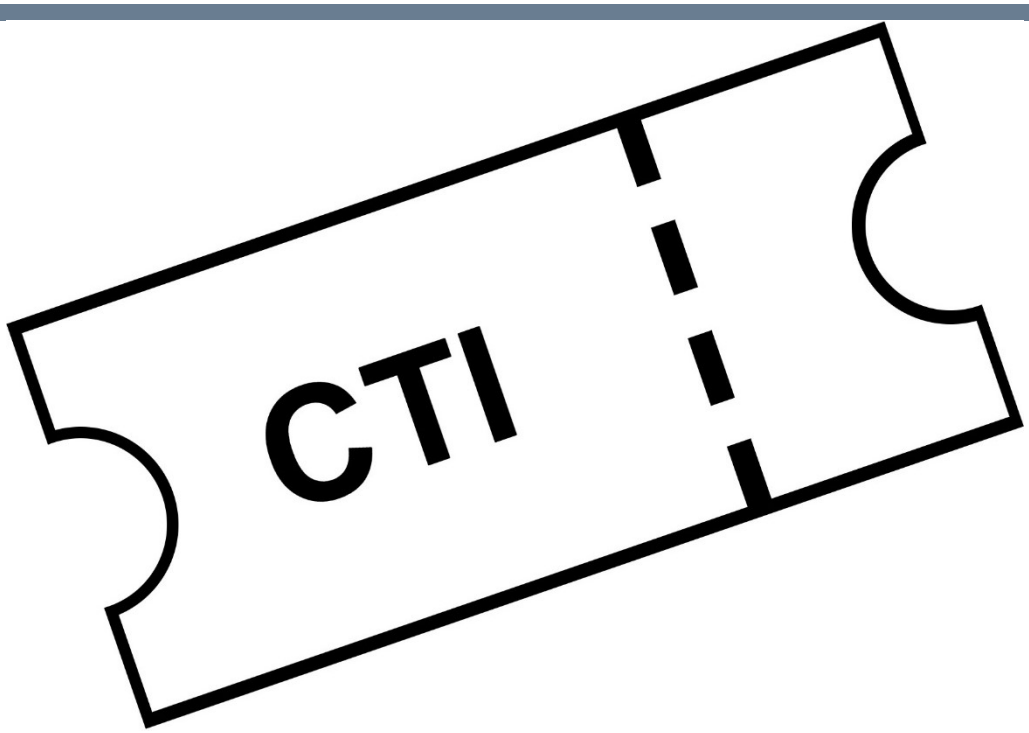

Technische Dokumentation CTI

Projektteam 10

Lukas Seglias
Luca Ritz

V. 1.0 17.01.2019

Berner Fachhochschule
Departement für Technik und Informatik

Inhaltsverzeichnis

1	Zweck des Dokuments	3
2	Projektstruktur	3
	2.1 CTI-Library	4
	2.2 JavaCTI-Library	6
3	Matching-Algorithmus	8
	3.1 Training	8
	3.2 Verarbeitung	9
	3.3 Abgleich	10
	3.4 Auswahl	11
4	Extraktion-Algorithmus	11
	4.1 Verarbeitung	11
	4.2 Normalisierung	11
	4.3 OCR	12
	4.4 Nachverarbeitung	13
5	Glossar	13
6	Abbildungsverzeichnis	14
7	Versionskontrolle	14

1 Zweck des Dokuments

Dieses Dokument beschreibt die technische Umsetzung des Projekts CTI.
Im Kapitel Projektstruktur wird der Aufbau und die Struktur des Projekts ausgeführt.
Unter Matching-Algorithmus und Extraktion-Algorithmus wird die Funktionsweise und Implementation der Hauptfunktionalität der Bibliothek beschrieben.

2 Projektstruktur

Das Gesamtprojekt besteht aus drei aufeinander aufbauenden Bibliotheken. So stellt CTI die Basisimplementation für C++-Anwendungen bereit. Des Weiteren wird die Logik durch einen Java-Adapter soweit abstrahiert, dass darauf aufbauend eine Java-Bibliothek aufgebaut werden konnte, welche von Java-Anwendungen eingesetzt werden kann.

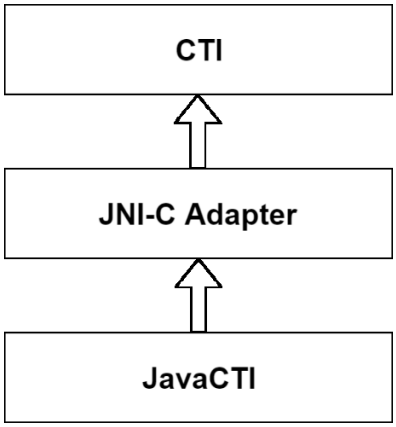

Abbildung 1: Projektaufbau und Abhängigkeiten

Das Projekt CTI besteht aus zwei Modulen, der CTI-Library sowie der Testanwendung. Ausgeliefert wird lediglich die CTI-Library. In der CTI-Library ist die öffentliche Schnittstelle (C++ header files) von der Implementation getrennt.

