit, respektive wie viele Codes trotz perspektivischer Verzerrung oder geringer Auflösung noch erkannt werden.

Die Performance ist für uns wichtig, da wir die Marker-Erkennung in Echtzeit über den Kamerastream laufen lassen möchten. Eine höhere Robustheit bringt uns den Vorteil, mit kleineren Markern arbeiten zu können, und weniger Vorgaben an das Kamera-Setup stellen zu müssen.

Die Resultate der Evaluation waren ernüchternd. Unter den 5 getesteten Bibliotheken gab es lediglich eine, die bei hochauflösenden Bildern Performance Werte von unter einer Sekunde lieferte. Auch die Erkennungsrate, also in wie vielen unserer Testbilder erfolgreich ein Code erkannt wurde, lag bei der besten Bibliothek nur bei 75 %.

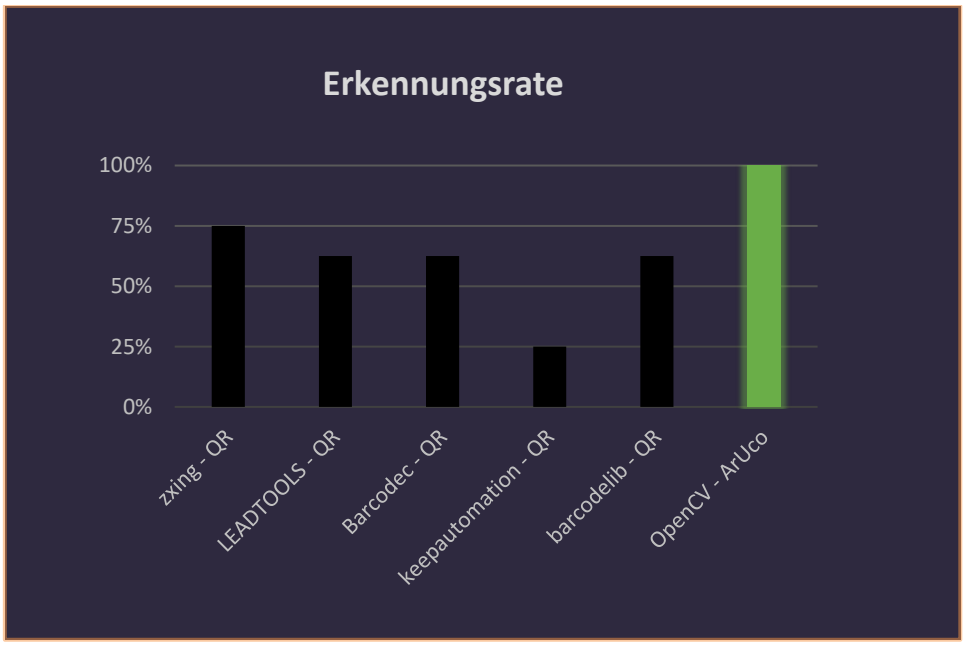
Mit diesen für uns ungenügenden Werten, mussten wir uns nach anderen Lösungen umsehen. Unsere Recherche[11] zu alternativen Methoden zum Objekt-Tracking führte uns schliesslich zu den ArUco-Codes.

Diese Codes sind optimiert für «Augmented-Reality»-Applikationen. In diesen Applikationen sind eine hohe Robustheit und Performance ausschlaggebend, da sämtliche Berechnungen in Echtzeit laufen müssen. In unserem Test mit ArUco-Testbildern (vergleichbare Bilder, wie für die QR-Code-Evaluation verwendet wurden) zeigte sich diese Robustheit mit einer 100 % Erkennungsrate, heisst es wurden alle Marker in allen Testbildern erkannt. Und auch die Performance-Resultate waren jenen der QR-Implementationen überlegen.

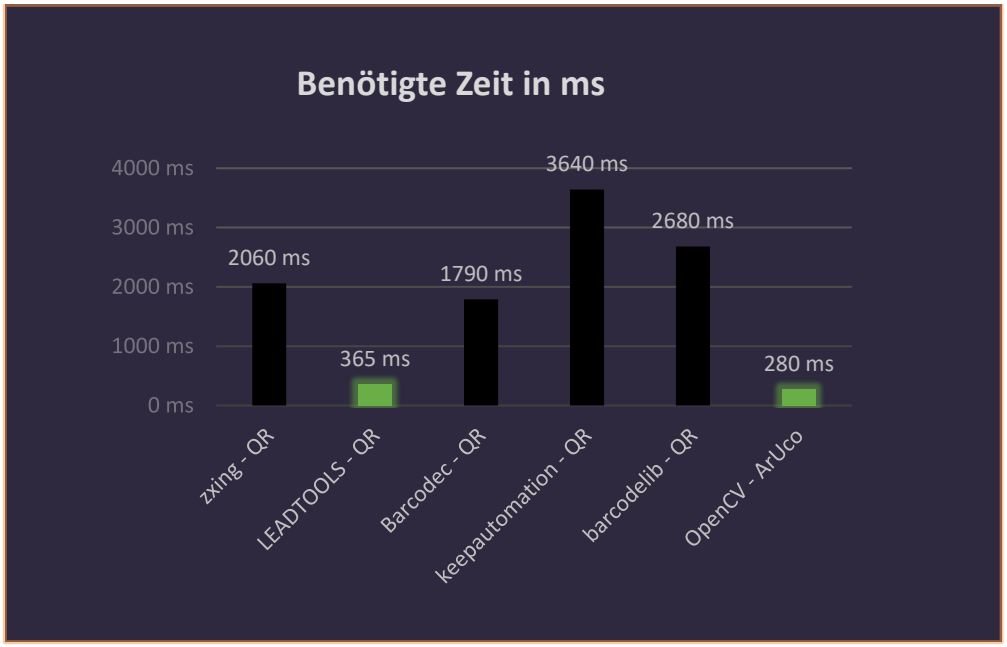
---

[11] *"Speeded up detection of squared fiducial markers", Francisco J.Romero-Ramirez, Rafael Muñoz-Salinas, Rafael Medina-Carnicer, Image and Vision Computing, vol 76, pages 38-47, year 2018*

