UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Software zur Ground-Truth-Verfeinerung des DIVA-HisDB Datensatzes

Max G. Haller[1]

Supervision: Lars Vögtlin[2], Rolf Ingold[3]

2. September, 2022

Department of Informatics - Bachelor Project Report

Département d'Informatique - Departement für Informatik • Université de Fribourg - Universität Freiburg • Boulevard de Pérolles 90 • 1700 Fribourg • Switzerland

phone +41 (26) 300 84 65 fax +41 (26) 300 97 31 Diuf-secr-pe@unifr.ch http://diuf.unifr.ch

[1] max.haller@unifr.ch
[2] lars.voegtlin@unifr.ch, DIVA group, DIUF, University of Fribourg
[3] rolf.ingold@unifr.ch, DIVA group, DIUF, University of Fribourg

Zusammenfassung

Diese Arbeit stellt eine Software vor, die den Datensatz DIVA-HisDB für Trainingszwecke flexibel aufbereiten lässt. Die entwickelte Python-Bibliothek unterstützt eine Reihe nützlicher Werkzeuge für die Manipulation und Repräsentation dessen Bilder und Ground-Truths. Mit einer erweiterbaren und flexiblen Softwarearchitektur geben wir Nutzer:innen zudem volle Kontrolle über den Entwicklungsprozess und ermöglichen ihnen, ihre eigene Algorithmen zur Dokumentenanalyse zu implementieren. Damit bieten wir eine effiziente und wiederverwendbare Alternative für die manuelle Ground-Truth-Entwicklung der historischen Dokumente des Datensatzes an.

Keywords: Bachelor project report, DIVA group, Ground-Truth, Image Processing

Inhaltsverzeichnis

Kapitel 1

Einführung

Im letzten Jahrzehnt haben maschinelle Lernmethoden grosse Fortschritte in der bildbasierten Dokumentenanalyse bzw. Document Image Analysis (DIA) gemacht [29]. Mit wachsender Rechenleistung von Computern und grossen Mengen von einfach verfügbaren Daten sind hochentwickelte Methoden für die Objekterkennung [30], Bildsegmentation [34] und Bildsynthese [35] entwickelt worden. Wegen mangelnden Datensätzen bleiben dieselben Aufgaben für historische Dokumente jedoch grossenteils ungelöst [10][23]. Zwar liegen bereits verschiedene DIA-Methoden vor – Bildwiderherstellung, Layoutanalyse, Binarisierung, Texterkenneung etc. [23] - jedoch kaum Ground-Truth, um diese oft sehr lernintensiven Methoden zu trainieren.

1.1 Verwandte Arbeit

Neue, annotierte Datensätze historischer Dokumente zu entwickeln, bedingt das Fachwissen von Expert:innen auf dem Gebiet, und ist somit aufwendig und kostspielig [32][33]. Deswegen sind verschiedene Programme zur Ground-Truth-Synthetisierung entickelt worden [29][38]. Mit Software wie DocCreator und DocEmul können Dokumente generiert werden, welche handgeschriebene Texte emulieren und Zerfallsformen von Textdokumenten simulieren (Verblassen, Phantomtext, Zerknittern, Löcher, durchgesickerte Tinte etc.) [23][8]. Andere Projekte befassen sich mit der effizienteren Annotation historischer Manuskripte [4][24], bleiben aber kostspielig, weil sie für die Beschriftung der Textelemente weiterhin auf Fachpersonen angewiesen sind [23]. Wir konzentrieren uns deshalb auf die algorithmische Überarbeitung und Erweiterung von bereits existierendem Ground-Truth.

1.2 Ausrichtung

Der Datensatz DIVA-HisDB ist ein grosser, annotierter Datensatz von historischen Dokumenten, die nach ihrer Komplexität ausgewählt worden sind [33]. In dieser Arbeit stellen wir eine öffentlich zugängliche Software zu dessen Verfeinerung vor.[1] Sie ermöglicht es Forschenden, den existierende Ground-Truth um zusätzliche Layoutklassen (wie z.B. x-Height, Ascender und Descenders[2]) zu erweitern und auf eine gewünschte Dimension zu verkleinern bzw. zuzuschneiden. Seine Textelemente können verziert, neu gruppiert und kombiniert werden, wobei die Softwarearchtektur Forschenden erlaubt, ihre eigenen Algorithmen zu implementieren und den Ground-Truth somit beliebig zu modifizieren. Damit bieten wir ein wiederverwendbares und erweiterbares Framework, das eine effiziente Verfeinerung des Datensatzes ermöglichen soll.

[1] `https://github.com/DIVA-DIA/HisDB_GT_Refinement`

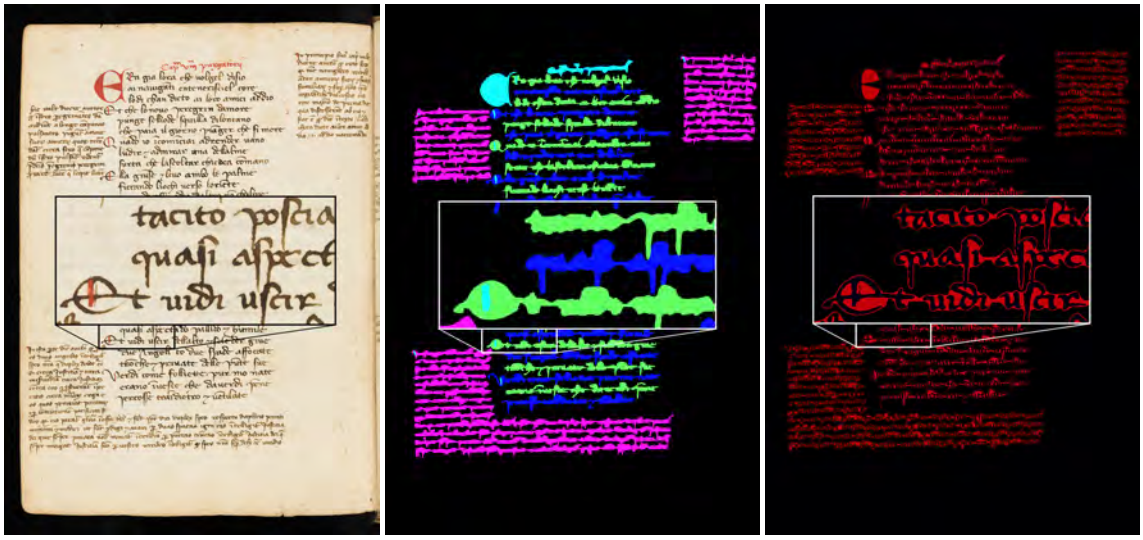
[2] Die Kategorisierung einer Textzeile in x-Height, Ascenders und Descenders ist eine populäre Methode zur Unterteilung von Textlinien [5].

Kapitel 2

Ausgangslage

2.1 Datensatz

Im Rahmen dieser Arbeit verwenden wir den DIVAHisDB-Datensatz [33], ein Datensatz von drei herausfordernden, mittelalterlichen Manuskripten, die nach ihrer Komplexität ausgewählt worden sind. Der Datensatz besteht aus 150 Seiten gescannter Originalbilder, einem vektorisierten und einem pixelbasierten Ground-Truth. Die Originalbilder und der pixelbasierte Ground-Truth haben eine Auflösung von 4872 mal 6496 Pixel und sind als Joint Photographic Experts Group (JPEG) respektive Portable Network Graphic (PNG) gespeichert. Während PNG-Bilder verlustfrei komprimiert werden, kann die Kompression von JPEG-Bildern zu Informationsverlust und Farbanomalien führen. Die Dateigrösse der beiden Bilder unterscheiden sich deshalb deutlich mit durchschnittlich ca. 500 KB für die JPEG-, und 5 MB für die PNG-Bilder.



(a) Originalbild (b) Vektorbasierter Ground-Truth (c) Pixelbasierter Ground-Truth

Abbildung 2.1: Illustriert sind die drei Repräsentationsformen des DIVA-HisDB-Datensatzes: Das JPEG-basierte Originalbild (2.1a), der vektorbasierte Ground-Truth (2.1b) und das PNG-basierte Ground-Truth-Bild (2.1c). Der vektorbasierte Ground-Truth ist zwar als Extensible Markup Language (XML) in Textform gespeichert, zu Illustrationszwecken hier jedoch mit willkürlichen Farben in Bildform dargestellt (Main-Text-Elemente alternierend und die anderen beiden Layoutklassen, Kommentare und Dekorationen, einfarbig).

Der vektorbasierte Ground-Truth ist im PAGE-Format, nach S. Pletschacher and A. Antonacopoulos (2010) [28], als XML-Dokument gespeichert, welches die Dokumente einheitlich als hierarchische Datenstruktur repräsentiert [6]. Er ist mit GraphManuscribble, einer halbautomati-

RGB-Kanäle	Blaukanal (binär)	Layoutklassen
[0 0 1]	0b00001	Background (out of page)
[0 0 2]	0b00010	Comment
[0 0 4]	0b00100	Decoration
[0 0 6]	0b00110	Comment & Decoration
[0 0 8]	0b01000	Main-Text-Body
[0 0 10]	0b01010	Main-Text-Body & Comment
[0 0 12]	0b01100	Main-Text-Body & Decoration
[0 0 14]	0b01110	Main-Text-Body & Decoration & Comment
[128 0 2]	0b00010	Boundary-Pixel & Comment
[128 0 4]	0b00100	Boundary-Pixel & Decoration
[128 0 6]	0b00110	Boundary-Pixel & Comment & Decoration
[128 0 8]	0b01000	Boundary-Pixel & Main-Text-Body
[128 0 10]	0b01010	Boundary-Pixel & Main-Text-Body & Comment
[128 0 12]	0b01100	Boundary-Pixel & Main-Text-Body & Decoration
[128 0 14]	0b01110	Main-Text-Body & Decoration & Comment

Tabelle 2.1: Zeigt die 15 Layoutklassen des pixelbasierten Groundtruth (rechte Spalte), die dazugehörigen Red-Green-Blue (RGB)-Werte (linke Spalte), sowie deren binären Representationsform (mittlere Spalte).

