Julius-Maximilians-Universität Würzburg
Philosophische Fakultät
Institut für deutsche Philologie
Lehrstuhl für Computerphilologie und neuere deutsche Literaturgeschichte
Modul: Projekt in den Digital Humanities
Betreuer: Dr. Holger Essler
Sommersemester 2017

Entwicklung einer Android-Applikation zur automatisierten Kombination von Zeilensegmentierungen und OCR-Texterkennungen bei Kupferstichen zu herkulanischen Papyri

22.05.2017

Markus Bald (10. Semester)
HF: MA Digital Humanities (120 ECTS)
Karlstadter Straße 89
97737 Gemünden-Wernfeld
Tel.: 09351 / 4057
E-Mail: markusbald92@gmail.com

Inhaltsverzeichnis

1. Konzept zur Realisierung

Die Integration mobiler Anwendungen in Unternehmen oder Institutionen bietet ein hohes Nutzenpotential. Durch die flexible Nutzbarkeit können Informationen jederzeit abgerufen, erstellt oder bearbeitet, Prozesse optimiert und somit auch die Produktivität gesteigert werden. Dementsprechend nimmt die Anzahl mobiler Applikationen stetig zu. Doch auch das Internet spielt bei diesem Trend eine wichtige Rolle, da viele Anwendungen mit Servern kommunizieren, um Daten zu senden oder zu erhalten. Allerdings müssen bei der Entwicklung auch einige Aspekte berücksichtigt werden. Ein wichtiges Beispiel ist die geringere Größe der Displays. Zu bedenken ist dies beispielsweise bei Bildern mit Text, dessen Lesbarkeit bei einer geringen Bildschirmbreite stark beeinträchtigt sein kann. Die Abbildungen können zwar skaliert werden, jedoch muss der Nutzer aufgrund des „Clippings“ immer wieder umständlich den sichtbaren Bereich verschieben. Eine Lösung, die den kompletten Text innerhalb des „Viewports“ gut leserlich sichtbar darstellt, ist eine Segmentierung der einzelnen Textzeilen.

Hierauf zielt das im Folgenden dokumentierte Projekt ab. Dabei sollte eine Android-Applikation entwickelt werden, welche Zeilen auf digitalen Abbildungen von Kupferstichen zu herkulanischen Papyri segmentiert und jeweils mit dessen entsprechendem OCR-Text kombiniert. Dies soll direkte Vergleiche zwischen den Bildtextzeilen und den Ergebnissen aus deren Texterkennung, und Korrekturen, ermöglichen. Die Google-Plattform „Android“ ist das Betriebssystem, mit dem die meisten mobilen Geräte ausgestattet sind. Die über 2200 Bilder aus dem grundlegenden Werk „Herculanensium Voluminum quae supersunt collectio altera“ enthalten im Hinblick auf die Segmentierung jedoch einige Schwierigkeiten. Die in Herculaneum gefundenen Papyri enthielten Löcher, die bei den in Graustufen dargestellten Abbildungen ebenso wie eine weitläufige Umrandung stets durch Schraffuren dargestellt werden. Zudem wurde Text, der vom Kupferstecher nicht erkannt werden konnte, durch kleine Punkte gekennzeichnet. Insbesondere die Schraffuren können die Zeilenerkennung wesentlich beeinträchtigen, da diese „Geräusche“ sind, die für die Anwendung farblich nicht von Schriftzeichen unterscheidbar sind. Optimal für die Erkennung sind Schwarz-Weiß- oder Graustufen-Bilder mit hohem Kontrast und geraden Zeilen, da die Segmentierung in vertikalem Verlauf jeweils pro Pixelzeile die Anzahl dunkler Bildpunkte misst und daraus die Zeilen und deren Zwischenräume bestimmt.