"Generation of fiducial marker dictionaries using mixed integer linear programming",S. Garrido-Jurado, R. Muñoz Salinas, F.J. Madrid-Cuevas, R. Medina-Carnicer, Pattern Recognition:51, 481-491,2016



In wie viel Prozent aller Testbilder der Code erkannt wurde. (Grösser ist besser)



Dauer, in Millisekunden, bis der Code erkannt wurde. (Kleiner ist besser)

Die Nachteile der ArUco-Codes liegen in der Menge an Bits, die in dem Code gespeichert werden können. Die Spezifikation der Codes erlaubt es zwar, die Grösse des Markers und somit die Anzahl der Bits frei zu wählen, jedoch hat sich in der Praxis gezeigt, dass das Generieren solcher grösseren Codes, in sämtlichen von uns getesteten Bibliotheken, nicht performant implementiert ist. Das Erstellen eines ArUco-Dictionary, welches die Bit-Muster der einzelnen Marker hält, dauert teils mehrere Minuten.

Wir entschieden uns deshalb für die Verwendung von ArUco-Markern in Standartgrössen und kompensierten die geringe Datenkapazität mit der Verwendung von mehreren Markern pro Seite.

Implementiert haben wir die Generierung und Erfassung der Marker mit der Bibliothek Emgu[12]. Hierbei handelt es sich um ein C#-Wrapper für die C++ Bibliothek OpenCV[13]. Wir entschieden uns für OpenCV, weil es eines der wenigen Tools ist, das ArUco Funktionalitäten beinhaltet und wir gleichzeitig noch andere Funktionalitäten in Bezug auf Bildverarbeitung und Perspektivenberechnung nutzen können. Zudem existiert eine grosse Userbasis mit viel Support und Dokumentation.

5.5.4.2 AruCo Optimierung

Evaluationen zur Grösse der ArUco-Marker ergab, dass wir die Codes bis 13 cm x 13 cm verkleinern können und dennoch eine hohe Erkennungsrate beibehalten können.

Unsere erste Implementation sah 2 ArUco-Markern pro Seite vor. Einer beinhaltet die Seitenzahl und einer die Studenten-ID. Die Prüfungsseite wird von unserem App verkleinert (gleichmässige Skalierung) und nach rechts-unten verschoben, sodass am linken und am oberen Rand Platz frei wird. Die Codes werden nun im freien Bereich am oberen Rand der Seite platziert, da dort am meisten Platz zur Verfügung steht.

Tests dieser Implementation zeigten, dass eine kleine Ungenauigkeit in der Positionsbestimmung der Marker zu einem grösseren, störenden Fehler in der Positionsbestimmung der Seite führen konnte. Dies, weil sich der absolute Fehler vom kleinen Marker, für die viel grössere Seite multiplizierte. Die Fehler, welche wir dabei beobachteten, entstanden vor allem auch durch leicht gewölbtes Papier.

Zur Korrektur dieser Fehler implementierten wir in der nächsten Iteration einen dritten Marker in der unteren, linken Ecke der Seite. Dieser dient allein der exakteren Positionsbestimmung. Um den Marker identifizieren zu können, hält dieser einen Hashwert der ersten beiden Marker (Seitenzahl und Studenten-ID).

5.5.5 Perspektivenkorrektur

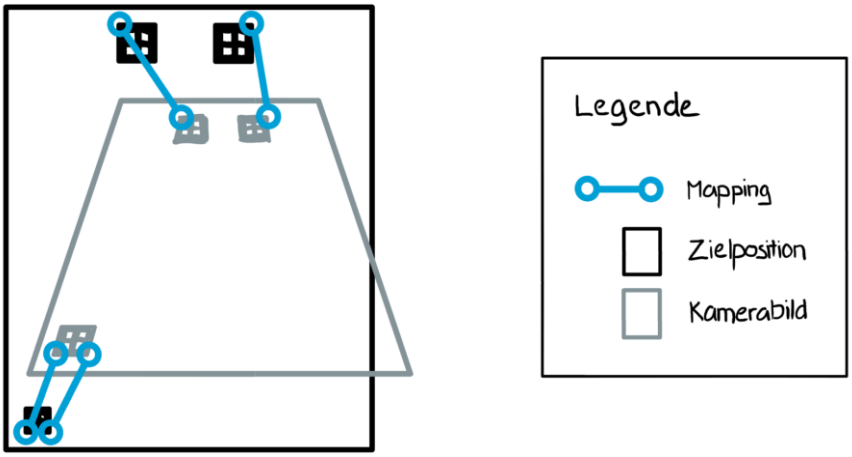
Für die Perspektivenkorrektur nutzen wir die Bildverarbeitungsbibliothek OpenCV. Diese bietet Funktionen zur Erzeugung eines Vektor-Mappings, sowie die Möglichkeit dieses Mapping auf Bilder oder Punkte anzuwenden.

Die OpenCV Funktion `FindHomography(sourcePoints, destinationPoints)` erlaubt es uns die perspektivische Transformation zwischen dem Kamerabild und dem gewünschten unverzerrten Bild zu berechnen. Von den ausgelesenen ArUco-Codes stehen uns die Positionsdaten in Form von vier Eck-Punkten zur Verfügung. Diese Punkte verwenden wir als Ursprung (`sourcePoints`) für die Transformation. Als Ziel (`destinationPoints`) definieren wir jene Positionen, an denen die Punkte währen ohne perspektivische Verzerrung.

---

[12] *http://www.emgu.com/wiki/index.php/Main_Page*

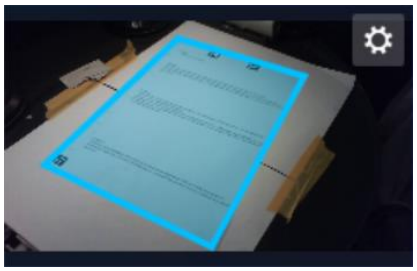
[13] *https://opencv.org/*



Schematische Skizze der Perspektivenkorrektur

Die so entstehende Transformation wird nun mit der OpenCV Funktion `WarpPerspective()` auf das Kamerabild angewendet, wodurch ein Scan-ähnliches, unverzerrtes Bild der Seite entsteht.