schen, interaktiven Schriftannotierungssoftware der Universität Freiburg, erarbeitet worden [14]. Dabei werden drei Layoutklassen unterschieden: Main-Text-Body (Haupttext, bzw. Main-Text), Comments (Kommentare) und Decorations (Dekorationen). Jedes Textelement befindet sich in einer Textregion, hat eine Boundingbox (bzw. Rahmen) und kommt mit einer Baseline (bzw. Schriftlinie), sofern es Textliniencharakter hat.[1] Ein Textelement gehört zu jeweils genau einer Layoutklasse. Der pixelbasierte Ground-Truth kategorisiert die gleichen drei Layoutklassen und speichert diese Information als Binärwert im Blaukanal des RGB-Farbraums jedes Pixels. Wie in Abbildung 2.1b hervorgehoben wird, entstehen beim Überlappen zweier Seitenelemente, wie bspw. der hellblauen Dekorierung und der darunterliegenden, grünen Textlinie, Konflikte in der Darstellung eines pixelbasierten Ground-Truths. Fontini et al. [33] haben sich dafür entschieden, die drei Basisklassen des Pixel-Ground-Truth um 12 Hilfsklassen zu insgesamt 15 Layoutklassen zu erweitern (siehe Tabelle 2.1). Mischformen sind dank der binären Codierung einfach zu generieren und dekodieren. Überschneiden sich bspw. Kommentare (0b0010) und Dekorationen (0b0100), ist die Mischform (0b0110) einfach zu interpretieren. Als Boundary-Pixel sind all jene Pixel markiert, die sich innerhalb der Polygone des vektorisierten Ground-Truths befinden, jedoch nicht Schriftpixel sind. Diese Information ist im Rotkanal des pixel-basierten Ground-Truths gespeichert (siehe Abbildung 2.1c). Sie werden im Grossteil dieser Arbeit ignoriert (als Background-Pixel behandelt), können jedoch jederzeit wiederhergestellt werden.

2.2 Zielsetzung

Ziel dieser Arbeit ist es, Forschenden eine möglichst flexible Software anzubieten, die ihnen erlaubt, neue Ground-Truths zu entwickeln, und diese um neue Layoutklassen zu erweitern. Wir beweisen die Relevanz und Anwendbarkeit der Software, indem wir eine Liste von Funktionalitäten implementieren, die zeigen, dass die Software in der Lage ist, die Bedürfnisse der Forschenden zu erfüllen. Die Anforderungen werden nach ISO-Standard [21] in funktionale und nicht funktionale Anforderungen unterteilt. Funktionale Anforderungen legen konkretes Verhalten bzw. Implementationsanforderungen fest, während nichtfunktionale Anforderungen über diese hinausgehen und oft als Qualitätsmerkmale verstanden werden [12].

[1] Kommentare und Main-Text haben Textliniencharakter und kommen mit einer Schriftlinie. Dekorationen hingegen fehlt eine solche Linie.

2.2.1 Funktionale Anforderungen

Die funktionalen Anforderungen an das Programm können in drei Kategorien unterteilt werden: Extraktion, Interpretation und Repräsentation (siehe Abbildung 2.2).

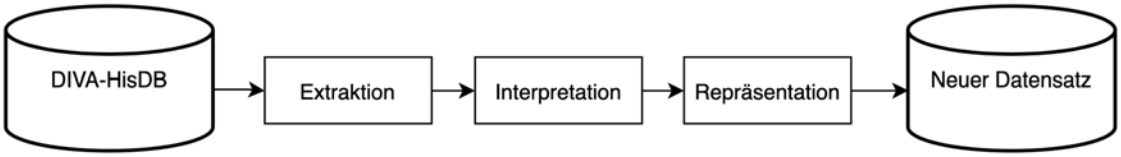


Abbildung 2.2: Flowdiagram des Ground-Truth-Refiner.

Extraktion Ziel der Extraktion ist das verlustfreie Einlesen und sinnvolle Zwischenspeichern der Informationen in einer objektorientierten Architektur, die anschliessend flexibel manipuliert werden kann. Die Software...

- kann den vektorbasierten Ground-Truth verlustfrei einlesen und speichert die Textelemente in einer objektorientierten Architektur.
- kann den pixelbasierten Ground-Truth verlustfrei einlesen und speichern.
- kann die Originalbilder verlustfrei einlesen und speichert diese zwischen.

Interpretation Der Hauptzweck des Programms besteht in der Interpretation und Manipulation des Ground-Truth. Die Software...

- kann die beiden Ground-Truths und die Originalbilder auf eine gewünschte Grösse skalieren (z.B. auf 3000px zu 4000px).
- kann die beiden Ground-Truths und die Originalbilder auf eine gewünschte Grösse zuschneiden (z.B. oben und unten 300px weglassen und auf der linken Seite 200px).
- kann die Textlinien dekorieren. Als Proof of Concept sollen die Textlinien mit Ascender und Descender verziert werden.
- kann die Kommentare sortieren und gruppieren.
- kann den Ground-Truth um neue Layoutklassen erweitern.

Repräsentation Schlussendlich muss die Software in der Lage sein, den neuen Ground-Truth zu repräsentieren und zu speichern. Die Software...

- kann den Groundtruth mit verschiedenen Strategien färben (bspw. einfarbig, abwechselnd, jedes Element unterschiedlich).
- kann einzelne Elemente, Subregionen und ganze Regione sichtbar bzw. unsichtbar machen.
- kann den neuen vektorbasierter Ground-Truth zur einfachen Weiterverarbeitung als JavaScript Object Notation (JSON) [11] speichern.
- kann den neuen pixelbasierten Ground-Truth speichereffizient als indexiertes Bild speichern (wie bspw. Graphics Interchange Format (GIF)[19]).
- kann die dazu gehörigen Originalbilder verlustfrei speichern (bspw. als PNG).

2.2.2 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen beschreiben im Unterschied zu den funktionalen Anforderungen allgemeine Qualitätsmerkmale an die Software und können nach ISO-9126-Standard in Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit/Wartbarkeit und Übertragbarkeit eingteilt werden [20]. Weil das Ziel dieser Arbeit kein fertiges Produkt, sondern ein erweiterbares Framework für die automatisierte Ground-Truth-Verfeinerung ist, stehen die Kriterien der Änderbarkeit/Wartbarkeit und Benutzbarkeit im Vordergrund. Fachpersonen des DIA-Gebiets sollen in der Lage sein, die Softwarearchitektur schnell zu verstehen und für ihre Zwecke anzupassen. Mit einer robusten und intuitiven Softwarearchitektur wollen wir Nutzer:innen die Möglichkeit geben, den Ground-Truth um zusätzliche Layoutklassen und Textelemente beliebig zu erweitern, z.B. mit eigenen Algorithmen zur Layoutanalyse wie dem Run Length Smoothing Algorithm (RLSA) [26]. Effizienz ist als untergeordnetes Ziel definiert, solange der neue Ground-Truth innert nützlicher Frist entwickelt werden kann. Es folgt eine detaillierte Auflistung der nichtfunktionalen Anforderungen.

Änderbarkeit/Wartbarkeit Die Software...

- kann problemlos erweitert werden (beispielsweise Layoutklassen und Dekorationen) und Algorithmen der Layoutanalyse (Modifizierbarkeit).
- ist robust gegenüber Änderungen und weist eine niedrige Wahrscheinlichkeit für unerwartetes Verhalten auf (Stabilität).
- ermöglicht einfache Fehlerfindung (Analysierbarkeit).
- muss keine umfassenden Tests unterstützen (Testbarkeit).

Benutzbarkeit Die Software...

- stellt einige Beispiele für Entwicklungsszenarien zur Verfügung, die auch von einem bzw. einer unerfahrernen Programmierer:in bedient werden können (Bedienbarkeit).
- umfasst eine intuitive Softwarearchitektur, die für eine Fachperson verständlich ist (Verständlichkeit).
- kann von einer Fachperson schnell erlernt und für ihre Zwecke angewendet werden (Erlernbarkeit).

Effizienz Die Software...

- kann den DIVA-HisDB in einer Nacht (8 Stunden) überarbeiten. Mit 150 Seiten muss die Laufzeit pro verarbeiteter Seite unter 3.2 Minuten liegen.

Funktionalität Die Software...

- implementiert die funktionalen Anforderungen aus Sektion 2.2.1 angemessen und richtig (Funktionalität).
- muss nicht in der Lage sein mit anderen Systemen zusammenzuarbeiten (Interoperabilität).
- braucht keinen Schutz vor unberechtigtem Zugriff (Sicherheit).
- muss keine gestzlichen Vorschriften erfüllen (Ordnungsmässigkeit).

Übertragbarkeit Die Software...

- lässt sich ohne grossen Aufwand in eine andere Umgebung übetragen (Anpassbarkeit).
- kann einfach installiert werden (Installierbarkeit).
- kann mit anderen Anwendungen koexistieren (Koexistenz).

Zuverlässigkeit Die Software....

- scheitert nicht an Nicht-Einhalten der spezifizierten Schnittstelle (Fehlertoleranz).
- kann 8 Stunden lang fehlerfrei 150 Seiten entwickeln (Reife).
- bricht das Programm während seiner Laufzeit, gehen die bereits entwickelten Seiten nicht verloren (Widerherstellbarkeit).

2.3 Mittel

Die Software wird in Python Version 3.8 [37] und den Packages Pillow [9], Numpy [16] und Skyimage [36] geschrieben. Sie wird auf Github des Forschungsteams Document, Image, and Video Analysis Group (DIVA) der Universtität Freiburg/Fribourg veröffentlicht und für alle zugänglich (open-source) gemacht. Zur einfachen und sicheren Übetragbarkeit wird die Software in einem Anaconda-Environment entwickelt [1]. Mit Sphinx stellen wir eine detaillierte Dokumentation als PDF und Website zur Verfügung [7].[2]

[2] `https://github.com/DIVA-DIA/HisDB_GT_Refinement`

Kapitel 3

Softwareentwicklung

Der Ground-Truth-Refiner ist prototypenbasiert entwickelt worden. Die finale Version der Software umfasst drei Packete (siehe Abbidlung 3.2). Das erste kümmert sich um die Ground-Truth-Speicherung, das zweite stellt Tools für dessen Manipulation zur Verfügung und das dritte orchestriert diese Tools im Sinne des bzw. der User:in. Wir stellen eine umfassende Dokumentation des Codes zu Verfügung und zeigen mit Beispielen, zu was diese Software in der Lage ist.[1] Es folgt eine Erläuterung der wichtigsten Designentscheidungen und Algorithmen.