Aufgrund der „Geräusche“ ist daher zunächst eine Form von (manuellem) Präprozessieren notwendig, um die Zeilen segmentieren zu können. Da die Schriftzeichen mit den Schraffuren direkt verknüpft sein können, ist eine automatische Trennung von Text und Geräuschen nicht in jedem Fall möglich. Ein Ansatz dazu wurde in „Anagnosis“, einem Teilprojekt von Kallimachos, entwickelt. Kallimachos führt Geisteswissenschaftler, Informatiker und Bibliothekare zu einem regional basierten Digital Humanities-Zentrum mit besonderem Funktionsprofil zusammen, das institutionell an der Universitätsbibliothek Würzburg angesiedelt ist.[1] Ziel des Projekts Anagnosis ist die automatisierte Verknüpfung zwischen Transkriptionen von Papyri und den Schriftzeichen der dazugehörigen Bilddatei.[2] Dabei wird vom Nutzer eine „Normbox“ aufgezogen, in der nach „Clustern“ von dunklen Pixeln gesucht wird, die in etwa die Größe von Schriftzeichen besitzen. Zwar kann mit diesem Verfahren nicht jeder Buchstabe erkannt werden, dennoch könnten bereits über einzelne Schriftzeichen Zeilen gefiltert werden. Der Code war zum Zeitpunkt des Projekts noch nicht freigegeben, das dokumentierte Vorgehen beruht jedoch auf einer OCR-Segmentierung, die benötigt wird, um Bildtexte zu erkennen.

Dieses Projekt greift daher ebenfalls auf eine OCR-Software zur Segmentierung zurück. Die Open-Source-Bibliothek „Tesseract“ lässt sich auch in Android einbinden. Der Ansatz in „Anagnosis“ zur Nutzerauswahl mit „Normboxen“ schließt jedoch weiterhin „Geräusche“ ein, weshalb hier zuvor manuell ausgeschnittene Bilder nach Zeilen segmentiert werden sollen. Die Punkte werden anschließend weitgehend automatisch entfernt. Dabei kommt die Verwendung „OpenCV“, einer „Open Source Computer Vision Library“ zum Tragen. Die Bibliothek enthält über 2500 optimierte Algorithmen zur Erkennung von Objekten sowie zum maschinellen Lernen.[3] Darunter befinden sich auch zahlreiche Funktionen zum Prozessieren von Bildern. Auch auf dieser Basis lässt sich eine Zeilensegmentierung implementieren, wesentlich einfacher und präziser ist jedoch die Durchführung einer bereits optimierten OCR-Segmentierung. Allerdings bietet „OpenCV“ auch die Möglichkeit, Geräusche zu reduzieren. Einige der Funktionen erzielen auf die Punkte angewandt optimale Ergebnisse. Die Bibliothek lässt sich ebenfalls in Android einbinden, jedoch ist dies allein zur Geräuschreduktion nicht sinnvoll und würde unnötig Gerätespeicherplatz benötigen. Zum Zuschneiden der Bilder verwendet wird Photoshop verwendet.

[1] http://kallimachos.de/kallimachos/index.php/Projektbeschreibung. (22.05.2017)
[2] http://kallimachos.de/kallimachos/index.php/Anagnosis:Main. (22.05.2017)
[3] http://opencv.org/about.html. (22.05.2017)

2. Konkrete Umsetzung

Die manuell zugeschnittenen Bilder bilden die Grundlage für die Umsetzung. Zahlreiche Bilder wurden bereits im Rahmen von Seminaren bearbeitet und sollen die über 600 Bilder, die in diesem Projekt ausgeschnitten wurden, künftig noch ergänzen. Ein Grund für die manuelle Bearbeitung liegt auch in der Unterschiedlichkeit der Bilder, zum Beispiel variieren die Schriftstärken, aber auch die Schraffuren und Platzhalter-Punkte sind nicht immer gleich deutlich zu erkennen. Der Zuschnitt ist mit dem Lasso-Werkzeug einfach möglich, je nach Anzahl der Löcher im Papyrus, dass dem Kupferstich zugrunde liegt, kann jedoch der Zeitaufwand für einen Zuschnitt deutlich variieren. Im Folgenden sollen die Bilder nun für die Segmentierung optimiert werden, bevor die Android-Applikation erstellt wird. Zur Installation der verwendeten Bibliotheken und Umgebungen existieren bereits zahlreiche Tutorials, weswegen darauf im Folgenden nicht konkret eingegangen wird.