Um die Eckpunkte der Seite auf dem unkorrigierten Kamerabild einzeichnen zu können, wir eine zweite Transformation erstellt, invers zur Ersten, das heisst es werden dieselben Source- und Destination-Punkte wie vorher verwendet, aber jeweils vertauscht. Mit dieser umgekehrten Transformation werden nun die bekannten Eckpunkte der Seite ohne Perspektive transformiert auf die Perspektive des Kamerabildes. Die so resultierenden Punkte entsprechen exakt den vier Eckpunkten der Seite im Kamerabild:



Darstellung der Seitenumrisse im UI

5.5.6 Verhalten bei mehreren Seiten

Unsere App verarbeitet zwar immer nur eine Seite aufs mal, sie muss allerdings mit mehreren Seiten im Kamerabild umgehen können. Damit dies gelingt, haben wir zwei Mechanismen implementiert.

Der Erste betrifft die Auswahl der Seite, falls mehrere Seiten erkannt wurden. Wir haben uns entschieden, immer jene Seite zu priorisieren, welche näher in der Bildmitte ist. Dazu berechnen wir den Mittelpunkt jeder erkannten Seite, und dessen Distanz zum Mittelpunkt des Kamerabildes.

Der zweite Mechanismus dient dazu, dem User klarzumachen, welche Seite gerade verarbeitet wird. Dazu haben wir ein UI Element implementiert, das in Echtzeit das unveränderte Kamerabild zeigt und als Overlay die aktuelle Seite farblich markiert. (Siehe Grafik oben «Darstellung der Seitenumrisse im UI»)

5.5.7 User Interface Implementation

Da es für WPF-Applikationen keine offizielle Bibliothek für das Windows Metro Design gibt, mussten wir auf eine «3rd Party»-Bibliothek zurückgreifen. Zum Einsatz kommt das «MahApps.Metro»[14] Framework. Um unserem Designkonzept gerecht zu werden, musste das Styling einiger Komponenten jedoch manuell angepasst und teils komplett überarbeitet werden.

Eine alternative Lösung wäre die Verwendung von sogenannten XAML-Islands[15] gewesen. Mit dieser relativ neuen Technologie ist es möglich, UWP-XAML-Elemente auch in WPF-Applikationen zu verwenden. Da für UWP eine offizielle Bibliothek mit Windows Metro UI Elementen existiert, wären die manuellen Designanpassungen nicht mehr nötig gewesen. Wir haben uns jedoch gegen die XAML Islands entschieden, da der aktuelle Release sich noch im Alpha-Status befindet und mindestens die Windows 10 Version 1903 voraussetzt.

Perspektivenbereinigte Seite mit Punkte-Overlay

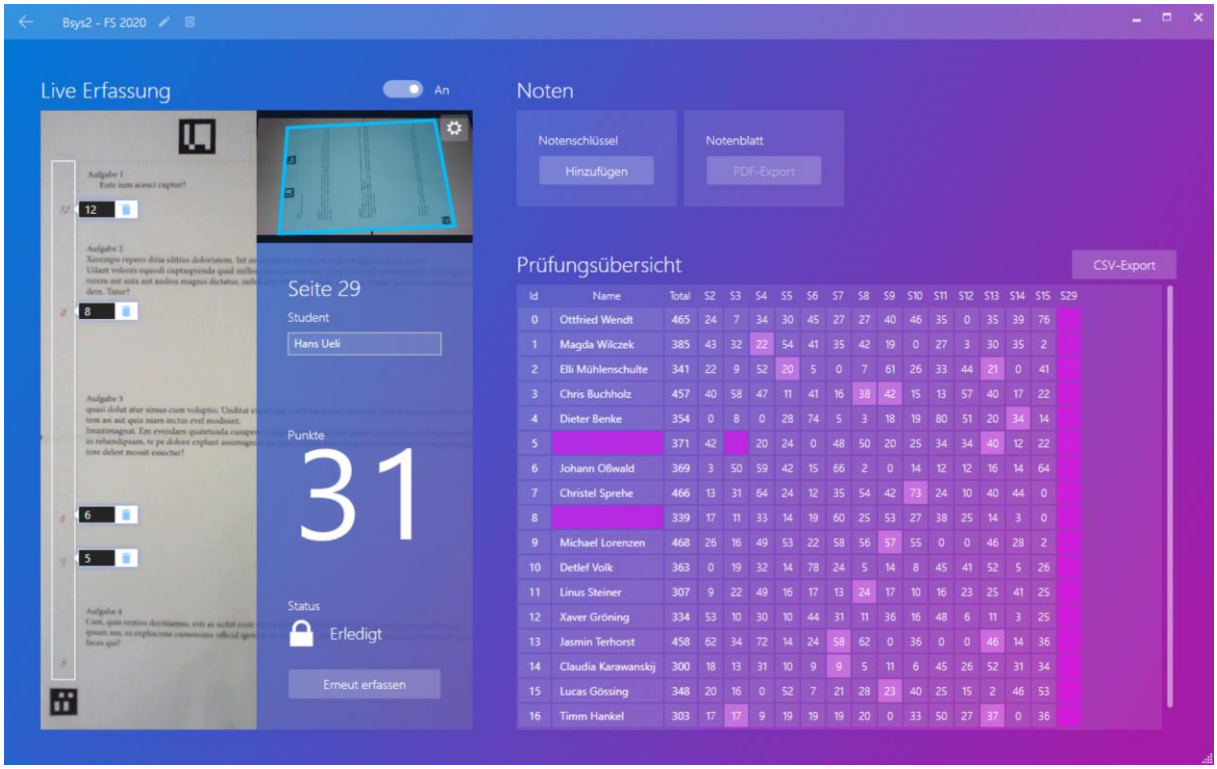
Die Darstellung der Punkte haben wir gemäss Lösungskonzept mit einem Overlay über der perspektivenbereinigten Seite implementiert. Die erkannten Punkte erscheinen jeweils als Popup neben der gefilmten Punktzahl.