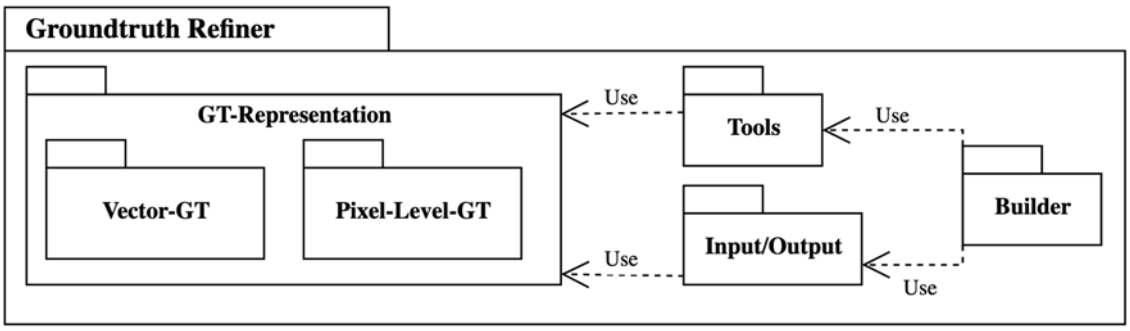


Abbildung 3.1: Package-Diagram der Software. Der Builder koordiniert, ob und in welcher Reihenfolge die Tools (Werkzeuge) gebraucht werden, und verwaltet den den Input/Output (I/O).

3.1 Ground-Truth-Repräsentation

Die Ground-Truth-Repräsentation bringt mit Abstand am meisten softwaretechnische Komplexität mit sich. Über die Hälfte der Klassen dienen dazu, den Ground-Truth zu repräsentieren und dessen reibungslose Manipulation zu garantieren. Wie in Sektion 2.1 beschrieben, unterscheiden sich die Datenstrukturen des vektorisierten Ground-Truths fundamental vom pixelbasierten Ground-Truth. Es liegt nahe, die beiden Repräsentationen auch in der Softwarearchitektur voneinander zu trennen. Übergreifendes Verhalten wie z.B. das Zuschneiden (`crop()`) oder Skalieren (`resize()`) der Groundtruths kann durch Interfaces zur Verfügung gestellt werden werden.

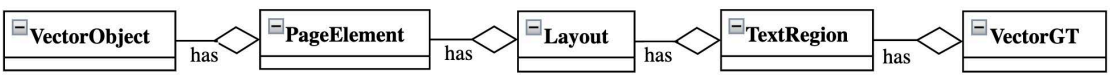


Abbildung 3.2: Aufbau des vektorisierten Ground-Truth. Ein Vektor-GT besteht aus Textregionen, Layouts, Seitenelementen und Vektorobjekten.

[1] `https://github.com/DIVA-DIA/HisDB_GT_Refinement`

3.1.1 Vektorbasierter Ground-Truth

Der Vektor-Ground-Truth ist im `VectorGT`-Modul definiert und speichert die Seitenelemente als vektorbasierte Objekte, die in Layouts und Textregionen gruppiert werden können.

VectorObject Vektorobjekte, d.h. Polygone, Rechtecke, Linien etc., sind im `VectorObject`-Modul definiert und bilden die kleinsten Bauteile einer Seite. Sie werden ausschliesslich vom `PageElement`-Modul gebraucht.

PageElement Seitenelemente (`PageElement`) repräsentieren jegliche Ausprägung eines Textobjekts. Sie sind aus mindestens einem Vektorobjekt aufgebaut, enthalten Informationen zu ihrer Layout-Klasse, Farbe und Sichtbarkeit.

TextRegion Textregionen (`TextRegion`) organisieren den Inhalt einer Seite. Sie speichern die Textelemente in Layouts (`Layout`) und können geteilt oder vereint werden.

Die Herausforderung besteht darin, diese Objekte mit geringem Aufwand um neue Eigenschaften zu erweitern und in Verbindung miteinander setzen zu können. Wir lösen diese Herausforderung mit dem Decorator- und dem Composite-Entwurfsmuster.

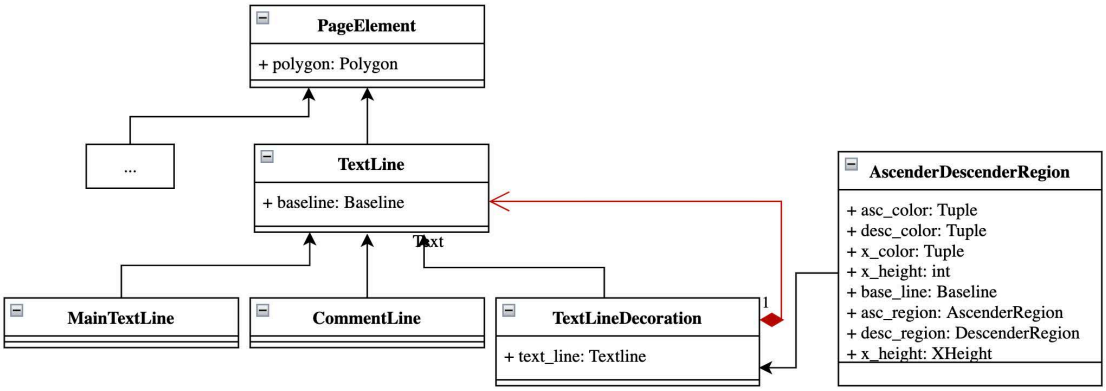


Abbildung 3.3: Ausschnitt aus dem UML Diagram des `PageElement`-Modul und dessen Decoration-Pattern. Das Entwurfsmuster dient dazu, eine Textlinie auszuschmücken. In diesem Fall kann eine Textlinie mit Ascenders und Descenders dekoriert werden.

Decorator-Pattern Das Decorator-Pattern [13] ist ein strukturelles Entwurfsmuster, dass einem Objekt ermöglicht (in diesem Fall `TextElement)` eine beliebige Anzahl an Eigenschaften anzunehmen, ohne das Basiselement zu verändern – so kann beispielsweise eine Textlinie um Ascender und Descenders ergänzt werden, ein Initialelement hinzugefügt oder als Titel gelabelt werden (siehe Abbildung 3.3). Dadurch, dass die Decorator-Elemente vom gleichen Typ sind wie die Basiselemente (`PageElement`), muss ein Klient nicht zwischen verschiedenen Ausprägungen der dekorierten Textelemente unterscheiden. Sollte ein Verhalten des Basiselements nicht mehr erwünscht sein, kann auch dieses mit einem Decorator überschrieben und entfernt werden. Auf der Ebende der Seiten- und Textelemente ist der Ground-Truth-Refiner somit flexibel und erweiterbar.

Composite-Pattern In vielen Fällen ist es wünschenswert, die Textelemente in logischen Zusammenhang zu bringen, also in Gruppen teilen oder ordnen zu können. Hier setzt das Composite-Pattern an [13]. Mit der Unterteilung eines Textes in verschiedene Regionen und Subregionen (bspw. Illustrationen, die Teil des Haupttextes sind, Kommentare, die sich auf bestimmte Textelemte beziehen etc.) kann eine Textseite als komplexe, hierarchische Struktur beschrieben werden. Das Composite-Pattern ist ein strukturelles Entwurfsmuster, welches dem Klienten ermöglicht,

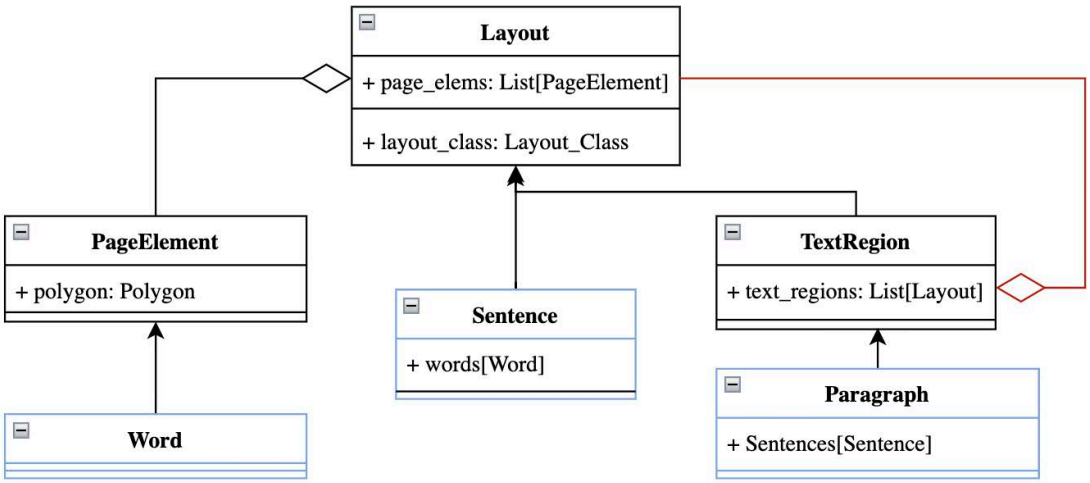


Abbildung 3.4: UML Diagram des Composite Entwurfsmuster zur Illustration mit zusätzlichen Klassen (blau). Eine Textregion ist sowohl ein Layout, als auch ein Container für Layouts - Aggregationsbeziehung in rot dargestellt. Ein Klient muss also nicht unterscheiden zwischen Textregion und Layout. In blau sind zusätzliche Klassen dargestellt die interessant sein könnten für den Fall, dass der Vektor-GT um Wörter, Sätze und Paragrafen ergänzt werden sollte.

alle Elemente einer verschachtelte Baumstruktur einheitlich zu behandeln und einfach zu traversieren [13]. Soll der Ground-Truth um Pargraphen, Sätze und Wörter erweitert werden, kann das dank diesem Entwurfsmuster mit wenig Aufwand implementiert werden (siehe Abbildung 3.4). Algorithmen, welche die Elemente in logischen Zusammenhang bringen sollen (bspw. sortieren, oder Subgruppen finden), können über die Layouts iterieren, um diese zu manipulieren. So kann es beispielsweise interessant sein, ein Comment-Text-Element basierend auf dessen Position und Abstand mit anderen zu gruppieren (siehe Sektion 3.2.4), oder die Main-Text-Elemente linksorientiert, aszendierend nach ihrer y-Position zu sortieren (siehe Sektion 3.2.6).