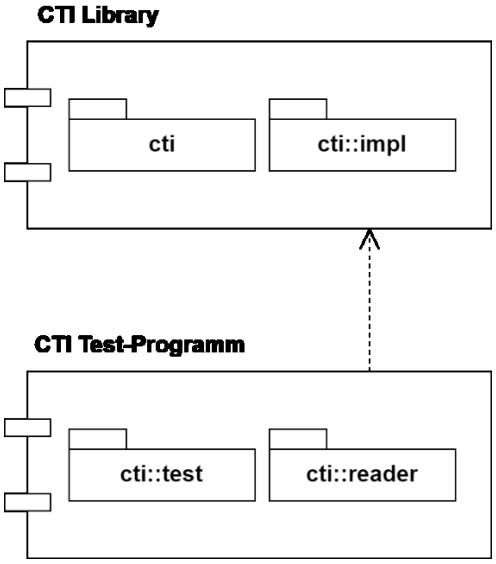

Abbildung 2: CTI-Library

2.1 CTI-Library

Nachfolgend sind die Domänenklassen aus der CTI-Library aufgeführt.

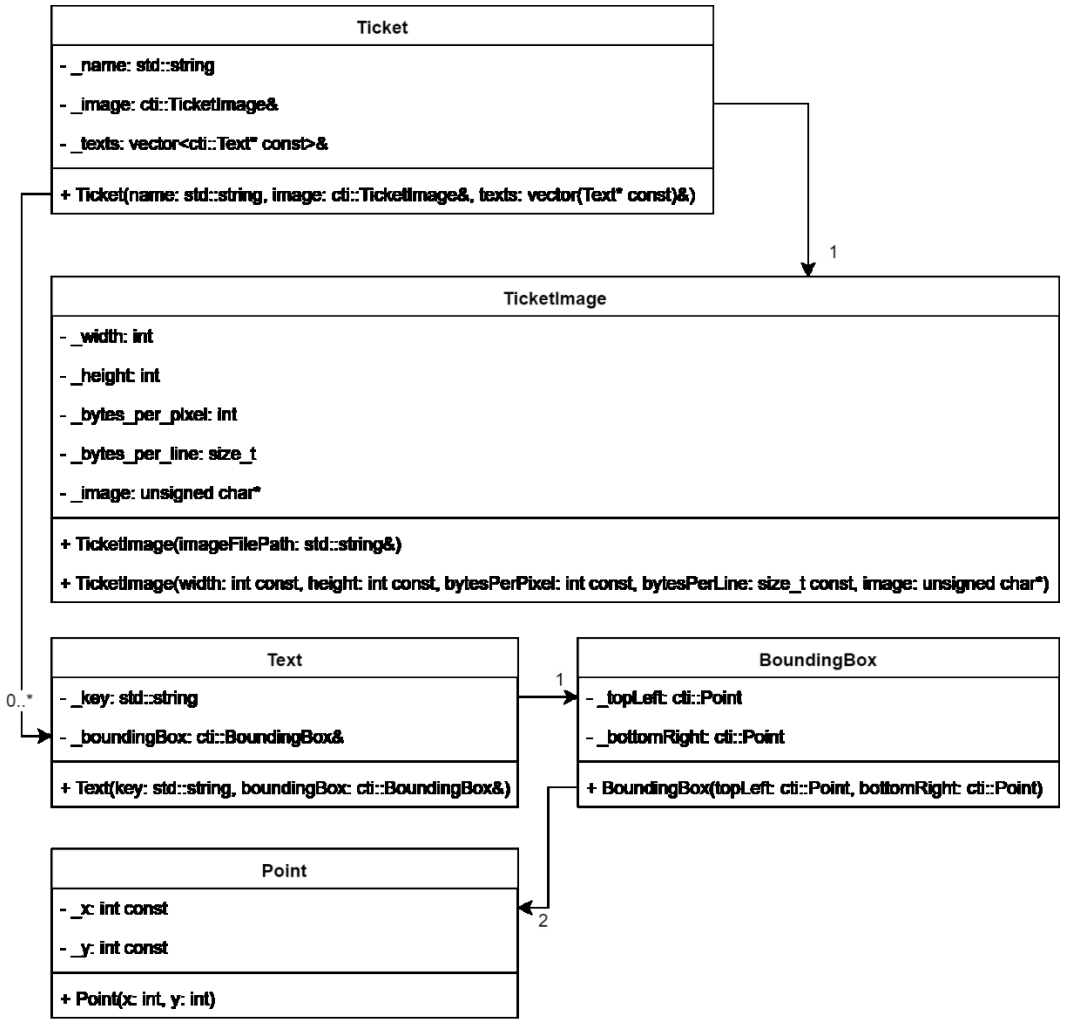

Abbildung 3: Domänenklassen CTI-Library

In Abbildung 4 sind die Klassen des öffentlichen API für die Matching-Funktionalität der CTI-Library abgebildet.