2.1 Optimierung der zugeschnittenen Bilder mit „OpenCV“

Im OpenCV-Programm zur Optimierung der Abbildungen müssen aufgrund der Unterschiedlichkeit der Bilder je nach Kupferstich die Parameter angepasst werden. Zunächst wird mit „System.loadLibrary(Core.NATIVE_LIBRARY_NAME);“ die Bibliothek geladen. Im Folgenden wird ein Band mit etwa 200 Bildern durchlaufen, günstiger ist noch eine Aufteilung in Gruppen nach den „PHerc“-Nummern, wie auf der Download-Seite.[4] Mit „File[] files = new File(directory).listFiles();“ wird ein Array aus Dateien eines Verzeichnisses „directory“ gebildet. Dieses wird anschließend durchlaufen. Bei Dateien wird mit „Mat img = Imgcodecs.imread(directory + "/" + file.getName());“ die Datei eingelesen. Die OpenCV-Klasse „Mat“ ist folgendermaßen dokumentiert:

> „The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat).”[5]

[4] Siehe http://epikur-wuerzburg.de/digitale-ressourcen/downloads/collectio-altera/.
[5] http://docs.opencv.org/3.1.0/d3/d63/classcv_1_1Mat.html#details. (22.05.2017)

Sie bildet in OpenCV die Grundlage für alle weiteren Funktionen. Zunächst wird eine neue „Mat“-Instanz „gray“ angelegt. Mit „Imgproc.cvtColor(img, gray, Imgproc.COLOR_BGR2GRAY);“ wird „img“ in ein Graustufen-Bild konvertiert und in „gray“ gespeichert. Anschließend wird „gray“ mit der Funktion „Imgproc.threshold(gray, gray, 100, 255, Imgproc.THRESH_BINARY);“ binarisiert. Durch den höchstmöglichen Kontrast können die Schriftzeichen bei der OCR-Segmentierung nun besser erkannt werden. Die folgenden Funktionen „erode“ und „dilate“ sind morphologische Operationen, durch welche sich die Schriftstärken der Buchstaben verändern. Die Buchstaben werden insgesamt etwas dicker und dadurch auch von der noch folgenden Geräuschreduktion weniger beeinflusst. Auch dadurch wird die Erkennung noch einmal verbessert. Die Reduzierung der Geräusche wird mit der Funktion Photo.fastNlMeansDenoising(gray, gray, 85, 7, 21); durchgeführt. Ein Unschärfefilter wird mit den angegebenen Parametern berechnet. Dabei wird die Farbe eines Pixels durch den Mittelwert der Farben ähnlicher Pixel ersetzt. Mit der Funktion „Imgcodecs.imwrite(outputPath, gray);“ wird das Bild anschließend im Output-Pfad gespeichert. Insgesamt bringt der Algorithmus gute Ergebnisse hervor. Generell ist das Bild nun für die geplante OCR-Segmentierung optimiert, da die Schriftzeichen deutlicher zu erkennen sind und die Geräusche weitgehend reduziert wurden.

2.2 Aufbau der Applikation und Aktivitäten zur Auswahl

Daher kann nun die Android-Applikation erstellt wird. Als Entwicklungsumgebung wird „Android Studio“ verwendet. Als Programmiersprache kann je nach Belieben das native C++ oder das im Folgenden präferierte Java gewählt werden. Allgemein sind die Anwendungen in „Aktivitäten“ gegliedert. Diese Klassen besitzen jeweils eine Methode „onCreate“, in der mit „setContentView“ unter anderem ein Layout definiert wird. Zu den Grundelementen zählt eine Toolbar sowie ein „Floating Action Button“, ein kreisförmiger Button, der meist am unteren, rechten Rand der Aktivität zu finden ist. Eine weitere Methode ist „onCreateOptionsMenu“, die der Toolbar Menüelemente zuweist. Wie das Layout sind diese in XML definiert. Das Wurzelelement ist dabei „menu“ für ein Menü und „android.support.design.widget.CoordinatorLayout“ für ein Layout. Darin werden drei Namespaces definiert: Mit „android“ und „app“ werden Stileigenschaften von Elementen definiert, in „tools:context“ wird die zugehörige Aktivität angegeben. Jedes Layoutelement besitzt eine Länge „android:layout_width“ und Breite „android:layout_height“, sowie