Die zwei strukturellen Entwurfsmuster arbeiten auf zwei unterschiedlichen Ebenen. Das Decorator-Pattern erweitert Seitenelemente und deren Funktionalitäten, während das Composite-Pattern die Seitenstruktur organisiert und eine einfache Schnittstelle für deren Manipulation zu Verfügung stellt.

3.1.2 Pixelbasierter Ground-Truth

Die Klassen des `PixelLevelGT`-Moduls speichern die Originalbilder und die dazugehörenden, pixelbasierten Ground-Truths. Der Pixel-Level-Ground-Truth besteht, wie der Name sagt, aus mehreren Informationsschichten. Inspiriert durch dieses Schichten-Prinzip ist die Klasse `Layer` erstellt worden. Den Layoutklassen des PixelGTs von Alberti et al. [2] wird je eine Schicht zugeordnet (siehe Abbildung 3.5), welche die Pixel-Informationen in ein binäres Numpy-Array speichert. Schichten können ein- oder ausgeblendet werden, eine bestimmte Farbe haben, mit anderen Schichten kombiniert werden, Vektorobjekte annehmen und auf sich zeichnen. Damit ist die `PixelLevelGT-` und `Layer`-Klasse elementar für die Kombination der beiden Ground-Truths.

3.1.3 Layoutklassen

Um die Erweiterbarkeit der Software und dessen reibungslosen Ablauf zu verwalten, steht in der Mitte der Architektur das Enum `LayoutClasses`. Die Layoutklassen dienen als eindeutiger Identifikationsschlüssel der Schichten des `PixelLevelGT`, der RGB-Werte der `ColorTable`, den booleschen Werte der `VisibilityTable` und den Seitenelementen des `VectorGT`. Damit entspricht die Aufgabe des Enums der Funktion eines Primärschlüssels in einer relationalen Datenbank [17]. Sollten neue

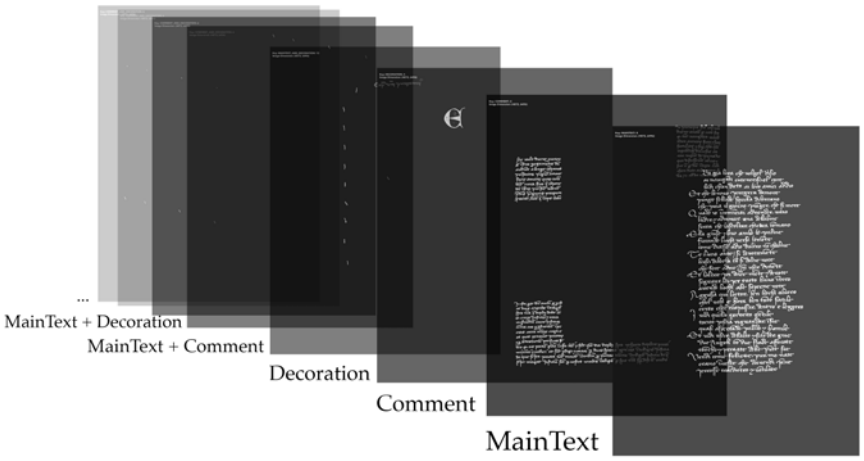


Abbildung 3.5: Illustration des Pixel-Level-Groundtruths und seinen Schichten (bzw. Layers).

Layoutklassen hinzukommen, kann das Enum erweitert werden, ohne bestehende Python-Skripte verändern zu müssen. Die Software ist in dieser Hinsicht rückwärtskompatibel, d.h., dass auch veraltete Versionen des Enums nach einem Update verwendet werden können (siehe Dokumentation für genauere Informationen).

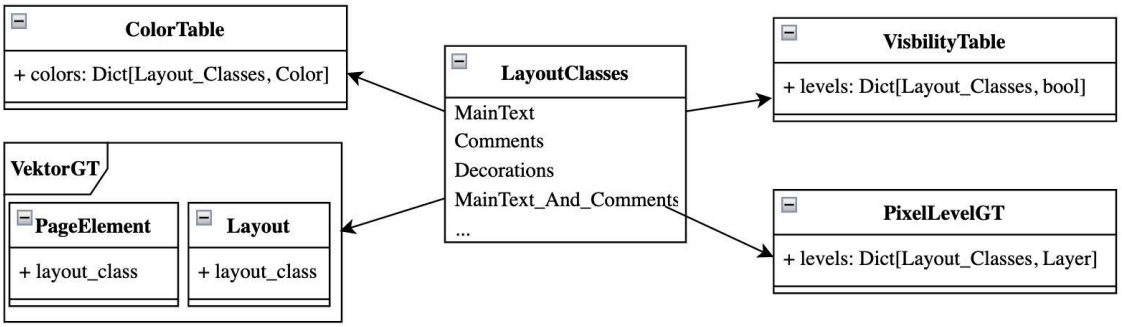


Abbildung 3.6: Abbildung des Enums und seiner Verwaltunsfunktion. Es dient als Schlüssel für die Wörterbücher in `ColorTable`, `VisibilityTable` und `PixelLevelGT` als Feld in `PageElements` und `Layout`.

3.1.4 Input/Output

Das Input- und Outputmodul stellt Leser- und Schreibklassen für verschiedene Formate zu Verfügung. Konkret sind das XML und JSON für den vektorbasierten Ground-Truth, jegliche von Pillow unterstützte Formate für das Originalbild und den pixelbasierten Ground-Truth. Die Klassen sind leicht zu erweitern, so dass auch neue Formate mit geringem Aufwand unterstützt werden können.

3.2 Tools

Mit dem Visitor-Pattern [13] stellen wir Entwickler:innen eine Schnittstelle für die Manipulation des Ground-Truth zur Verfügung. Das Verhaltensmuster entkoppelt algorithmische Aufgaben von den zu manipulierenden Objekten. Visitors, bzw. Besucher, können die `Page`-Klasse besuchen, welche alle Ground-Truth Informationen, inkl. dem Originalbild, referenziert. Verschiedene Versionen der Algorithmen können geschrieben, ausgetauscht und je nach Bedürfnis flexibel angewendet werden. Es folgt ein Überblick über die so implementierten Funktionalitäten.

3.2.1 Cropping

Der Cropping-Visitor schneidet das Bild auf eine Wunschdimension zu. Dabei kann angegeben werden, ob die Seite links- oder rechtsgebunden ist. Die aktuelle Anwendung der Cropping-Funktion zielt auf das Wegschneiden unnützer Randpixel ab, kann jedoch theoretisch auch für Illustrationszwecke oder gezieltes Ausschneiden von Textregionen benutzt werden. Das Verhalten wird durch das Interface `Croppable` vererbt.

3.2.2 Coloring

Mit Hilfe einer Farbtabelle im JSON-Format, kann ein:e Klient:in die verschiedenen Vektorobjekte des `VectorGT` bzw. Ebenen des `PixelGT` individuelle Farben zuordnen. Der Colorer übernimmt die Verantwortung, diese Information zu lesen und den Zielelementen zuzuordnen. Er unterstützt drei Strategien: Jedes `TextElement` einer Layoutklasse gleich anmalen, Jedes `TextElement` einer Layoutklasse anders anmalen oder jedem eine andere Farbe geben. Prinzipiell ist jede weitere Anfärbungsstrategie denkbar. Dazu müssen die Farben in der Color-Palette entsprechend gewählt werden. Sollen bestimmte Gruppen eine bestimmte Farbe (oder Iteration von Farben) haben, kann ein neuer `Colorer` geschrieben werden, der die Textregionen nach den gewünschten Reglen einfärbt.

3.2.3 Visibility

Analog zum `Colorer` liest der `Visibility-Visitor` eine Sichtbarkeitstabelle ein, die definiert, ob eine Layoutklasse sichtbar sein soll oder nicht. So kann mit einer `VisibilityTable` definiert werden, dass nur Main-Text-Elemente oder nur Kommentare illustriert werden sollen. Sind nur einzelne Textregionen oder Textelemente von Interesse, kann ein neuer Besucher geschrieben werden, der die gewünschte Funktionalität implementiert.

3.2.4 Grouping

Das Grouping-Tool wirkt auf der Ebene der Textregionen. Es kreiert, teilt und vereint die Textelemente je nach Algorithmus zu logischen (Sub)-Gruppen. Zurzeit unterstützt der `Grouper` das Clustering von Textelementen basierend auf ihrer kleinsten x-Koordinate (mit der `histogram()`-Funktion von Numpy [16]) und das Unterteilen in Blöcke von nahe, beieinandergelegenen Textelementen (basierend auf einem user-spezifizierten Threshold).

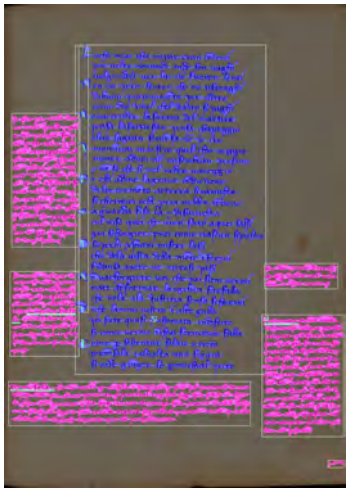


Abbildung 3.7: Grouping mit dem Klient-Code von Listing 3.1. Gruppen sind weiss umrandet.

3.2.5 Resizing

Mit 4872 auf 6496 Pixel sind sowohl die Ground-Truth-Bilder, als auch die Originalbilder, speicherintensiv und können nicht als Ganzes in die Graphics Processing Unit (GPU) gewöhnlicher Computer geladen werden, was das Testen neuer DIA-Methoden (wie z.B. neuronalen Netzwerken) verlangsamt und erschwert. Der `Resizer`-Visitor kümmert sich um die synchronisierte Verkleinerung aller Ground-Truth-Informationen und warnt Nutzer:innen, vor gefährlichen Zieldimensionen (die keine ganzen Teiler des Originals sind). Es handelt sich beim Resizer also tatsächlich um einen Down-Sizer, welcher die Informationen des Originals möglichst gut bewahren soll. Die Skalierung des vektorbasierten Ground-Truths ist eine triviale Vektorenrechnung und kann in wenigen Zeilen Code implementiert werden. Die grössere Herausforderung stellt die Skalierung des pixelbasierten Ground-Truth dar. Dazu sind verschiedene Vorgehensweisen getestet worden: Majority-Wins, Minority-Wins und Resizing mit Weichzeichnung und Binarisierung (Blur-And-Binarize).