---

[14] *https://mahapps.com/*

[15] *https://docs.microsoft.com/en-us/windows/apps/desktop/modernize/xaml-islands*

Homescreen (fertig implementiert)



Dashboard in einer aktiven Korrektursession (fertig implementiert)

5.6 Usability-Tests

Diese Usability-Tests wurden aufgrund der Anforderungen aus den Use-Cases definiert. Bei der Durchführung der Tests wurden die Teilnehmer manuell beobachtet. Beobachtungen wurden mit Notizen festgehalten. Die Erkenntnisse aus diesen Usability-Tests sind im Anschluss an die Testbeschreibung aufgeführt.

5.6.1 Tests

UT 1	
Name	Druck PDF einer Prüfung generieren lassen
Zustand	Das PDF einer fertigen Prüfung wird zur Verfügung gestellt.
Aufgabe	Das PDF soll für den Druck vorbereitet werden. Es fehlen noch die Marker für Studentennummer und Seitenzahl. Es sollen Kopien für 60 Studenten angefertigt werden.

UT 2	
Name	Korrektursessionen erstellen, aktivieren, ändern und löschen
Zustand	Keine Korrektursession aufgelistet
Aufgabe	Eine Korrektursession soll begonnen werden. Dazu muss sie zuerst erstellt werden. Der Name dieser Session soll nachträglich geändert werden und die Session dann gelöscht.

UT 3	
Name	Punkte auf einer Seite erkennen
Zustand	Eine Korrektursession ist vorhanden, aber nicht geöffnet. Eine korrigierte Prüfung liegt unter der Kamera.
Aufgabe	Die Korrektursession beginnen und eine automatisch erkannte Punktzahl abändern. Eine zusätzliche Punktzahl auf dieser Seite hinzufügen.

UT 4	
Name	Notenschlüssel erstellen
Zustand	Eine Korrektursession mit mehreren Prüfungsresultaten ist geöffnet.
Aufgabe	Der Notenschlüssel soll definiert werden und das Notenblatt soll exportiert werden.

5.6.2 Erkenntnisse und Massnahmen

Der erste Test, die Prüfung zum Druck vorbereiten, hat reibungslos funktioniert. Auf der Startseite ist gut ersichtlich, wo es ums PDF generieren geht und ab da ist das GUI sehr minimalistisch.

Beim zweiten Test, CRUD Operationen auf der Korrektursession, haben wir allerdings schon ein kleines Problem festgestellt. Die Schaltflächen zum Bearbeiten und Löschen waren nicht sofort ersichtlich. Die Testpersonen haben auch auf der Startseite nach diesen Schaltflächen gesucht. Es hat allerdings keine Testperson viel Zeit verloren und es ist etwas, dass ein User nur ein einziges Mal herausfinden muss, danach funktioniert es wie erwartet. Wir nehmen hier deshalb keine Änderung vor.

Beim dritten Test geht es darum, Punktzahlen zu erkennen und damit zu arbeiten. Hier konnten wir ebenfalls zwei Usability Schwachstellen erkennen. Die Testpersonen kamen erst nach einiger Zeit darauf, dass die Erkennung mit dem Ein/Aus-Schalter zuerst aktiviert werden muss. Diesen Ein/Aus-Schalter möchten wir allerdings beibehalten, damit die teuren Anfragen zur Handschrifterkennung an Google abgestellt werden können. Auch das ist etwas, dass ein User einmal kennenlernen muss, dann funktioniert es anschliessend. Wir nehmen keine Änderung an der Software vor, sondern weisen im Handbuch ausdrücklich darauf hin, dass die Handschrifterkennung eingeschaltet werden muss. Wir haben hier ebenfalls festgestellt, dass es Mühe bereitet, eine neue Punktzahl hinzuzufügen. Der User muss dazu auf den Balken mit den Punktzahlen klicken, dann wird an dieser Stelle eine Punktzahl hinzugefügt. Der Balken zeigt allerdings nicht an, dass er klickbar ist. Wir weisen auch auf diesen Umstand im Benutzerhandbuch hin. Wir weisen zusätzlich durch eine leichte Grauschattierung des Balkens beim Mouseover darauf hin, dass diese Schaltfläche klickbar ist.

Beim Test zum Notenschlüssel Export haben wir festgestellt, dass es zwei Schaltflächen gibt, die mit «Export» beschriftet sind. Das ändern wir in der Software zu «PDF-Export» und «CSV-Export».

Erkenntnisse	Massnahmen
Korrektursession bearbeiten und löschen	Keine
Handschrifterkennung aktivieren	Benutzerhandbuch
Nicht erkannte Punktzahl hinzufügen	Benutzerhandbuch und Schaltfläche per Mouseover Effekt als klickbar markieren
Zwei ‘Export’ Schaltflächen	Unterscheidung durch «PDF-Export» und «CSV-Export»

5.7 Ergebnisdiskussion

5.7.1 Eingetretene Risiken

5.7.1.1 Verfügbare Webcams

Wegen COVID-19 hatten wir Mühe an eine Webcam zu kommen. Der Liefertermin für eine gekaufte Webcam wurde mehrmals verschoben und sie traf bis zur Abgabe der Arbeit nicht ein. Die IT-Abteilung der HSR konnte uns dann allerdings eine Kamera zur Verfügung stellen, jedoch hat diese lediglich eine Auflösung von 720p. Für unsere Applikation ist eine höhere Auflösung zwar keine harte Vorgabe, doch steigt mit der Auflösung auch die Zuverlässigkeit der Punkteerkennung. Des Weiteren kann von der Kamera ein grösserer Bereich abgedeckt werden, was die Verwendung der App erleichtert.

5.7.1.2 QR Erkennung

Die QR-Codes wurden im Kamerabild oft nicht gefunden oder konnten nicht gelesen werden. Wir haben deshalb nach einer anderen Lösung gesucht. ArUco-Codes funktionieren ähnlich wie QR-Codes, haben allerdings eine viel kleinere Datendichte. Mit ArUco-Codes hat es dann um einiges besser funktioniert.