eine ID, um in der Aktivität darauf zu referenzieren. Die Palette von Elementen bietet zahlreiche Möglichkeiten. Viele von Ihnen wie Text, Bilder, Tabellen, Checkboxen, Radiobuttons und Inputfelder sind bereits aus HTML oder anderen Sprachen bekannt. Für diese Anwendung werden ein Suchleisten-Widget, Bilder, eine Listen- und eine Tabellenansicht, Eingabefelder sowie Fragmente benötigt. Letztere sind Unterklassen von Aktivitäten und kommen bei einer Aufteilung in Tabs zum Einsatz. Alle Aktivitäten werden in der XML-Datei „AndroidManifest“ definiert. Hier werden auch die Zugriffsrechte verwaltet. Für diese Anwendung sind unter anderem Rechte für den Zugriff auf den externen Speicher erforderlich, da in dessen Verzeichnis die jeweilige vom Nutzer ausgewählte TEI-Kodierung zu den Kupferstichen sowie die bisherigen Änderungen gespeichert werden sollen. Da diese von „GitHub“ bzw. dem Server des Würzburger Zentrums für Epikureismusforschung heruntergeladen werden, wird auch ein Zugriff auf das Internet benötigt. Im „Android Manifest“ werden unter anderem auch die Startaktivität, der Name der App sowie das Logo definiert. Eine weitere wichtige Datei ist „build.gradle“. Mit „Gradle“ wird die App gebildet. Hier werden auch die Bibliotheken definiert, die für das Projekt kompiliert werden sollen. Darunter befinden sich unter anderem die Open-Source-OCR-Software „Tesseract“, sowie Picasso, mit der Onlinebilder geladen und im Cache gespeichert werden können.

In der Aktivität, die beim Start der Anwendung ausgeführt wird, soll der Nutzer eine „TM-Nummer“ eingeben können. Diese entsprechen den Dateinamen der TEI-Kodierungen. Jede Kodierung enthält ein Set von Bildern. Notiert sind die Links zu den Bildern, die Texte sowie weitere Metadaten. Wurde eine Nummer eingegeben, wird ein „AsyncTask“ initiiert. Diese asynchronen Threads sind häufig notwendig, damit die graphische Nutzeroberfläche nicht während einer einzelnen Ausführung eingefroren wird. Die Tasks besitzen eine Methode „doInBackground“, die im Hintergrund zunächst eine Verbindung zum GitHub-Server herstellen und die Kodierung downloaden und anschließend eine weitere zum Server des Würzburger Zentrums für Epikureismusforschung, um die XML-Datei mit den Änderungen herunterzuladen. Beide Dateien werden im externen Speicher des Geräts gesichert und können somit im Folgenden verwendet werden.

Konnten beide Dateien erfolgreich heruntergeladen werden, soll dem Nutzer nun in einer neuen Aktivität eine Liste der Kupferstiche angezeigt werden, aus denen er ein Bild zur Segmentierung auswählen kann. Dafür muss in der Aktivität zunächst die XML-Datei nach den Links zu den Abbildungen durchsucht werden. Java bietet die Möglichkeit, mit

einem „DocumentBuilder“-Objekt einen XML-Baum zu parsen. Danach wird eine Liste von „graphic“-Knoten gebildet. Beginnt der Dateiname, der im Attribut „url“ gespeichert ist, mit „VH2“, ist die Abbildung ein Kupferstichbild und wird einem LinkedHashSet hinzugefügt. Dieses unifiziert die Links, da diese mit mehreren IDs auch mehrfach notiert sein können. Für die Listenansicht wird jedoch eine Liste benötigt, daher werden die Links durch Instanziieren einer „ArrayList“ mit dem Set als Parameter in eine solche übertragen. Mit „Collections.sort“ werden die Einträge sortiert, um die Suche zu vereinfachen. Die „ListView“ in Android wird durch einen „ArrayAdapter“ erzeugt, dem neben der „ArrayList“ auch ein XML-Layout übergeben wird, wie die einzelnen Listeneinträge angezeigt werden sollen. Wird ein Element angeklickt, wird der Link über ein Intent-Objekt mit „putExtra“ an die neue Aktivität weitergegeben.