Majority-Wins vs. Minority-Wins

Die beiden Strategien Majority-Wins und Minority-Wins basieren auf der gleichen Idee: Gegeben sei ein Skalierungsfaktor: 2,3,4,5,..., welcher den ursprünglichen Ground-Truth ganzzahlig teilt. Wird beispielsweise der Skalierungsfaktor 4 gewählt, wird im Raster von 4 mal 4 Pixel, das Pixel gewählt, das am häufigsten vorkommt (Majority-Wins) oder am seltensten (Minority-Wins). Wie Abbbildung 4.1 zeigt, führen diese Strategien zu einer Intensivierung bzw. Verblassung der Textelemente.

(a) 2436 × 3248 Pixel, Skalierungsfaktor = 2
(b) 1218 × 1624 Pixel, Skalierungsfaktor = 4
(c) 609 × 812 Pixel, Skalierungsfaktor = 8
(d) 348 × 464 Pixel, Skalierungsfaktor = 14
(e) 2436 × 3248 Pixel, Skalierungsfaktor = 2
(f) 1218 × 1624 Pixel, Skalierungsfaktor = 4
(g) 609 × 812 Pixel, Skalierungsfaktor = 8
(h) 348 × 464 Pixel, Skalierungsfaktor = 14

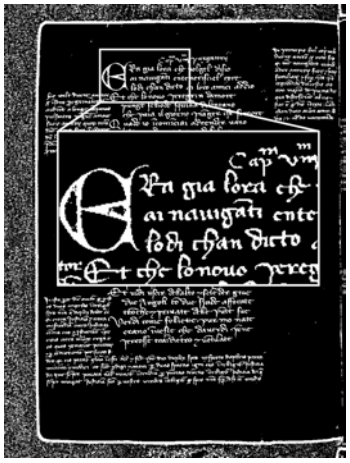
Abbildung 3.8: Zeigt die Verkleinerungsstrategien Majority-Wins (a, b, c und d) und Minority-Wins (e, f, g und h) mit verschiedenen Skalierungsfaktoren.

Blur-And-Binarize

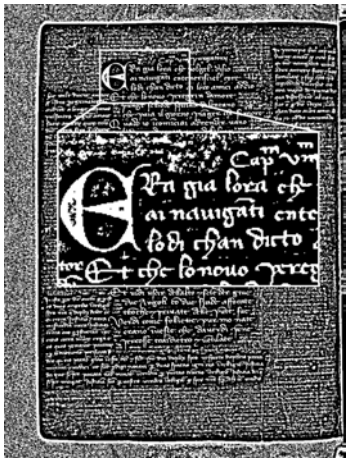
Blur-And-Binarize skaliert den pixelbasierten Ground-Truth in drei Schritten. Zuerst wird das binäre Ground-Truth-Bild mit dem Gauss'schen Weichzeichner zu einem Graustufenbild verschmiert, anschliessend mit bikubischer Interpolation skaliert und abschliessend wieder binarisiert. Um die Validität dieser Strategie zu testen, vergleichen wir das Vorgehen mit zwei Alternativen: Bikubische Skalierung des Originalbilds mit anschliessender Binarisierung und bikubische Skalierung des pixelbasierten Ground-Truths (ohne Weichzeichnen). Der erste Vergleich soll zeigen, ob es überhaupt Sinn macht, den pixbelbasierten Ground-Truth zu skalieren, oder ob nicht vom Originalbild ausgegangen werden soll. Währenddessen zeigt der zweite Vergleich, wie sich das Weichzeichnen auf die Skalierung des pixelbasierten Ground-Truths auswirkt.

Skalierung vom Original Die erste Strategie geht vom Originalbild aus - so, wie der pixelbasierte Ground-Truth ursprünglich erstellt wurde - und soll testen, ob die Binarisierung des Originalbilds auch nach dessen bikubischen Verkleinerung zu dem gewünschten Resultat führt. Dazu

(a) 900 × 1200 Pixel, Binarisierungsstrategie = Otsu



(b) 900 × 1200 Pixel, Binarisierungsstrategie = Sauvola



(c) 900 × 1200 Pixel, Binarisierungsstrategie = Niblack

Abbildung 3.9: Zeigt die Skalierung vom Original mit anschliessender Binarisierung nach Otsu, Sauvola und Niblack und den Standardparameter von Scikit [36].

wird das Originalbild zuerst mit der `resize()`-Funktion der Pillow-Bibliothek bikubisch interpoliert [9], wonach das Bild binarisiert wird. Binarisierungsalgorithmen, bzw. Schwellenwertsverfahren, verarbeiten Greyscale [22], RGB [18] und andere Bildformate anhand eines Schwellenwerts zu einem binären Bild, wo jeder Pixel unter dem Wert auf null bzw. schwarz und jeder Wert darüber zu eins bzw. weiss gesetzt wird. Während Otsu einen globalen Schwellenwert bestimmt [27], verfahren Sauvola und Niblack nach einem regionalen Schwellenwert [31][25]. Für letzteres verwenden wir die Standardparameter der Scikit-Bilbiothek [36]. Die generierten Bilder sind in Abbildung 3.9 ersichtlich: Während Otsus globaler Threshold zum Verblassen ganzer Wörter führt, entstehen mit Niblack deutliche Verschmutzungen im ganzen Bild. Auch mit Sauvola entstehen Artifakte, jedoch nicht so deutlich wie unter Niblack. Dabei ist anzumerken, dass Störungen ausserhalb der Textelemente eine untergeordnete Rolle spielen, weil sie durch das Übereinanderlegen mit dem vektorisierten Ground-Truth wegfallen (siehe Sektion 3.2.7). Störungen innerhalb von Textelementen hingegen - beispielsweise innherhalb des grossen Initials, welches in der Abbildung 3.9 hervorgehoben wird – richten dem Ground-Truth irreversiblen Schaden an. Damit scheint für die Skalierung vom Originalbild aus die Strategie unter Sauvola am zielführendsten. Wird das gleiche Experiment mit Weichzeichnen durchgeführt, ist das Noise-Level noch höher. Letzteres ist aus Platzgründen in keiner Abbildung ersichtlich.

Abbildung 3.10: Skalierung auf 900 mal 1200 Pixel ohne Weichzeichnen und Binarisieren.

Skalierung ohne Weichzeichnung und Binarisierung Die zweite Strategie geht vom Originalbild des Pixel-Level-Ground-Truths aus. Sie interpretiert die Textpixel als Sichtbar (`True`) und alle anderen (Background- und Border-Pixel) als unsichtbar (`False`). Ausgehend von diesem, binären Bild wird direkt skaliert. Danach wird der pixelbasierte Ground-Truth wie bei der ersten Strategie bikubisch interpoliert (bzw. verkleinert). Ein Schwellenwertsverfahren ist nicht nötig, weil das Bild bereits binär vorliegt. Wie die Abbildungen 3.10 und der direkte Vergleich mit den anderen Strategien in Abbildung 3.12 zeigen, wird das Bild durch das Verfahren zwar originalgetreu skaliert, führt jedoch hier und da zu eckigen und unnatürlich wirkenden Buchstaben.

Skalierung mit Weichzeichnung Die Skalierung mit Weichzeichung verkleinert den PixelGT in vier Schritten. Wie bei der zuletzt vorgestellten Strategie werden zuerst alle relevanten Textpixel als sichtbar (`True`) und alle anderen als unsichtbar (`False`) gesetzt. Im nächsten Schritt wird das Bild nach Gauss'schem Verfahren weichgezeichnet - das Ground-Truth-Bild liegt nun in Graustufen vor. Abschliessend wir das weichgezeichnete Bild bikubisch interpoliert und wieder binarisiert (nach Otsu, Niblack oder Sauvola). Wie in Abbildung 3.11 ersichtlich ist, führt das Weichzeichnen zu einem Verdicken der Textelemente. Je stärker weichgezeichnet wird, desto eher verschmelzen die Textelemente ineinander.

(a) 900 × 1200 Pixel, Binarisierungsstrategie = Otsu

(b) 900 × 1200 Pixel, Binarisierungsstrategie = Sauvola

(c) 900 × 1200 Pixel, Binarisierungsstrategie = Niblack

Abbildung 3.11: Zeigt die Skalierung des Pixel-Level-Ground-Truths mit Weichzeichnung nach der Strategie Otsu, Sauvola und Niblack. Mit den Standardparameter von Scikit. Weichgezeichnet wurde mit den Parametern s = 1, t = 3.

Schlussfolgerung

Betrachten wir die Vorgehensweisen mit und ohne Weichzeichnung im dirketen Vergleich, stellen wir fest, dass die Majority-Wins und Skalierung mit Weichzeichnung in diesem konkreten Fall zu ähnlichen Resultaten führt, siehe Abbildung 3.12. Welche der Strategien jedoch in der Praxis zu wählen sind, lässt sich anhand der vorliegenden Daten kaum schlissen. Die Software unterstützt deshalb alle, vorgestellten Strategien, wobei die Skalierung mit Weichzeichnen als Standardimplementierung gewählt worden ist.

(a) Min.-Wins. (b) Maj.-Wins (c) Biqubic (d) Otsu (e) Sauvola (f) Niblack

Abbildung 3.12: Vergleicht die Ausprägungen der sechs verschiedenen Skalierungs-Algorithmen in Bezug auf ein Wort. Alle Bilder haben die Gleiche Auflösung (900 × 1200 Pixel). Sie zeigen die Resultate der Skalierung mit Majority-/Minority-Wins, die bikubische Skalierung des pixelbasierten Ground-Truth und die Skalierung mit Binarisierung nach Otsu, Sauvola und Niblack respektive. (Die Strategie mit Weichzeichnung vom Original haben wir aus Platz- und Illustrationsgründen ausgelassen.)