5.7.2 Weitere Probleme

5.7.2.1 Handschrifterkennung

Bei der Handschrifterkennung mit Cloud-Diensten hatten wir das Problem, dass einzelne Ziffern nicht erkannt werden. Eine einzelne Ziffer in einem Bildausschnitt, der nur so gross ist wie die Ziffer, wird nicht erkannt. Auch eine einzelne Ziffer auf einem leeren, grossen Bild wird nicht erkannt.

Wir hatten deshalb überlegt von Google Vision AI auf Azure Cognitive Services zu wechseln. Nach einem kurzen Test war aber klar, dass wir bei Azure genau dasselbe Problem hätten.

Wir möchten nicht die ganze Prüfungsseite in die Cloud schicken, weil sonst die Texterkennung sehr viel länger dauern würde und wir aus den vielen erkannten Textausschnitten die korrekte Punktzahl heraussuchen müssten.

Das Problem haben wir so gelöst, dass wir die Spalte mit den Punktzahlen zwei Mal direkt nebeneinander kopieren. So sind es nie einzelne Ziffern, die erkannt werden müssen, sondern immer mindestens eine zweistellige Zahl. Die Antworten halbieren wir in der Mitte und erhalten so unsere Punktzahl.

5.7.2.2 Performance beim PDF generieren

Beim PDF-Generieren hatten wir Performance Probleme, weil dabei sehr viel RAM alloziert wurde. Wir konnten das Problem schnell auf die ArUco-Codes zurückführen, die wir beim PDF generieren, als Bilder einfügen. Bei grossen Dokumenten wurden über 20 GB RAM benutzt.

Wir haben dann kontrolliert, dass wir sicher alle Disposable-Objekte korrekt schliessen, weil es auf den ersten Blick nach einem Memory-Leak aussieht. Dabei haben wir einige Stellen gefunden, an denen wir Verbesserungen vornehmen mussten, eine relevante Reduktion im RAM-Verbrauch hat das allerdings nicht gebracht.

Dann haben wir sichergestellt, dass für jede Zahl nur eine einzige Instanz des ArUco-Codes besteht. Z.B. der ArUco-Code für die Zahl 5 kann auf jeder Prüfung für die Seitenzahl verwendet werden und auf der Prüfung von Student 5 als Studentennummer. Diese Verbesserung war deutlich kleiner als erhofft, weil unsere PDF-Bibliothek für jede Seite alle Bilder neu speichert. Auf Seite 5 von Student 5 wurde das Bild jetzt zwar nur noch 1-mal gespeichert, auf jeder anderen Seite von Student 5 gab es allerdings keine Verbesserung.

Wir haben schlussendlich herausgefunden, dass es eine Designentscheidung der PDF-Bibliothek ist, das gesamte PDF im RAM zu halten, bis es gespeichert wird. Bilder sind dabei unkomprimiert im RAM. Deshalb kann der RAM-Verbrauch hier deutlich grösser sein als das Bild auf der Festplatte.

Glücklicherweise brauchen ArUco Codes nur eine Auflösung von 7x7 Pixel. Als wir die Auflösung der Bilder reduziert haben, hat sich auch der RAM-Verbrauch normalisiert.

5.8 Zusammenfassung und Ausblick

5.8.1 Automatische Korrekturen

Weil die Prüfung sowieso die ganze Zeit unter der Kamera liegt, könnten einfache Korrekturen auch gut automatisch durchgeführt werden. Das betrifft hauptsächlich Aufgabentypen, die sich besonders für automatische Korrekturen eignen. Insbesondere Aufgaben, bei denen die richtige Antwort nur angekreuzt werden muss.

Dazu könnte ein Einsatz von Azure Form Recognizer geprüft werden. Dieser Dienst befindet sich zum Zeitpunkt unserer Arbeit allerdings noch in Preview. Deshalb haben wir diesen Service für unsere Arbeit nicht geprüft.

5.8.2 Texterkennung der Antworten

Teilweise sind die Antworten der Studenten schwierig zu entziffern für den Dozenten. Per Handschrifterkennung könnten die Antworten auf der Prüfung gelesen und dem Dozenten auf dem Bildschirm angezeigt werden.

Dieses Feature setzt allerdings eine offline Handschrifterkennung voraus, die besser ist als Menschen.

5.8.3 Prüfung drucken

Der Arbeitsfluss beim Erstellen der Prüfung könnte weiter vereinfacht werden, wenn nicht ein PDF generiert wird, sondern die Prüfung direkt gedruckt wird. Das Aufteilen der Dokumente nach Student würde sich so erübrigen.

5.9 Quellen / Literaturverzeichnis

Bentham Collection. (kein Datum). Von http://transcriptorium.eu/datasets/bentham-collection/ abgerufen

Google. (14. 06 2020). *Google Vision AI Pricing*. Von Google Cloud: https://cloud.google.com/vision/pricing abgerufen

IAM Handwriting Database. (kein Datum). Von http://www.fki.inf.unibe.ch/databases/iam-handwriting-database abgerufen

Microsoft. (14. 06 2020). *Cognitive Services Pricing*. Von Microsoft Azure: https://azure.microsoft.com/en-us/pricing/details/cognitive-services/computer-vision/ abgerufen

Saint Gall Database. (kein Datum). Von http://www.fki.inf.unibe.ch/databases/iam-historical-document-database/saint-gall-database abgerufen

SooK, K. (03 2016). *Training a Computer to Recognize Your Handwriting*. Von KDnuggets: https://www.kdnuggets.com/2016/03/training-computer-recognize-handwriting.html abgerufen

Washington Database. (kein Datum). Von http://www.fki.inf.unibe.ch/databases/iam-historical-document-database/washington-database abgerufen

"Speeded up detection of squared fiducial markers", Francisco J.Romero-Ramirez, Rafael Muñoz-Salinas, Rafael Medina-Carnicer, Image and Vision Computing, vol 76, Jahr 2018

"Generation of fiducial marker dictionaries using mixed integer linear programming",S. Garrido-Jurado, R. Muñoz Salinas, F.J. Madrid-Cuevas, R. Medina-Carnicer, Pattern Recognition:51, 481-491, Jahr 2016