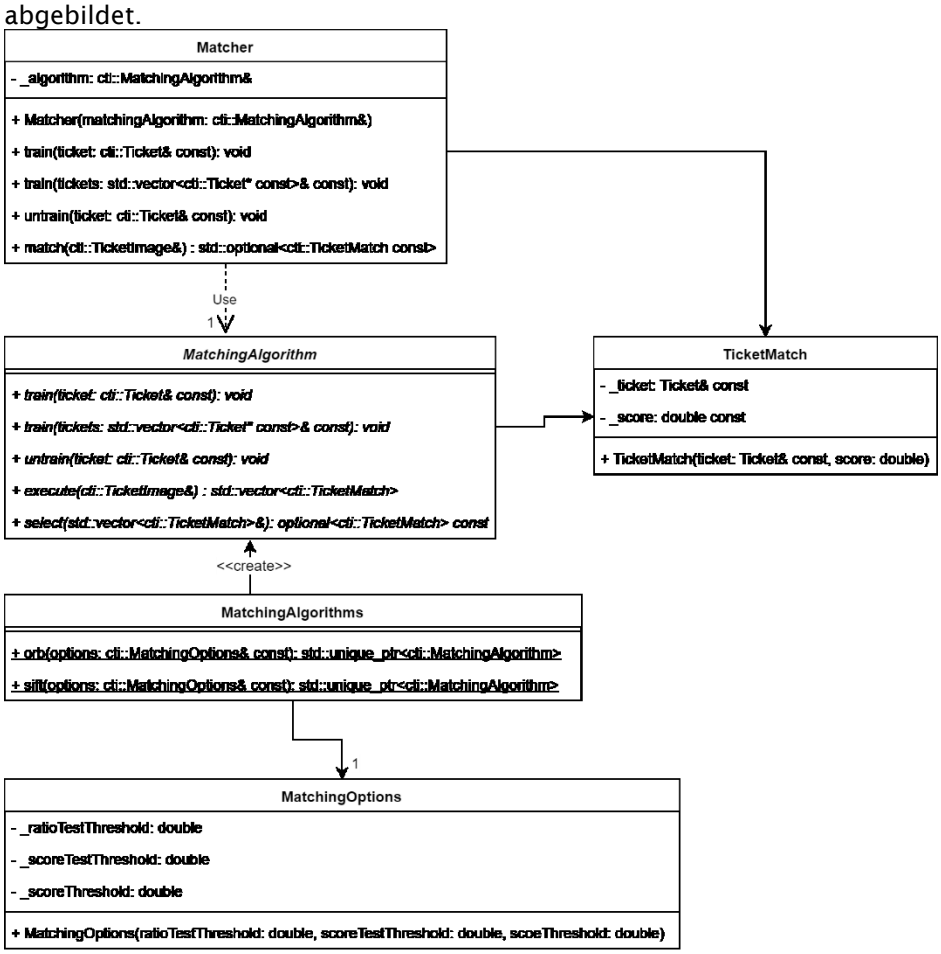

Abbildung 4: Matching-Klassen CTI-Library

In Abbildung 5 sind die Klassen des öffentlichen API für die Matching-Funktionalität der CTI-Library abgebildet.