3.2.6 Sorter

Die Textelemente des vektorisierten Ground-Truths des DIVA-HisDB sind nicht konsequent sortiert, weshalb der `Sorter` immer verwendet werden sollte, sofern die Reihenfolge der Textelemente eine Rolle spielt. Wir implementieren diese Funktion, indem die `Layout`- und `TextRegion`-Klasse beide die `__lt__()`-Grundfunktion des Pythonobjekts überschreiben. Damit stellen sie eine Schnittstelle für effizientes Sortieren (dank Python's eingebauter Sortierungsalgorithmen [37]) der Textelemente und Regionen zur Verfügung. Mit dem Sorter-Tool kann dieses Verhalten nach Wünschen aufgerufen, ergänzt und verändert werden. Der Sortierer geht Hand in Hand mit dem Grouper-Tool, siehe Sektion 3.2.4, und dem alternierenden Färber, siehe Sektion 3.2.2.

3.2.7 Layering

Der Layerer-Visitor dient dazu, die beiden Ground-Truths (GT) miteinander zu kombinieren. Er malt die gewünschten Vektorobjekte des Vektor-GT auf die Schichten des Pixel-Level-GT und fügt diese zu einem RGB-Bild zusammen. Dabei legt er den Vektor-GT als binäres Bild über die kombinierten Schichten des Pixel-GTs und behält nur Pixel, die sowohl im Vektor-GT als auch im Pixel-Level-GT sichtbar sind.

3.3 Builder

Der Builder (Englisch für Bauer:in oder Bauende:r) stellt eine Schnittstelle für die Konstruktion des Ground-Truths zur Verfügung und basiert auf dem Builder-Entwurfsmuster [13]. Er verbessert die Lesbarkeit, Komplexität, und damit auch die Sicherheit des Codes, indem er grosse Konstruktoren mit vielen, verschiedenen, optionalen Inputparametern durch eine Hilfsklasse ersetzt. Die Hilfsklasse nennt sich `Director` (bzw. Direktor) und bedient die vom Builder zur Verfügung gestellten Methoden in der erwünschten Reihenfolge. Soll ein neuer Ground-Truth gebaut werden, können Entwickler:innen ihre eigenen Direktoren schreiben. Methoden des Builders können iterativ aufgerufen werden und sind somit auch aus dieser Perspektive grossen Konstruktoren überlegen. Sollen Seitenobjekte des vektorbasierten Ground-Truths beispielsweise zuerst neu gruppiert und anschliessend sortiert werden, kann dies dank des Builder-Pattern in wenigen Zeilen Code instruiert werden, siehe Listing 3.1.

```
    ...
# Client-Code
grouper = BlockGrouper(layout_class=COMMENT)
builder.group(grouper) # group the comments
sorter = DescendingSorter()
builder.sort(sorter) # group all elements of the page
grouper = ThresholdGrouper(layout_class=COMMENT, x_th=300, y_th=300)
builder.group(grouper) # group only the comments
    ...
```

Listing 3.1: Client-Code, der in wenigen Zeilen den Comment-Text der Seite neu strukturiert. Zuerst teilt er die Comments basierend auf ihrer kleinsten x-Koordinate in Blöcke. Danach sortiert er die Seitenelemente dieser Blöcke nach ihrer y-Koordinate. Schlussendlich werden die sortierten Gruppen in kleinere Gruppen geteilt, sofern die Distanz zwischen den Elementen einen Schwellenwert überschreitet (in Pixel).

Kapitel 4

Schlussfolgerung

Zum Abschluss diskutieren wir, welche Ziele gescheitert sind, und wohin die Software in einem nächsten Schritt gehen könnte.

4.1 Kritische Sicht auf die Software

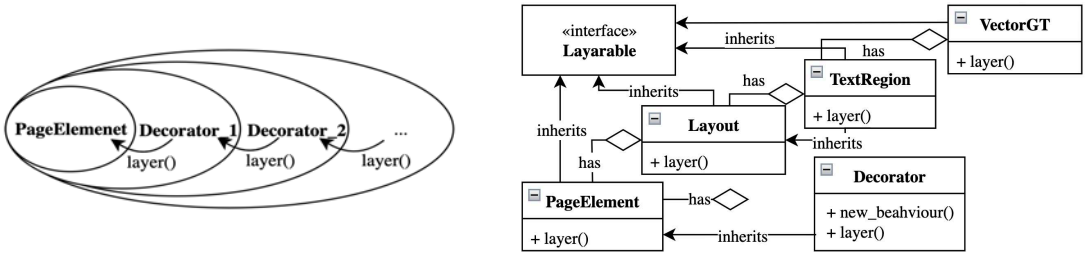
4.1.1 Funktionale Anforderungen

Die funktionalen Anforderungen sind alle implementiert worden. Der DIVA-HisDB-Datensatz kann problemlos eingelesen und gespeichert werden; er kann auf eine Wunschgrösse verkleinert und zugeschnitten werden; er kann um zusätzliche Layoutklassen erweitert werden; er kann mit den gewünschten Strategien (sowie jeder anderen erdenkbaren Strategie) eingefärbt werden; Elemente können unsichtbar geschalten, sortiert und gruppiert werden; der pixelbasierte Groundtruth und das Originalbild können in jedem von Pillow unterstützten Format [9] exportiert werden. Zusätzlich zu den genannten, funktionalen Anforderungen sind nicht-geforderte Debugging- und Illustrationsfunktionen implementiert worden (siehe Dokumention der `show()`-Funktion).

4.1.2 Nichtfunktionale Anforderungen

Wie gut die nichtfunktionalen Anforderungen - die Erweiterbarkeit/Wartbarkeit, Benutzerfreundlichkeit, Funktionalität, Übertragbarkeit, Effizient und Sicherheit - erfüllt worden sind, wird sich erst in der Praxis ergeben. Trotzdem ist es möglich, einige Aussagen dazu zu treffen. An dieser Stelle setzen wir uns kritisch mit dem Programmcode und den implementierten Entwurfsmuster auseinander.

Decorator Pattern Obwohl das Decorator-Pattern vektorbasierten Objekten erlaubt, ein beliebiges Verhalten anzunehmen, bringt dieses einen entscheidenden Nachteil. Muss ein Algorithmus die Baumstruktur einer Textseite traversieren, scheitert dieser, sobald er zwischen den verschiedenen Dekorationen unterscheiden muss. Während Verhalten wie das Gruppieren oder Sortieren nicht zwischen diesen unterscheidet, gibt es durchaus Algorithmen, die auf diese Information angewiesen sind. Ein Beispiel ist der Layering-Visitor. Er muss über alle Seitenelemente iterieren und deren Vektorobjekte auf die richtige Schicht des neuen, pixelbasierten Ground-Truths zeichnen (siehe Sektion 3.2.7). Ein Layering-Algorithmus, der nicht zwischen den verschiedenen Dekorationen unterscheiden kann, würde sie mit dem Basiselement auf die gleiche Ebene zeichnen. Dies wäre unvorteilhaft, weil daran das Kombinieren der verschiedenen Informationsschichten scheitern würde. Das Problem ist gelöst worden, indem das Seitenelement, und somit auch der Decorator, eine Schnittstelle mit dem gewünschten Verhalten erbt. Jede Subklasse von `PageElement` muss diese dann richtig implementieren, was ein gutes Verständnis der Softwarearchitektur voraussetzt. Die Wartbarkeit des Layering-Algorithmus sowie die Erweiterbarkeit der Klassenstruktur um ähnliche Algorithmen wird damit eingeschränkt.



(a) Zeigt ein dekoriertes Element mit dem `layer()`-Call.

(b) Vektor-GT Hierarchie inkl. Layerable-Interface, Composite und Deocrator-Pattern.

Abbildung 4.1: Muss über die verschiedenen Dekorationen iteriert werden, geht das beispielsweise, indem sie das neue Verhalten über eine gemeinsame Schnittstelle erben. Im Falle des Layering-Algorithmus wird die `layer()`-Funktion von der Layerable-Schnittstelle zur Verfügung gestellt. Soll die `layer()`-Funktion eines dekorierten Elements aufgerufen werden, müssen auch dessen Dekoratoren traversiert werden. Ansonsten wird das zusätzlich Verhalten der Dekoratoren nicht angwendet.

Composite-Pattern Das Composite-Pattern bietet eine intuitive Schnittstelle für das Besuchen des vektorbasierten Ground-Truth, weil nicht zwischen Components (bzw. Blätter oder Komponenten) und Composites (bzw. Kompositionen, Aggregationen) unterschieden werden muss. Dies ist jedoch nur nützlich, wenn der Klient auch keinen Anspruch hat, diese in Verbindung mit anderen Blättern zu setzen. Sobald ein:e Klient:in die Baumstruktur traversieren will, um Layouts in Verbindung zu setzen (beispielsweise um sie zu vereinen, siehe Sektion 3.2.4) muss der Klient die Baumstruktur der Textseite genau kennen. In diesen Fällen kann das Composite-Pattern schnell ein Hindernis werden, weil Änderungen an der Baumstruktur könnnen existierenden Code schnell brechen. Im Ausblick erläutern wir einen Lösungsvorschlag.

```
    ...
    for layout in page.vector_gt.regions:
        for region in layout.text_regions:
            for i in region.page_elements:
                do_something()
    ...
```

Listing 4.1: Beispielcode eines Clients (z.B. ein Visitor), der nicht zwischen Dekorationen unterscheidet.

Visitor-Pattern Obwohl das Visitor-Pattern hilft, einfach und unkompliziert neues Verhalten hinzuzufügen, sind die `Page`-Besucher eng an die bestehende Architektur gekoppelt. Bereits kleine Veränderungen können bestehenden Code in Gefahr bringen. Durch die Entkopplung der Algorithmen von den zu manipulierenden Objekten, bilden sie jedoch austauschbare Werkzeuge, die vom Builder-Pattern gebraucht werden können. Damit vereinfachen sie das Verständnis auf der Nutzungsebene (siehe nächster Paragraf).

Builder-Pattern Am Builder-Pattern ist wenig auszusetzen. Wird ein neuer Direktor implementiert, muss ein:e Nutzer:in zwar ein grundsätztliches Verständnis über die Softwararchitektur haben, insbesondere, welche Aufgaben die Visitors übernehmen, braucht aber kein Wissen über die fundamentale Klassenstruktur. Soll ein neuer Builder geschrieben werden, muss er nur die Builder-Schnittstelle erweitern. Damit bietet das Builder-Pattern eine nutzerfreundliche Top-Level Schnittstelle.