6. Anhänge

6.1 Software Engineering

Anhang A: User Manual

User Manual

1. INSTALLATIONSANLEITUNG 2
1.1 VORAUSSETZUNGEN 2
1.2 VORGEHEN 2
1.2.1 Cloud Vision API Key lösen *2*
2. BENUTZERHANDBUCH 3
2.1 PRÜFUNG VORBEREITEN 3
2.2 PRÜFUNG KORRIGIEREN 4
2.3 KORREKTURSESSION FORTSETZEN 5
2.4 KORREKTURSESSION ABSCHLIESSEN 5

1. Installationsanleitung

1.1 Voraussetzungen

- Windows 10 2004
- x64
- Internetverbindung
- Webcam
- .Net Core 3.1 Runtime
- .Net Framework 4.8 Runtime (vorinstalliert mit Windows)

1.2 Vorgehen

1. Cloud Vision API Key lösen (Kapitel 1.2.1)
2. Installer.exe ausführen und einen Installationsort wählen
3. DigitaleKorrekturunterstutzung.exe starten

1.2.1 Cloud Vision API Key lösen

Für Zugriff auf die Cloud Vision API ist ein Zugriffskey notwendig. Google beschreibt unter *https://cloud.google.com/vision/docs/before-you-begin* wie dieser Key installiert wird.

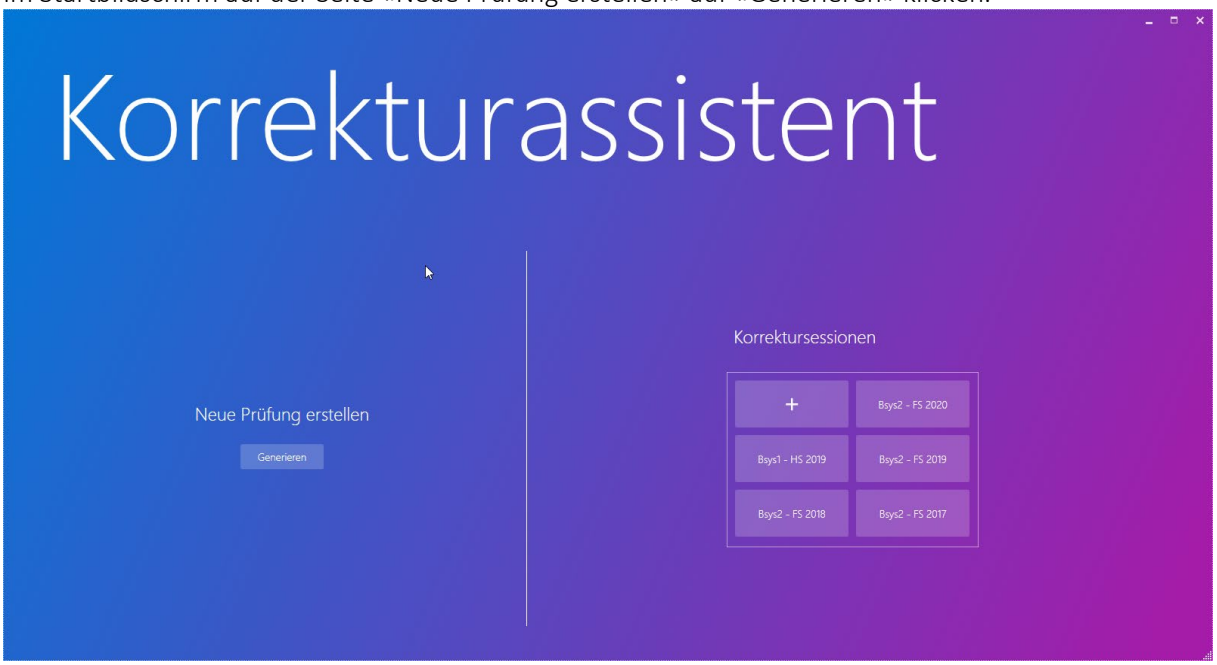
1. Ein Cloud Projekt auswählen oder erstellen auf *https://console.cloud.google.com/projectselector2/home/dashboard*
2. Billing aktivieren (oben links im Navigationsmenü ‘Billing’ auswählen)
3. Vision API aktivieren: Im Cloud Projekt Karte APIs > Go to APIs overview > oben: ENABLE APIS AND SERVICES > Cloud Vision API > Enable
4. Authentifizierung einrichten: *https://console.cloud.google.com/apis/credentials/serviceaccountkey* (New service account | Role: Project > Owner | JSON)
5. Der Schlüssel wird heruntergeladen. Die Datei ein einem sicheren Ort speichern.
6. Pfad zur Datei in Umgebungsvariable «GOOGLE_APPLICATION_CREDENTIALS» speichern

2. Benutzerhandbuch

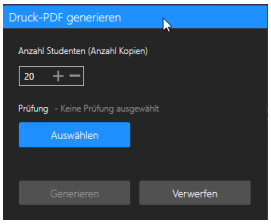
2.1 Prüfung vorbereiten

Vor dem drucken der Prüfung müssen die Marker zur Identifikation des Studenten auf der Prüfung angebracht werden.

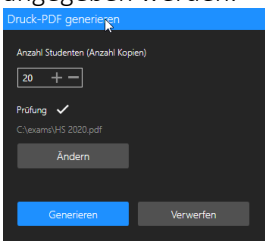
4. Im Startbildschirm auf der Seite «Neue Prüfung erstellen» auf «Generieren» klicken.