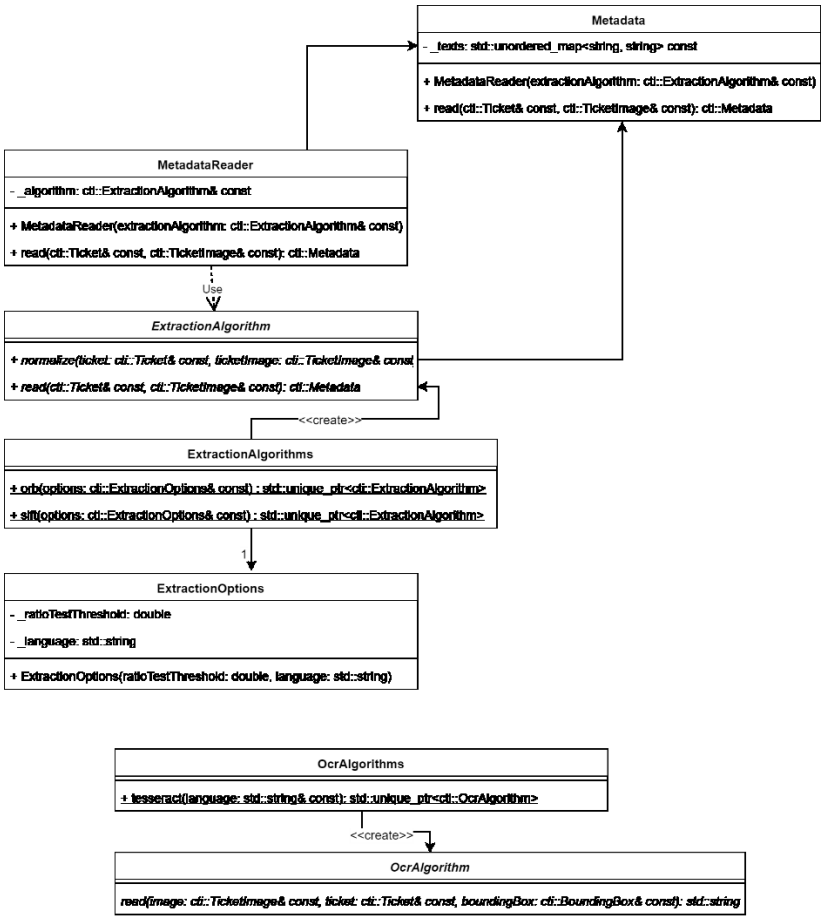

Abbildung 5: Extraktion-Klassen CTI-Library

2.2 JavaCTI-Library

Nachfolgend sind die Domänenklassen aus der JavaCTI-Library aufgeführt.

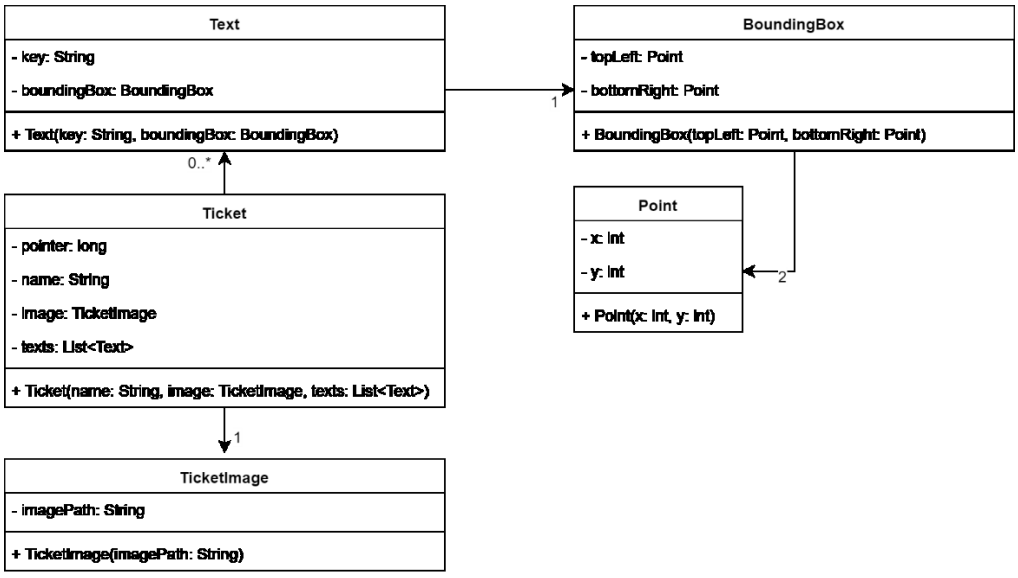

Abbildung 6: Domänenklassen JavaCTI-Library

In Abbildung 7 sind die Klassen des öffentlichen API für die Matching-Funktionalität der CTI-Library abgebildet.