2.3 Erstellung der Hauptaktivität

Diese bildet nun die Hauptaktivität. Aufgeteilt werden soll diese in Fragmente, die durch Tabs aufgerufen werden können. Ein „OverviewFragment“ soll einen Überblick über den Kupferstich mit allen verfügbaren Metadaten und dem Originalbild geben. „SegmentedImageFragment“-Darstellungen sollen dann für jeden Abschnitt bzw. jede ID das entsprechende Teilbild segmentieren und den zuvor extrahierten OCR-Text mit den Bildtextzeilen kombinieren. Klickt der Nutzer auf den „Floating Action Button“ am unteren Rand, sollen seine Änderungen in eine XML-Datei übertragen, lokal gespeichert und dann auf den Server hochgeladen werden. Durch Zeitstempel sollen dabei zudem auch parallele Bearbeitungen ermöglicht werden, ohne dass die vorherige Änderung überschrieben wird. Zunächst wird die Verwendung der OCR-Software „Tesseract“ vorbereitet. Dazu wird eine neue Instanz der Klasse „TessBaseAPI“ erzeugt. Diese wird mit einer Sprache initialisiert. Für die Zeilensegmentierung kann jede Sprache gewählt werden. Wichtig ist lediglich, dass die Trainingsdaten zuvor auf das Gerät kopiert werden. Daher wird die Existenz der Datei mit der Funktion „checkFile“ überprüft. Ist diese nicht vorhanden, kopiert die Methode „copyFiles“ die Trainingsdaten in den externen Gerätespeicher. Anschließend werden die Texte und die Metadaten aus der XML-Kodierung extrahiert. Das Vorgehen ist hierbei ähnlich zu dem der vorherigen Aktivität, allerdings wird nun zusätzlich eine Instanz der Klasse „XPathFactory“ erzeugt, die auch das Selektieren von Knoten über XPath-Ausdrücke ermöglicht. Gesucht wird nach dem Elternelement

„custEvent“ der Grafik mit der übergebenen URL, da in dessen Attribut „corresp“ die ID gespeichert ist, auf welche die Texte und Metadaten referenzieren. Da mehrere IDs in einem Attribut durch Kommas getrennt notiert sein können, werden diese mit der Funktion „split“ separiert und in einem Array gespeichert. Die Einträge werden anschließend jeweils durchlaufen, wobei mit XPath ein „div“-Element gesucht wird, welches ein Attribut „corresp“ mit dem Wert der aktuellen ID besitzt. Wurde ein solches Element gefunden, werden die „lb“-Elemente mit einem „|“ als Inhalt versehen. Dies ist notwendig, um wiederum mit „split“ im Folgenden die Zeilen trennen zu können. Der Inhalt der „supplied“-Elemente wird nicht benötigt und daher entfernt. Dann werden die Texte zu jeder ID in einer „LinkedHashMap“ gespeichert, welche die Reihenfolge der Einfügung aufrechterhält. Bei den Metadaten lassen sich auf ähnliche Weise der Name des Kupferstechers, das Jahr bzw. der Zeitraum und die Person zur Aufrollung des Papyrus, der antike Autor und der antike Titel sowie die PHerc-Nr. und die Zuordnung der IDs zu Spalten- und Fragmentbezeichnungen extrahieren.

Anschließend wird ein Iterator über die Texte initialisiert. Die URL wird überschrieben, sodass sie nun auf das erste Teilbild auf dem Server des Würzburger Zentrums für Epikureismusforschung verweist. Mit einem „ViewPagerAdapter“ wird nun eine Fragmentansicht erzeugt. Dieser bezieht sich auf ein „ViewPager“-Element, das im XML-Layout definiert wurde. Mit dem Adapter werden die Tabs gesteuert. Das „ViewPager“-Element wird durch die Funktion „setupWithViewPager“ mit dem „Tab Layout“ verknüpft, wodurch die Fragmente über die Tabs gesteuert werden können und gleichzeitig auch mit jedem Fragment, dass dem Adapter hinzugefügt wird, ein neuer Tab erzeugt wird.