Kombination Der Kombinationsprozess ist mühselig und unverständlich. Nur solange das Schichtenprinzip des Pixel-Level-Ground-Truths, die `layer()`-Methode des Vektor-GTs und die Funktionen der `Layer`-Klasse beibehalten werden, wird der `Layerer` seine Aufgabe meistern. Damit ist die `Layerer`-Klasse unter allen Besucher am schwierigsten wart- und erweiterbar.

4.2 Ausblick

4.2.1 Technische Verbesserungen

Verbessertes Vektorobjekt

Auf atomarer Ebene sind die Vektorobjekte ausschlaggebend für die Flexibilität des Ground-Truth. Aktuell sind die vier Klassen `Polygon`, `Quadrilateral`, `Rectangle` und `Line` implementiert. Sie unterstützten fundamentale Operationen wie `get_bbox()`, `resize()`, `draw()` und `crop()`. Sobald jedoch Überschneidungen und Schnittmängen von Polygonen gefunden oder Koordinaten darauf untersucht werden sollen, ob sie innerhalb oder ausserhalb eines dieser Vektorobjekte liegen, wird die Implementation schnell komplex und fehleranfällig. Wir schlagen deshalb vor, das `VectorObject`-Modul mit Shapely, zu ergänzen - eine umfassende, gut getestete Bibliothek für die Manipulation zweidimensionaler, geometrischer Objekte [15].

Verbessertes Composite

Für die Verbesserung des Composite-Patterns präsentieren wir zwei Lösungsvorschläge. Der erste ist schnell, unkompliziert, dafür nicht besonders elegant. Der zweite bedingt eine umfassendere Restrukturierung des Programmcodes.

Bequeme Erweiterung Das Composite-Pattern, wie es vorliegt, probiert zwei Aufgaben zu lösen. Zum einen will es eine nutzerfreundliche Schnittstelle bilden, mit welcher alle Seitenelementen eines Layouts einheitlich manipuliert werden können (bspw. `resize() und crop()`). Auf der anderen Seite will es das Traversieren über alle Composites (bzw. Kontainer) einer Textseite vereinfachen (bspw. `group()`). Beides ist zwar mit der aktuellen Implementation möglich, jedoch unter eingeschränkter Verständlichkeit, Erweiterbarkeit und Wartbarkeit. Deshalb schlagen wir die Implementation der `get_all_elems()`-Funktion vor. Sie soll eine einfach zu verstehende Schnittstelle für Klient:innen bieten, die über alle Elemente einer Region iterieren wollen.

```python
class Layout:
    # Leaf
    page_elements: List[PageElement] = []
    ...
    def get_all_elems(self, elems: List[PageElement] = None) -> List[PageElement]:
        return elems.append(self.page_elems)
    ...

class Region(Layout):
    # Composite
    layouts: List[Layout] = []
    ...
    # This is the method the client will be calling.
    def get_all_elems(self, elems: List[PageElement] = None) -> List[PageElement]:
        all_elems: List[PageElement] = []
        # instanciate the list for all elements
        if elems != None:
            all_elems = elems
        # append all elements
        for layout in self.layouts:
            all_elems.append(layout.get_all_elems())
        return all_elems
    ...
```

Listing 4.2: Implementation der Funktion `get_all_elems()` in der `Layout`-Klasse.

Ein Methodenaufruf `get_all_elems()` würde eine Liste aller Seitenelemente zurückgeben, die von den Page-Visitors manipuliert werden könnten. Algorithmen, die nicht über die Struktur der Seite informiert sein müssten - konkret sind das der `Resizer`, `Cropper`, `Colorer`, `VisibilityVisitor`, `Decorator` - müssen so bloss über eine einzige Liste iterieren. Die Schwäche des Composite-Patterns bleibt jedoch für den `Sorter` und `Grouper` vorhanden, weil sie über die vorhandene Struktur informiert sein müssen. Die `get_all_elems()` löst also das fundamentale Problem nicht.

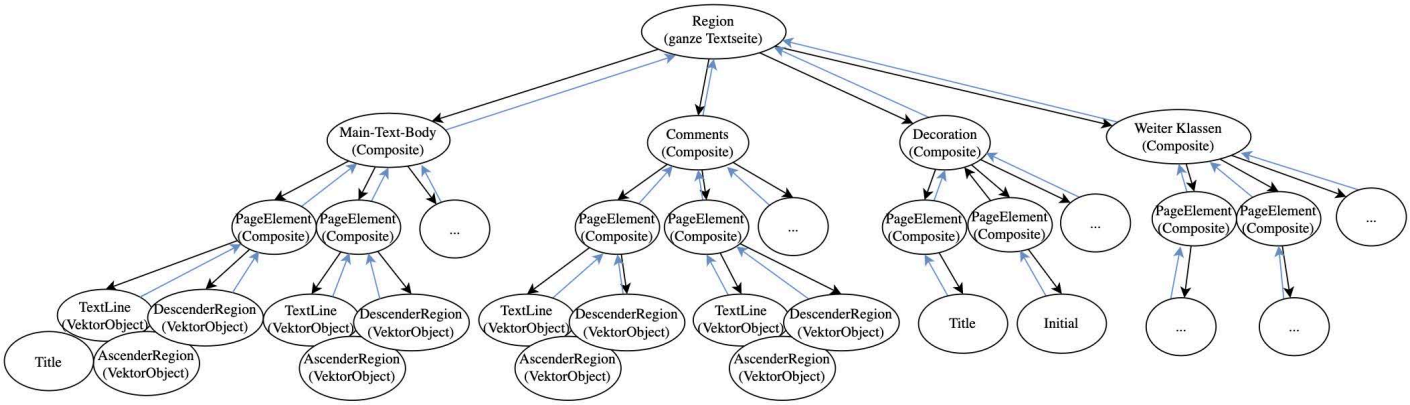


Abbildung 4.2: Baumstruktur einer Textseite, das nur aus Composites (bspw. MaintText, Comments, PageElement ect. hat) und Vektorobjekten (TextLine, DescenderRegion, Title, Initial, etc.) besteht. Das Decorator-Pattern ist überflüssig, weil neues Verhalten durch neue Blätter hinzugefügt werden kann. Die Blätter haben alle eine Referenz zu ihren Eltern, was einfaches Traversieren ermöglicht.

Verbessertes Composite Das Composite-Pattern ist ein populäres Entwurfsmuster für die Darstellung von Seitenstrukturen. So wird es beispielsweise im java.AWT.Component und javax.-swing.JComponent verwendet [3], um komplexe User-Interfaces zu gestalten. Angelehnt an deren Implementation könnte auch der Ground-Truth-Refiner implementiert werden. Analog zu den beiden Schnittstellen kann das Decorator-Pattern komplett ausgelassen werden. Stattdessen würde das Composite-Pattern mit der Schnittstelle der Superklasse `Component` alle Seitenelemente definieren und sie in einer Containerklasse `Region` speichern (siehe Abbildung 4.3). Für einfaches Traversieren kann die Superklasse, wie es auch von der Gang of Four vorgeschlagen wird [13], zusätzlich die Referenzierung von Kinderklassen zu ihren Elternklassen übernehmen. Gemeinsames Verhalten wird weiterhin über Schnittstellen vererbt (`Layarable`, `Scalable`, `Croppable`).

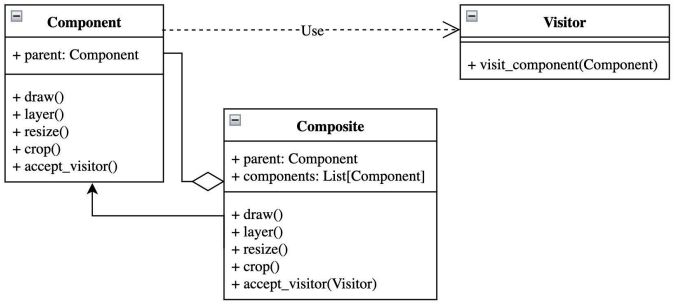


Abbildung 4.3: UML des verbesserten Composite-Pattern. Es zeigt lediglich die Superklassen und wird durch das Visitor-Pattern ergänzt, dass es ermöglich, dem Composite zusätzliches Verhalten zu geben, wenn es dieses brauchte.

Composite- und Visitor-Pattern Die ebenso vorgeschlagene Restrukturierung des Composite-Patterns nach Javas AWT- und Swing-Modul hat den Nachteil, dass jedes, neue Verhalten durch die Vererbung von Schnittstellen hinzugefügt und dann in jeder Subklasse implementiert werden muss. Während das eine durchaus praktikable Lösung ist, stellen wir an dieser Stelle noch eine Alternative vor, und zwar die Symbiose aus dem Composite- und Visitor-Pattern. Anstelle von Interfaces kann ein Komponentenbesucher zusätzliches Verhalten hinzufügen, ohne, dass eine Schnittstelle definiert werden muss. Stattdessen implementiert der `Visitor` für jedes `Component` das gewünschte Verhalten dynamisch hinzu. Ein Skalierungs-Visitor würde für jeden Komponenten eine Methode (`vistit_component()`), welches den Besucher empfängt (`accept_visitor()`). Je nach Mänge der Komponenten können die Visitor-Klassen sehr gross werden, weil für jedes eine Methode im Visitor geschrieben werden müsste. Gerade in einer dynamisch typisierten Sprache wie Python ist die

Anwenderfreundlichkeit dieses Patterns fraglich. Nichtsdestotrotz bleibt es eine gute Notlösung für das Hinzufügen von zusätzlichem Verhalten, falls das Composite-Pattern nicht verändert werden soll.

4.2.2 Weitere Funktionalitäten

Nun, dass eine erweiterbare Softwarearchitektur existiert, kann der Ground-Truth erweitert werden. Logische, nächste Schritte sind eine präszisere x-Height-Entdeckung und die genauere Klassifizierung von Textelementen.