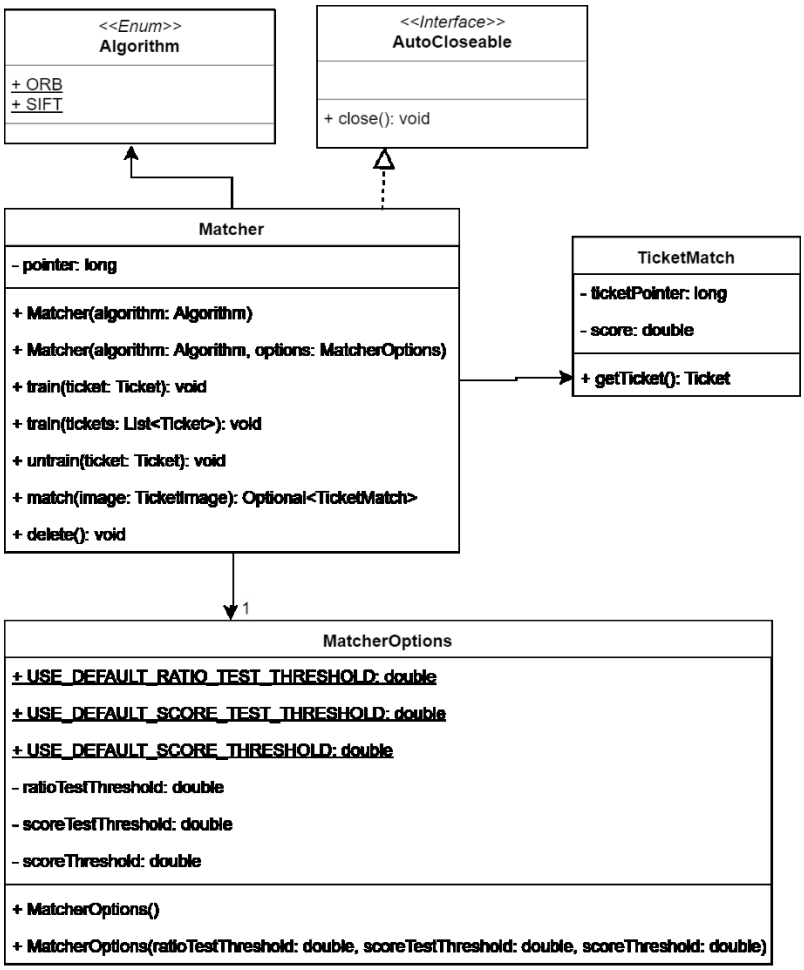

Abbildung 7: Matching-Klassen JavaCTI-Library

In Abbildung 8 sind die Klassen des öffentlichen API für die Extraktion-Funktionalität der CTI-Library abgebildet.

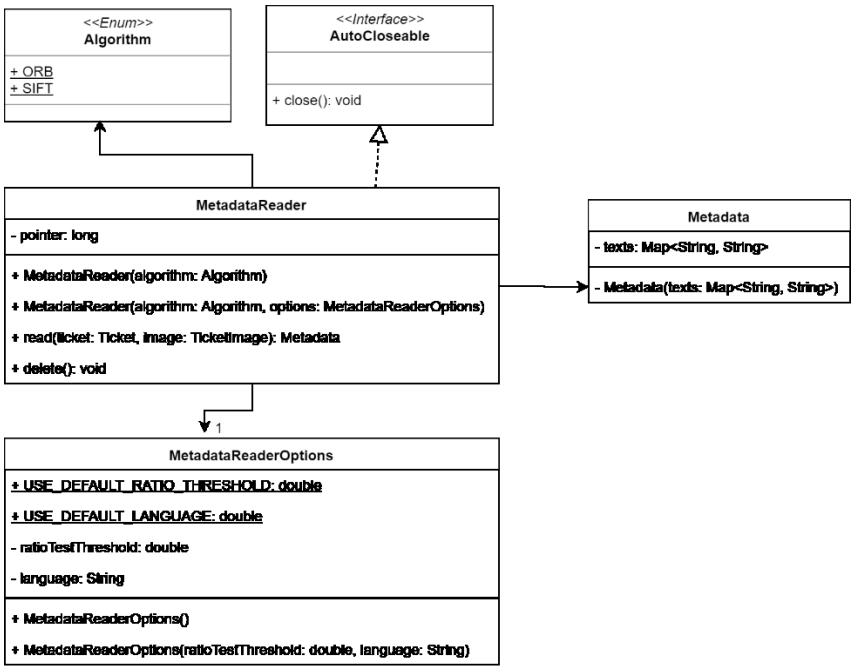

Abbildung 8: Extraktion-Klassen JavaCTI-Library

3 Matching-Algorithmus

In diesem Kapitel wird der Matching-Algorithmus beschrieben, mit dessen Hilfe ein Eingabebild eines in Papierform vorliegenden Vouchers mit vorgängig hinterlegten Vorlagen abgeglichen werden kann, um die richtige auszuwählen.

Der grundsätzliche Ablauf ist wie folgt:

1. Training der Voucher-Vorlagen
2. Verarbeitung des Eingabebilds
3. Abgleich Eingabebild und Vorlagen
4. Auswahl der passendsten Vorlage

Der entscheidende Bestandteil des Matching-Algorithmus ist die Verwendung von sog. Keypoints und Feature Descriptors, mit deren Hilfe identifizierende Punkte zweier Bilder gefunden und verglichen werden können. Es wurden zwei Implementationen von solchen Algorithmen über die Software-Bibliothek OpenCV integriert, welche dem Benutzer der Library zur Verfügung stehen: SIFT [1]und ORB[2].

Nachfolgend werden die aufgeführten Schritte einzeln Beschrieben.

3.1 Training

Jede Voucher-Vorlage wird im Training-Schritt so weit wie möglich vorbereitet, um den Abgleich mit Eingabebildern zu beschleunigen.