5. Im Fenster «Druck-PDF generieren» auswählen, wie viele Kopien der Prüfung benötigt werden.



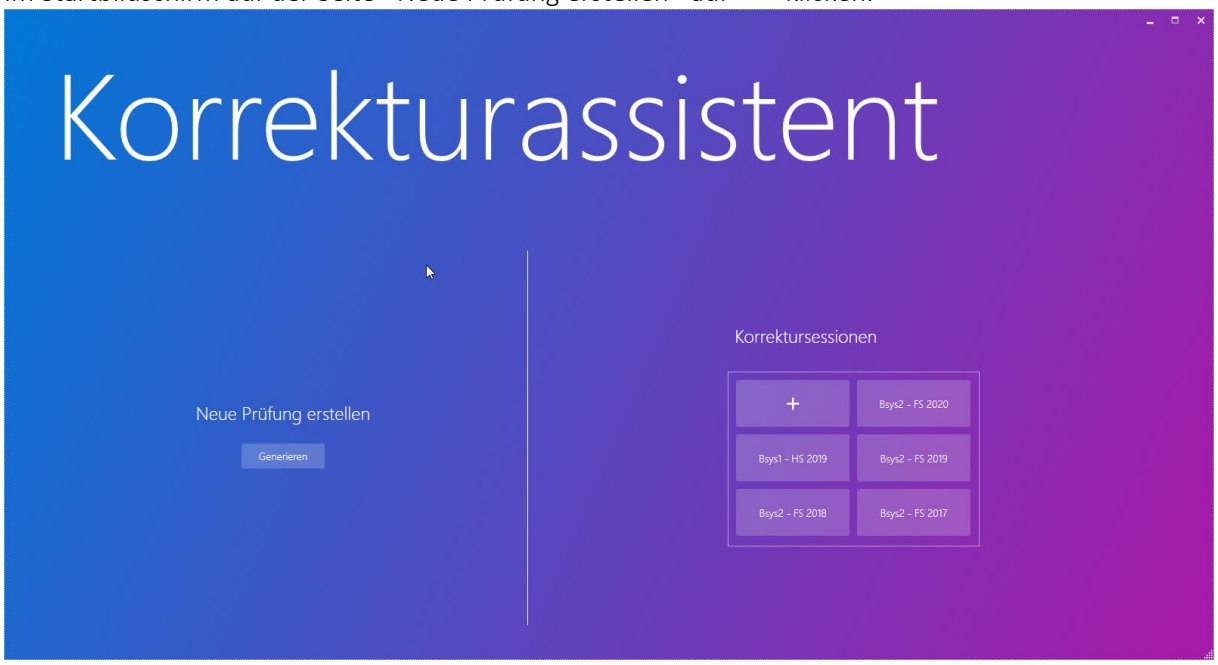
6. Das ursprüngliche PDF der Prüfung (ohne Marker) muss mit einem Klick auf «Auswählen» angegeben werden.



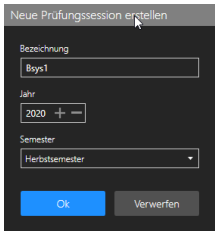
7. Mit einem Klick auf «Generieren» den Speicherort der überarbeiteten Prüfung wählen. Dieses Dokument ist nun bereit zum Druck.

2.2 Prüfung korrigieren

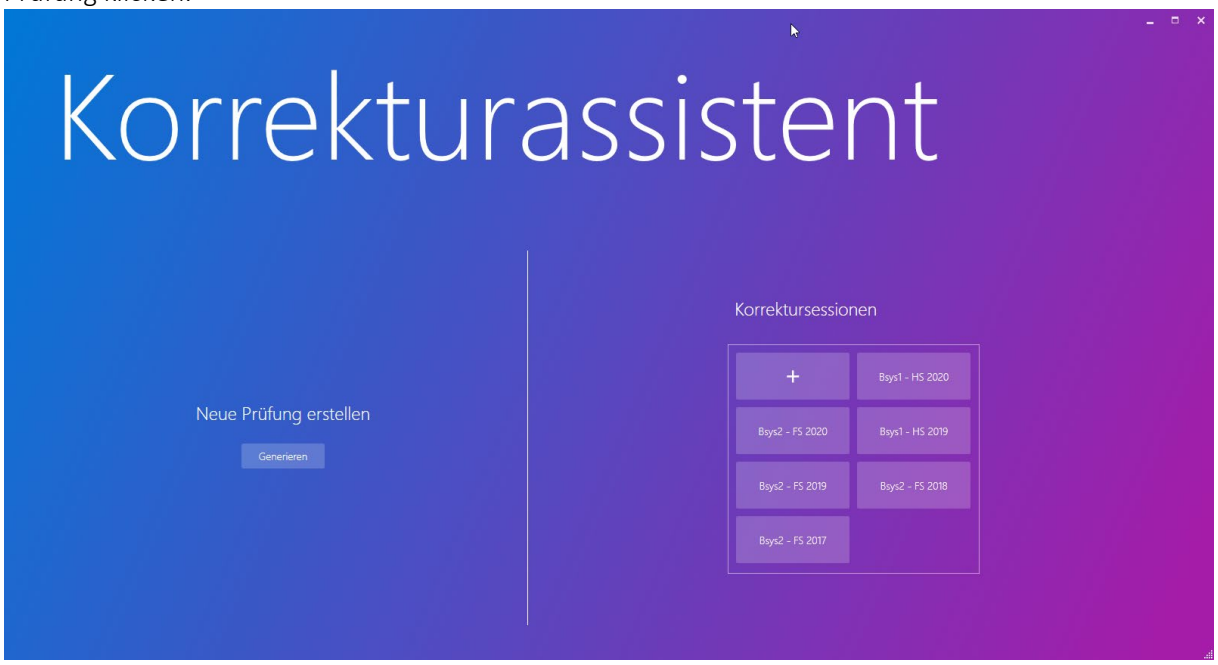
1. Im Startbildschirm auf der Seite «Neue Prüfung erstellen» auf «+» klicken.