Die Fragmente werden wie Aktivitäten generiert. Sie enthalten ebenfalls eine Methode „onCreate“ und zusätzlich eine Funktion „onCreateView“. Dort wird mit „inflate“ das Inhaltslayout geladen. Jedes Fragment erhält ein leeres, lineares Layout, das anschließend im Programmcode dynamisch angepasst wird. Das Überblicksfragment soll eine Tabelle von Metadaten und das Originalbild anzeigen. Mit „getActivity“ kann eine Methode der übergeordneten Aktivität ausgeführt werden. Die „LinkedHashMap“ mit den Metadaten wird durch eine Getter-Methode übergeben. Die Tabelle wird auf ähnliche Weise erstellt wie eine HTML-Tabelle. Einem „TableLayout“ wird eine beliebige Anzahl von „TableRows“ hinzugefügt. Jedes Element, welches anschließend einer „TableRow“ zugewiesen wird, bildet eine Spalte. Mit einer For-Schleife wird die Metadaten-Map durchlaufen und für jeden Eintrag eine TableRow mit zwei Textelementen gebildet. Die erste Spalte

mit Überschriften zeigt jeweils den Key an, die zweite den Value. Anschließend wird eine „ImageView“ generiert, die das Originalbild anzeigen soll. Mit der Bibliothek Picasso wird das Bild verkleinert online geladen und in die „ImageView“ eingefügt. Durch die Verkleinerung kann das Bild schneller geladen werden. Zudem wird die Abbildung im Cache gespeichert, sodass sie bei wiederholter Anzeige nicht erneut geladen werden muss.

Im „SegmentedImageFragment“ wird zunächst das „TessBaseAPI“-Objekt aus der Aktivität übertragen. Auch wird der nächste Texteintrag mit dem Iterator abgerufen. Die URL des aktuellen Teilbilds wird ebenfalls benötigt. Eine „ArrayList“ von „EditText“-Objekten wird erzeugt. Darin werden die Inputfelder gespeichert, um die Änderungen beim Klick auf den Button abrufen zu können. Mit Picasso wird nun eine Bitmap geladen. Diese wird als Grundlage zur Anwendung von Tesseract benötigt. Anschließend wird der Index erhöht, da mehrere Teilbilder durch Endziffern unterschieden werden. Der Index, verglichen mit der Anzahl der Texte löst die Frage, ob noch weitere Bilder bzw. Fragmente folgen. Ist dies der Fall, wird der Index und die neue Bild-URL gesetzt. Konnte die Bitmap erfolgreich geladen werden, wird anschließend direkt die Segmentierung durchgeführt. Dabei wird dem „TessBaseAPI“-Objekt das Bild zugewiesen. „getStrips().getBoxRects()“ liefert ein Array von Rechtecken zurück, deren Koordinaten jeweils abgerufen werden können. Ein Screenshot eines frühen Testentwurfs der OCR-Segmentierung visualisiert die Präzision, mit der die Zeilen erkannt werden, aber auch die Problematik der „Geräusche“ durch die Schraffuren. Im unteren Teil sind die Koordinaten zu finden:

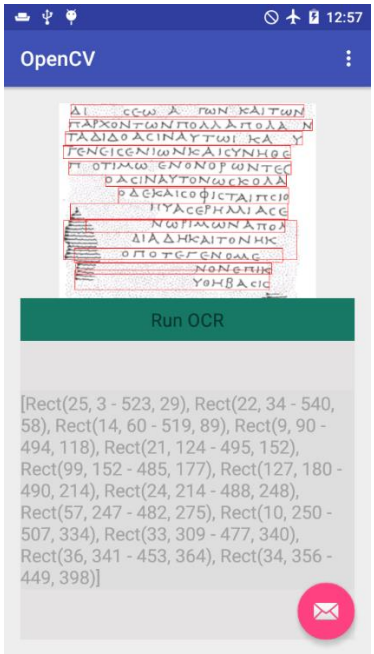

Abbildung 2.3.1: Screenshot eines frühen Testentwurfs, der die OCR-Zeilenerkennung und die Beeinflussung durch die Schraffur visualisiert. Da zu diesem Zeitpunkt einige Versuche zur Segmentierung mit OpenCV unternommen wurden, ist dies der Titel der Aktivität, die Zeilenerkennung wurde allerdings mit „Tesseract“ durchgeführt.