Zunächst werden die SIFT-/ORB-Keypoints des in der Vorlage hinterlegten Bildes gefunden. Diese sind Punkte, welche der entsprechende Algorithmus als markant oder identifizierend klassifiziert hat. Die Punkte müssen möglichst unabhängig von der Drehung des Objekts, der Distanz der Kamera zum Objekt und teilweisen Verdeckung des Objekts sein. Das ist allerdings mit den gefundenen Keypoints

[1] Object Recognition from Local Scale-Invariant Features, David G. Lowe, https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf
[2] Oriented FAST and rotated BRIEF (ORB), Ethan Rublee, http://www.willowgarage.com/sites/default/files/orb_final.pdf

noch nicht der Fall. Der Vergleich zweier Keypoints ist nicht sinnvoll, da dieser abhängig von der Drehung sind. Um den Vergleich zu ermöglichen, werden sog. Feature Descriptors für die Keypoints berechnet, welche u.a. Drehungsinvariant sind.

Um den Abgleich der Vorlagen und eines Eingabebilds effizient zu gestalten, werden die Feature Descriptors in einen Index für die Nearest Neighbour Suche eingereiht. Die genauere Funktionalität wird im Kapitel Abgleich beschrieben.

In Abbildung 9 ist ein beispielhaftes Vorlagebild abgebildet.

Abbildung 9: Vorlagebild

In Abbildung 10 ist dasselbe Vorlagebild mit eingezeichneten Keypoints abgebildet.

Abbildung 10: Vorlagebild mit Keypoints

Im nächsten Schritt wird das Eingabebild für den Abgleich vorbereitet.

3.2 Verarbeitung

Das Eingabebild wird in diesem Schritt ebenso wie die Vorlagebilder im Schritt Training verarbeitet, es werden die Keypoints gefunden und die Descriptors berechnet.

Ein beispielhaftes Eingabebild ist in Abbildung 11 abgebildet.

Abbildung 11: Eingabebild

Die gefundenen Keypoints sind in Abbildung 12 abgebildet.

Abbildung 12: Eingabebild mit Keypoints

3.3 Abgleich

Mithilfe der Feature Descriptors des Eingabebildes wird eine Nearest Neighbour Suche auf dem im Schritt Training aufgebauten Suchindex ausgeführt.
Dazu wird die Software-Bibliothek FLANN[3] verwendet, mit der ein Suchindex in Form eines K-d-Baums[4] aufgebaut und durchsucht werden kann. Im Suchindex sind die Feature Descriptors so abgelegt, sodass zu jedem Feature Descriptor des Eingabebildes die «ähnlichsten» zwei Descriptors der Vorlagen effizient gefunden werden können. Das Tupel eines Descriptors des Eingabebildes und von zwei Descriptors der Vorlagen bilden einen Match.

Es werden die zwei ähnlichsten Descriptors gesucht, damit sichergestellt werden kann, dass der Descriptor des Eingabebildes auch genügend eindeutig ist. Dazu werden Matches verworfen, wenn die Distanz des Nächsten zum zweitnächsten Descriptor nicht genug deutlich ist. Dadurch können Descriptors aussortiert werden, welche mit vielen Vorlagen eine Übereinstimmung haben. Dieses Vorgehen ist in SIFT[5] weiter beschrieben.

Das Ergebnis der Suche ist eine Liste von Matches, mit der die korrekte Vorlage im nächsten Schritt ausgewählt werden muss.

[3] Fast Library for Approximate Nearest Neighbors, Marius Muja & David Lowe, https://www.cs.ubc.ca/research/flann/uploads/FLANN/flann_manual-1.8.4.pdf

[4] K-d-Baum, https://de.wikipedia.org/wiki/K-d-Baum

[5] Object Recognition from Local Scale-Invariant Features, David G. Lowe, https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf

3.4 Auswahl

Die Liste der Matches aus dem letzten Schritt wird darauf untersucht, wie viele übereinstimmende Descriptors zu jeder Vorlage gefunden wurden. Anschliessend werden die zwei Vorlagen mit den meisten Übereinstimmungen ausgewählt und untersucht, ob die beste Vorlage deutlich mehr Übereinstimmungen hat als die Zweitbeste. Somit kann verhindert werden, dass eine Vorlage nicht akzeptiert wird, wenn die Übereinstimmung nicht eindeutig ist.

Des Weiteren muss die Beste eine Mindestanzahl an Übereinstimmungen besitzen, um akzeptiert zu werden. Dadurch wird sichergestellt, dass Eingabebilder, welche keiner Vorlage entsprechen und trotzdem einzelne Übereinstimmungen haben, fälschlicherweise einer Vorlage zugeordnet werden.

Wenn eine Vorlage gefunden wurde, welche alle oben beschriebenen Bedingungen erfüllt, wird diese als Ergebnis an den Benutzer der Bibliothek zurückgeliefert.

4 Extraktion-Algorithmus

Nachfolgend wird der Extraktion-Algorithmus beschrieben, welcher es ermöglicht, die Texte eines in Papierform vorliegenden Vouchers mithilfe dessen Eingabebildes und der dazugehörigen Voucher-Vorlage, welche möglicherweise mit dem Matching-Algorithmus herausgefunden wurde.