2. Im Fenster «Neue Prüfungssession erstellen» den Namen und Zeitpunkt des Faches angeben.

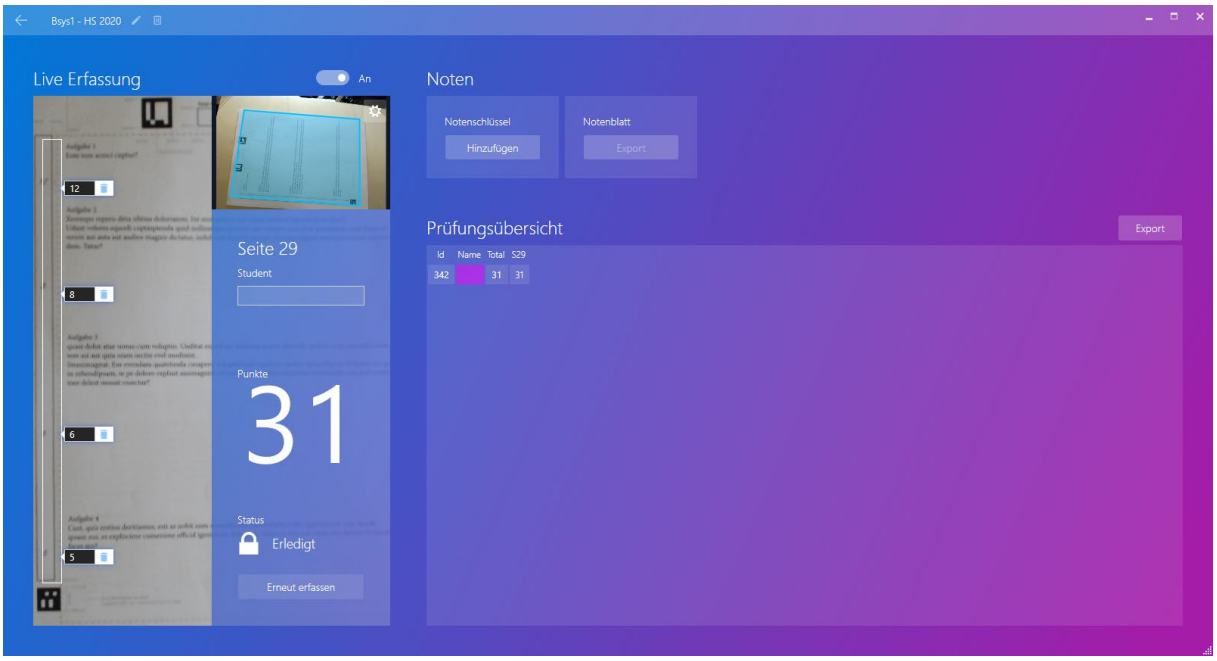


3. Im Startbildschirm auf der Seite «Neue Prüfung erstellen» auf den Namen der neu erstellten Prüfung klicken.



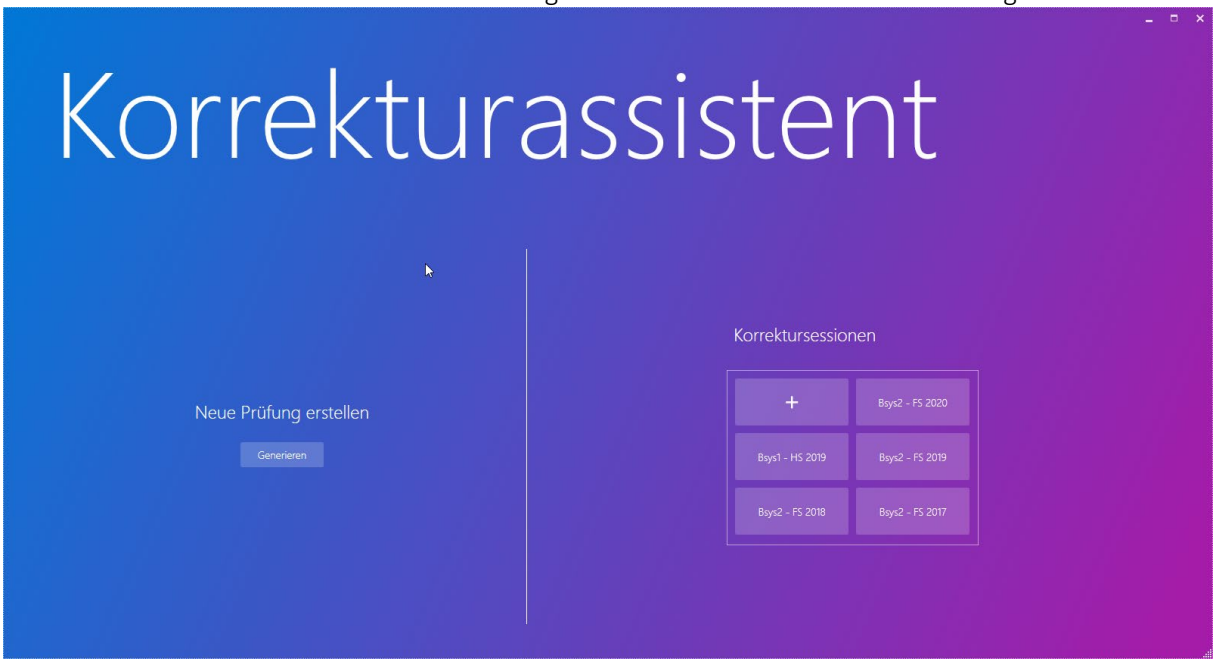
4. Wenn Live Erfassung auf «An» gestellt wird, werden auf dem Kamerabild die Marker und Punktzahlen analysiert. Erkannte Punktzahlen werden rechts in der Prüfungsübersicht dargestellt.

Wenn die Punktzahl einer Aufgabe nicht erkannt wurde, kann links im Balken auf die Zahl geklickt werden. Dann geht ein Feld auf, in dem die Punktzahl manuell eingetragen werden kann.



2.3 Korrektursession fortsetzen

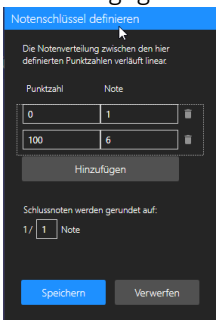
1. Im Startbildschirm auf der Seite «Neue Prüfung erstellen» auf den Namen der Prüfung klicken.



2.4 Korrektursession abschliessen

1. Mit einem Klick auf «Hinzufügen» im Feld «Notenschlüssel» kann ein linearer Notenschlüssel für den automatischen Notenexport hinzugefügt werden. Die Punktzahl für mehrere Noten kann angegeben werden. Dazwischen werden die Punktzahlen linear verteilt und auf die

unten angegebene Genauigkeit gerundet.



2. Mit der Schaltfläche «Export» wird die Übersicht aller Noten als PDF exportiert.