Aus der Bitmap wird eine Teilbitmap ausgeschnitten und als „ImageView“ hinzugefügt. Darunter wird jeweils ein leeres Eingabefeld gesetzt. Die OCR-Texte werden erst im Nachhinein eingefügt. Die „ArrayList“ mit den Eingabefeldern wird jeweils um dieses „EditText“-Objekt erweitert. Nun wird der Text durch das Trennzeichen „|“, dass zuvor für die „lb“-Elemente in der Kodierung gesetzt worden war, aufgesplittet und durchlaufen. Leere Zeilen werden übersprungen. Ansonsten wird der Text jeweils dem nächsten Eingabefeld zugewiesen. Die Zeile mit Nummer wird wiederum in einer „LinkedHashMap“ gespeichert. Nun müssen nur noch die bereits verfügbaren Änderungen hinzugefügt werden. Eine Methode „refreshLines“ fragt dazu in der entsprechenden XML-Datei mit XPath alle Zeilen mit der Text-ID ab, die das Attribut „latest=“true“ besitzen. Dieses ist aufgrund der Zeitstempel notwendig, die beim Speichern der Änderungen gesetzt werden sollen. Alle geänderten Zeilen werden durchlaufen und die Eingabefelder aktualisiert. Eine „Deep Copy“ der „ArrayList“ mit den Eingabefeldern ermöglicht die Überprüfung auf Änderungen beim Speichern.

Bei Klick auf den Button werden die Änderungen gespeichert und hochgeladen. Dabei müssen jedoch einige Fälle beachtet werden. Ziel ist eine stetig sortierte Struktur. Daher müssen auch die Knotenpositionen beachtet werden. Alle Zeilen werden durchlaufen. Ist der Zeilentext ungleich dem in der Kopie der Map, werden mit XPath die „lines“-Elemente extrahiert. Im ersten Fall wurden noch keine Änderungen hinzugefügt, demnach ist die Knotenliste mit „lines“-Elementen leer. Daher wird ein neues Element „lines“ angelegt, dass als Attribut die ID des Textes speichert. Darin wird die aktuelle geänderte Zeile mit einem neuen Zeitstempel notiert. Das Attribut „latest“ wird ebenfalls hinzugefügt. Wurden „lines“-Elemente gefunden, zum Beispiel nun bei einer zweiten geänderten Zeile, wird überprüft, ob die Zeile bereits einmal geändert wurde. In diesem Fall wird ein neuer Knoten mit einem Zeitstempel hinzugefügt und das Attribut „latest“ auf diesen gesetzt. Ansonsten wird über die Zeilen-Nummer überprüft, an welche Position ein neuer Zeilenknoten gesetzt wird. Gleiches gilt für „lines“-Elemente. Über die ID-Nummer wird geprüft, an welche Position ein neues „lines“-Element eingefügt werden muss. Die XML-Datei wird zunächst lokal gespeichert und ist somit für den Nutzer verfügbar. Wichtig ist

in allen Fällen auch, die „LinkedHashMap“ mit den Ursprungszeilen auf den aktuellen Zustand der Bearbeitung zu aktualisieren, da sonst bei mehrfacher Speicherung jeweils ein neuer Knoten für die gleiche Änderung erzeugt wird. Das Hochladen erfordert Schreibrechte auf dem Server und das Erstellen eines PHP-Skripts, dass die Datei letztlich auf den Server kopiert. Die Methode „uploadFile“ in der Aktivität stellt lediglich eine Verbindung her und führt die PHP-Datei aus. Zugleich sendet sie jedoch auch einige Header-Anweisungen. Folgende Screenshots zeigen die Aktivität:

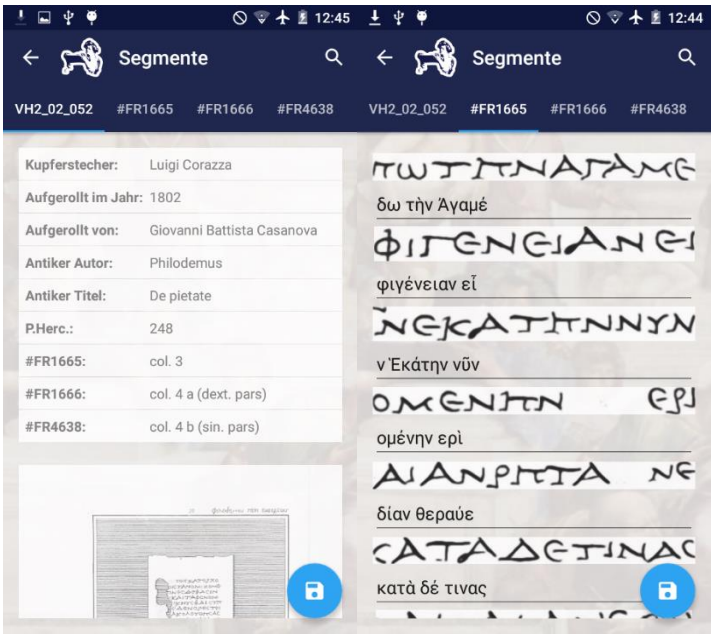

Abbildung 2.3.2: Beispiel-Screenshots der Aktivität mit Fragmenten, links ein Überblick mit Metadaten und dem Originalbild, rechts die Text-Bild-Kombination der Zeilen.

3. Weitere Möglichkeiten zur Implementierung

Mit der in dieser Arbeit vorgestellten Implementierung der App gelingt die Kombination der Bildtextzeilen- und des OCR-Texts in vielen Fällen bereits sehr gut, dennoch existieren noch einige Probleme, die im Folgenden betrachtet werden sollen.

Ein Problem ergibt sich in einigen Fällen durch die Kodierung. Einige Zeilen, die der Kupferstecher nicht erkennen konnte und erst später durch Infrarot-Technologie sichtbar gemacht werden konnten, lassen sich in der Kodierung leider nicht erkennen und ausschließen. Somit verschieben sich die Zuordnungen in der App, da die Zeile im Kupferstichbild nicht existiert. Hier wäre ein Button für jede Zeile ideal, mit dem der Nutzer ab der gewünschten Position die Zuordnung dieser und der Folgezeilen um eine Stelle nach oben verschieben könnte. Gleiches gilt für den Fall, dass die Segmentierung eine Zeile ausgelassen hat. Durch die Handschrift ist auch eine Erkennung von zwei Zeilen als eine möglich, wenn ein Schriftzeichen in die nächste Zeile hineinragt. Aber auch Kommentare in den Kupferstichen können bei der Erkennung problematisch sein.

Ein weiteres Problem ist die Verwendung einer ID für mehrere Textteile. In diesen Fällen können Texte nicht immer klar bestimmten Bildern zugeordnet werden. Ein Fragment kann dann den Text für ein anderes enthalten und umgekehrt, wodurch die Bild-Text-Kombination nicht mehr übereinstimmt. Auch hier wäre ein Button pro Fragment wichtig, der die Vertauschung der Texte mit einem anderen Fragment ermöglicht. In einem früheren Projekt habe ich außerdem bereits eine Datenbank zu Metadaten online verfügbar gemacht. Eine interessante Möglichkeit wäre, diese zu nutzen, um eine erweiterte Suche mit den Metadaten durchführen zu können. Zum Beispiel könnten die Tabelleneinträge im Überblicksfragment mit einem Klick-Event verlinkt werden, welches eine Datenbankabfrage startet. Um frühere Änderungen sichtbar zu machen, könnte jedes Eingabefeld mit einer Dropdown-Liste mit Werten vorheriger Zeitstempel verknüpft werden. Wichtig wäre auch eine Tastatur mit altgriechischen Schriftzeichen. Hierfür existieren verschiedene Tutorials.

Insgesamt ist der Anteil an optimalen Bild-Text-Kombinationen bei den über 600 im Projektrahmen verfügbar gemachten Bildern bereits sehr hoch. Auch der Nutzen an der Anwendung ist groß. Die Anwenderfreundlichkeit jedoch ist noch ausbaufähig. Einige hier dargestellte Verbesserungsvorschläge könnten die Nutzbarkeit noch deutlich steigern.

4. Literaturverzeichnis

Sekundärliteratur:

http://docs.opencv.org/3.1.0/d3/d63/classcv_1_1Mat.html#details

http://kallimachos.de/kallimachos/index.php/Anagnosis:Main

http://kallimachos.de/kallimachos/index.php/Projektbeschreibung

http://opencv.org/about.html