Der Ablauf ist wie folgt:

1. Verarbeitung
2. Normalisierung
3. OCR
4. Nachverarbeitung

Die einzelnen Schritte werden in den nachfolgenden Kapiteln näher beschrieben.

4.1 Verarbeitung

Im ersten Schritt werden für das Eingabebild erneut die Keypoints gefunden und die Feature Descriptors berechnet, was bereits im Kapitel Training beschrieben wurde. Das gleiche wird für die erkannte Voucher-Vorlage gemacht. Dies ist relevant für den nächsten Schritt, die Normalisierung.

4.2 Normalisierung

Das Eingabebild muss in die Position, Drehung und Skalierung des Vorlagebildes überführt werden, damit die rechteckigen Textbereiche der Voucher-Vorlage darübergelegt und der OCR-Algorithmus über jedem Textbereich angewendet werden kann.

Dazu werden die im Verarbeitungsschritt gefundenen Keypoints mit denjenigen der Vorlage abgeglichen, wie im Kapitel Abgleich beschrieben.

Die Keypoints-Übereinstimmungen zwischen dem Eingabebild und dem Vorlagebild können in eine geometrische Relation gesetzt werden. Das ist möglich unter der Annahme, dass in zwei Bildern dasselbe Objekt aus verschiedenen Perspektiven abgebildet ist. Das bedeutet, dass mithilfe ein oder mehreren markantes Design-Elementen des Vouchers, welches auf beiden Bildern sichtbar ist, die anzuwendende Rotation, Skalierung und Verschiebung bestimmt werden kann.

Mithilfe von OpenCV wird eine sogenannte Homography oder perspective transformation zwischen den beiden Bildern anhand von Bildpunkten gefunden werden. Dies ist eine Matrize, welche die 3D-Transformation beschreibt.

Diese Homography wird anschliessend verwendet, um das Eingabebild in die Position, Drehung und Skalierung des Vorlagebildes zu überführen. Durch diesen Schritt ist das Bild bereit für die Anwendung des OCR-Algorithmus.

4.3 OCR

Nach der Normalisierung des Eingabebildes im letzten Schritt kann nun die Texterkennung stattfinden. Nun kann für jeden der definierten Textbereiche der entsprechende Bildbereich extrahiert werden und dem OCR-Algorithmus übergeben werden. Als OCR-Algorithmus wird Tesseract eingesetzt. Vor Übergabe des Bildbereiches muss dieser allerdings vorbereitet werden, damit der OCR-Algorithmus den Text korrekt erkennt.

Tesseract funktioniert gemäss best priactices [6]am besten, wenn das Bild binär, sprich schwarz/weiss, ist und der Text schwarz und der Hintergrund weiss ist. Um das zu erreichen entstand die Idee, den Hintergrund des Texts in den Bildbereichen zu entfernen und so den Text zu isolieren.

Dazu werden die folgenden Schritte durchgeführt:

1. Konvertierung des farbigen Eingabebildes in Graustufen (= Bild A)
2. Morphologische Dilatation[7] von Bild A (= Bild B)
3. Verwischung von Bild B mithilfe eines Box blur[8] (= Bild C)
4. Subtraktion von Bild C – Bild A (= Bild D)
5. Binary thresholding von Bild D (= Bild E)
6. Invertierung von Bild E (= Bild F)
7. Zeichnen eines dünnen weissen Randes am Rande von Bild F (= Bild G)

Das farbige Eingabebild wird zunächst in ein Graustufenbild konvertiert, um weitere Verarbeitungen einfacher zu gestalten (siehe Abbildung 13).

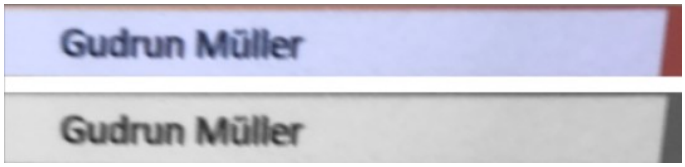

Abbildung 13: Eingabebild und Graustufenbild

Anschliessend wird eine Morphologische Dilatation mehrmals angewandt, was den Effekt hat, dass der Text (fast) vollständig verschwindet (siehe Abbildung 14).

Abbildung 14: Graustufenbild nach Dilatation

Die anschliessende Verwischung bewirkt, dass Rückstände des Texts, welche nach der Dilatation noch übrig sind, verwischt werden (siehe Abbildung 15).

Abbildung 15: Bild nach Verwischung

Anschliessend wird das Graustufen-Bild von dem so gewonnenen Hintergrundbild subtrahiert, um den Text zu isolieren (siehe Abbildung 16).

Abbildung 16: Resultat der Subtraktion

[6] Tesseract improve quality, https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality

[7] Dilation, https://en.wikipedia.org/wiki/Dilation_(morphology)

[8] Box blur, https://en.wikipedia.org/wiki/Box_blur

Das dadurch entstandene Bild ist in Graustufen und muss noch zu einem Binärbild konvertiert werden. Dazu wird ein Algorithmus namens Otsu's method [9]verwendet, welcher anhand eines Histogramms der Intensitätswerte einen globalen Grenzwert berechnet, mit dessen Hilfe entschieden wird, welche Pixel zu weiss und welche zu schwarz geändert werden (siehe Abbildung 17).

Abbildung 17: Schwarz-weiss-Bild

Im so entstandenen Bild ist der Text weiss und aus diesem Grund wird das Bild invertiert, da Tesseract schwarzen Text auf weissem Hintergrund bevorzugt (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**).

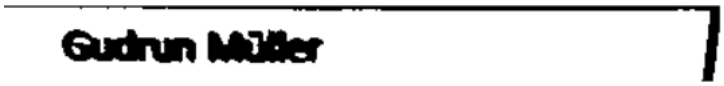

Abbildung 18: Invertiertes Bild

Nun wird das Bild von allen Ecken her mit weisser Farbe geflutet, wodurch der Rand des Textbereichs, der im Bild erscheinen kann, verschwinden. Dieser Rand ist im Bild sichtbar, da bei der Normalisierung des Eingabebildes keine Pixelgenaue Überführung passiert und dadurch ein Teil des Randes in das Bild gelangt. Dieser Rand kann für Tesseract problematisch werden, da dieser dazu führen kann, dass zusätzliche Zeichen wie etwa «|», «-» oder «_» erkannt werden. Die Flutung füllt eine zusammenhängende schwarze Fläche aus, wodurch der Text nicht bedroht ist. Zuletzt wird ein kleiner weisser Rand am Rande des Bildes gezeichnet, um allfälligen Pixellärm, der vom Rand noch übrig ist, zu entfernen (siehe Abbildung 19).

Abbildung 19: Bild nach Flooding und Rand

Nach diesen Verarbeitungsschritten wird das Bild an Tesseract übergeben und der ausgelesene Text wird dem Benutzer der Bibliothek zurückgegeben.

4.4 Nachverarbeitung

Nach erfolgreicher Extraktion der Texte eines Eingabebildes muss der Benutzer der Bibliothek die erhaltenen Texte prüfen und evtl. nachverarbeiten. Dies kann der Benutzer der Bibliothek am besten, da dieser das genaue Format des Textes kennt.
Beispielsweise kann es sinnvoll sein, doppeldeutige Zeichen wie etwa «8» & «B», «5» & «S», «1» & «l» oder «6» & «G» zu ersetzen, falls einige davon in gewissen Texten nicht vorkommen sollten.

Bei einem Datum kann es Sinn machen, eine Nachverarbeitung zu machen, um den Tag, Monat und das Jahr unabhängig von der Präsenz von Trennzeichen, wie etwa einem Punkt, auszulesen, da kleine Zeichen beim OCR verloren gehen könnten.

Es ist daher empfehlenswert, dass das Format der auszulesenden Texte sorgfältig gewählt ist und mithilfe von konkreten Testdaten geprüft wird, welche Nachverarbeitungen notwendig sein könnten.

5 Glossar

Begriff	Beschreibung

[9] Otsu's method, https://en.wikipedia.org/wiki/Otsu%27s_method

CTI	Computerunterstützte Ticketerkennung und Informationsextraktion

6 Abbildungsverzeichnis

Abbildung 1: Projektaufbau und Abhängigkeiten 3
Abbildung 2: CTI-Library 3
Abbildung 3: Domänenklassen CTI-Library 4
Abbildung 4: Matching-Klassen CTI-Library 5
Abbildung 5: Extraktion-Klassen CTI-Library 6
Abbildung 6: Domänenklassen JavaCTI-Library 6
Abbildung 7: Matching-Klassen JavaCTI-Library 7
Abbildung 8: Extraktion-Klassen JavaCTI-Library 8
Abbildung 9: Vorlagebild 9
Abbildung 10: Vorlagebild mit Keypoints 9
Abbildung 11: Eingabebild 10
Abbildung 12: Eingabebild mit Keypoints 10
Abbildung 13: Eingabebild und Graustufenbild 12
Abbildung 14: Graustufenbild nach Dilatation 12
Abbildung 15: Bild nach Verwischung 12
Abbildung 16: Resultat der Subtraktion 12
Abbildung 17: Schwarz-weiss-Bild 13
Abbildung 18: Invertiertes Bild 13
Abbildung 19: Bild nach Flooding und Rand 13

7 Versionskontrolle

Version	Datum	Beschreibung	Autor
1.0.0	02.01.2020	Beschreibung Extraktion aktualisiert, bessere Bilder eingefügt.	Luca Ritz & Lukas Seglias
0.8.0	30.12.2019	Dokumentation	Luca Ritz & Lukas Seglias