Run Length Smoothing Algorithm Die Ascenders, Descenders und x-Height sind basierend auf einer Grundlinie (eine Gerade Schriftlinie von zwei Punkten (x1,y1),(x2,y2)) ermittelt worden. RLSA ist eine weit verbreitete Technik zur Layoutanalyse und Textsegmentation [39] und könnte dazu dienen, die Textzeilen des Ground-Truths um eine genauere x-Height zu erweitern. Der von Nikolaou vorgestellte Adaptive Run Length Smoothing Algorithm (ARSLA) [26] kann handgeschriebene, historische Dokumente in die einzelnen Textzeilen, Wörter und deren Buchstaben segmentieren. Sie ist robust gegenüber verschieden geneigten Textlinien und unterschiedlichen Schriftgrössen. Damit könnte dieses Verfahren für weit mehr als die x-Height-Entdeckung verwendet werden.

Weitere Layoutklassen Für die Weiterentwicklung des Datensatzes DIVA-HisDB, könnte die Unterteilung in weitere Layoutklassen interessant sein. Wie im letzten Paragrafen erwähnt, könnte das ARSLA-Verfahren verwendet werden, um den Ground-Truth in Wörter und Buchstaben zu segmentieren. Aber auch andere Layoutklassen könnten von Interesse sein: Initialen, Ornamente, Titel. Es könnte interessant sein, die Textelemente miteinander zu verbinden, bspw. den Titel einem Paragrafen zuzuordnen, Initialen mit der dazugehörigen Textzeile zu assoziieren, etc. Diese Kombinationsmöglichkeiten werden bereits von der Software unterstützt und müssen lediglich implementiert werden.

Kapitel 5

Besipiele

In dieser Sektion demonstrieren wir die wichtigsten Funktionalitäten der Software anhand konkreter Beispiele. Dazu stellen wir drei verschiedene Ground-Truth-Direktoren vor. Sie bedienen das zur Verfügung gestellte `GTBuilder`-Modul.

5.1 Ascenders und Descenders

Die erste Demonstration ist auf Github unter `GroundTruthRefinerDemo1.py` zu finden. Es bedient das `GTBuilder`-Modul und dessen Funktionen, um den Ground-Truth auf die Zielgrösse zu verkleineren und mit Ascenders und Descenders zu verzieren.

Als erstes laden wir die gewünschten Dateien aus dem DIVA-hisDB.

```
if __name__ == '__main__':
    # import original ground-truth
    original = Path("../../../../CB55/img/public-test/e-codices_fmb-cb-0055
    _0098v_max.jpg")
    pixel = Path("../../../../CB55/pixel-level-gt/public-test/e-codices_fmb-cb-0055
    _0098v_max.png")
    vector_gt = Path("../../../../CB55/PAGE-gt/public-test/e-codices_fmb-cb-0055
    _0098v_max.xml")
```

Im nächsten Schritt definieren wir die Sichtbarkeits- und Farbregeln zur Rerpräsentation des Ground-Truths. In diesem Fall importieren wir die Farb- und Sichtbarkeitstabelle im JSON-Format.

```
    # rules for coloring and visibility
    color_table = Path("../../../Resources/ColorTables/color_table_for_demo.json")
    visibility_table = Path("../../../Resources/VisibilityTables/visibility_table.
    json")
```

Folgendes ist ein Ausschnitt aus der importierten Farbtabelle. Sie ordnet jeder Layoutklasse eine oder mehrere Farben zu.

```
color_table: Dict = {
    "BACKGROUND": [(0, 0, 0)],
    "COMMENT": [(255, 0, 0), (244, 0, 0)],
    "DECORATION": [(0, 255, 0)],
    "MAINTEXT": [(0, 0, 255)],
    ...
    "ASCENDER": [(250, 120, 120), (20, 120, 212), (20, 212, 120), (255,20,255)],
    "DESCENDER": [(120, 120, 250), (20, 20, 20)],
}
```

Als nächstes definieren wir, wohin die neuen Ground-Truth-Files exportiert und in welcher Zieldimension sie gespeichert werden sollen.

```
    # output-directory
    out_put_directory = Path("../../../Resources/NewGTs/DemoForPresentation/")
    out_put_name: str = f"Demo-For-Presenation-{now}"
    # target dimension
```

```
    crop_dim: ImageDimension = ImageDimension(4500, 6000)
    target_dim: ImageDimension = ImageDimension(900, 1200)
```

In einem nächsten Schritt instanziieren wir den Ground-Truth-Builder `BuilderV1`. Er speichert eine Seite `Page`, die von den Besuchern `Visitor` besucht und manipuliert werden kann.

```
    # initialisation of the builder
    builder = BuilderV1(orig_img=original, px_gt_path=pixel, vector_gt_path=
    vector_gt, col_table=color_table, vis_table=visibility_table)
```

Danach schneiden wir die unnötigen Randpixel der Seite `Page` mit dem `Cropper` weg.

```
    # cropping
    cropper = Cropper(target_dim=crop_dim)
    builder.crop(cropper)
```

Als nächstes skalieren wir die Seite `Page` auf die gewünschte Grösse runter.

```
    # resizing
    resizer = Resizer(target_dim=target_dim)
    builder.resize(resizer)
```

Will ein:e Nutzer:in überprüfen, ob alle Ground-Truth-Informationen synchron verkleinert worden sind, kann dies mit folgendem Funktionsaufruf überprüft werden. Das Resultat sind die übereinandergelegten Bilder der beiden Groundtruths.

```
    # you can test if it is correctly cropped and scaled
    builder.page.vector_gt.show(builder.page.px_gt.merged_levels(all_vis=True).
    img_from_layer(rgb=True))
```

Mit dem Dekorierer fügen wir dem Ground-Truth Ascenders und Descenders hinzu. In diesem Beispiel ist willkürlich eine x-Height von zehn Pixel gewählt worden.

```
    # decorate
    decorator = AscenderDescenderDecorator(x_height=10)
    builder.decorate(decorator=decorator)
```

Die zwei folgenden Programmzeilen setzen das in der Farb- und Sichtbarkeitstabelle festgelegte Verhalten um.

```
    # set the colors and visibility
    builder.set_color()
    builder.set_visible()
```

Schlussendlich werden der vektor- und pixelbasierter Ground-Truth miteinander kombiniert.

```
    # combine the two ground-truths to one pixel-ground-truth
    layerer = Layerer()
    builder.layer(layerer=layerer)
```

Zu Debuggingzwecken kann die Seite `Page` mit der `show()`-Funktion inspiziert werden.

```
    # get the final page
    page: Page = builder.get_GT()

    # illustrate the page
    page.raw_img.show()
    page.px_gt.show()
    page.vector_gt.show()
```

Im letzten Schritt werden die neuen Grund-Truth-Informationen am gewünschten Ort gespeichert.

```
    # write the page to the output directory
    builder.write(output_path=Path(str(out_put_directory) + out_put_name))
```

Das Resultat ist in Abbildung 5.1 illustriert.

5.2 Iterierende Linien

In diesem Beispiel zeigen wir, wie Layoutklassen unterschiedlich gefärbt werden können. Dazu muss lediglich die Farbtabelle angepasst werden. Im Vergleich zum ersten Beispiel verzichten wir auf die

Ascenders und Descenders und verändern lediglich die Wahl der Farben der Layoutklassen. Für die Kommentare wählen wir drei verschiedene Farben und für den Main-Text 10 verschiedene Farben (wie im Ausschnitt aus der Farbtabelle ersichtlich ist). Das daraus resultierende Bild ist in der Abbildung 5.2 zu sehen.

```
color_table = {
    ... # other layout classes
    'COMMENT': [[177, 50, 255], [255, 50, 177], [50, 255, 177]],
    'DECORATION': [[55, 88, 200]],
    'MAINTEXT': [[255, 200, 20],[255, 200, 40],[255, 200, 60],[255, 200, 80],[255,
    200, 100],[255, 200, 120],[255, 200, 140],[255, 200, 160],[255, 200, 180],[255,
     200, 200]],
    ... # other layout classes
    }
```

5.3 Gruppieren und Sortieren

Sollen Elemente neu gruppiert werden, kann dieses Verhalten mit einem `Grouper`-Besucher hinzugefügt werden. Die Software stellt bereits zwei Gruppierer zur Verfügung. Der erste versucht die Elemente basierend auf ihrer kleinsten x-Koordinate in sinnvolle Blöcke zu unterteilen. Der zweite teilt die Elemente in neue Gruppen auf, wenn ein Threshold überschritten wird. Sie gehen Hand in Hand miteinander und führen zum besten Resultat, wenn die Seite zuerst sortiert wird. Der Source-Code ist auf Github unter `GroundTruthRefinerDemo2.py` zu finden.

Wir erweitern das Beispiel aus der letzten Sektion um die Gruppierungs- und Sortierungsfunktionalität. Für Illustrationszwecke werden nur die Kommentare gruppiert. Zusätzlich zu der in Sektion 5.1 und 5.2 beschriebenen Prozedur wenden wir den `DescenderSorter` auf alle Seitenelemente an. Damit gehen wir sicher, dass alle Elemente von oben nach unten geordnet sind.

```
    # sort all ground-truth elements based on their y-coordinate
    sorter = DescendingSorter()
    builder.sort(sorter)
```

In einem zweiten Schritt teilen wir die Kommentare basierend auf ihrer x-Koordinate in mehrere Blöcke.

```
    # make blocks base on x values
    grouper = BlockGrouper(layout_class=LayoutClasses.COMMENT)
    builder.group(grouper)
```

Mit dem `ThresholdGrouper` teilen wir die Elemente in neue Gruppen auf, sobald eine bestimmte Distanz (in diesem Fall 80 Pixel) übeschritten wird.

```
    # split regions if a given threshold is exceeded
    grouper = ThresholdGrouper(layout_class=LayoutClasses.COMMENT, x_th=80, y_th
    =80)
    builder.group(grouper)
```

Um die Textregionen bzw. den Rahmen der Regionen sichtbar zu machen, ist ein `Illustrator`-Visitor geschrieben worden. Er legt den vektorbasierten Ground-Truth über das Originalbild und zeichnet all seine Textregionen. Das Resultat ist in Abbildung 5.3 dargestellt.

```
    # illustrate the newly sorted and grouped page
    illustrator = Illustrator(background=builder.page.vector_gt.show(), color_table
    =col_table_demo_3)
    img: Image = illustrator.visit_page(builder.page)
    img.show()
```

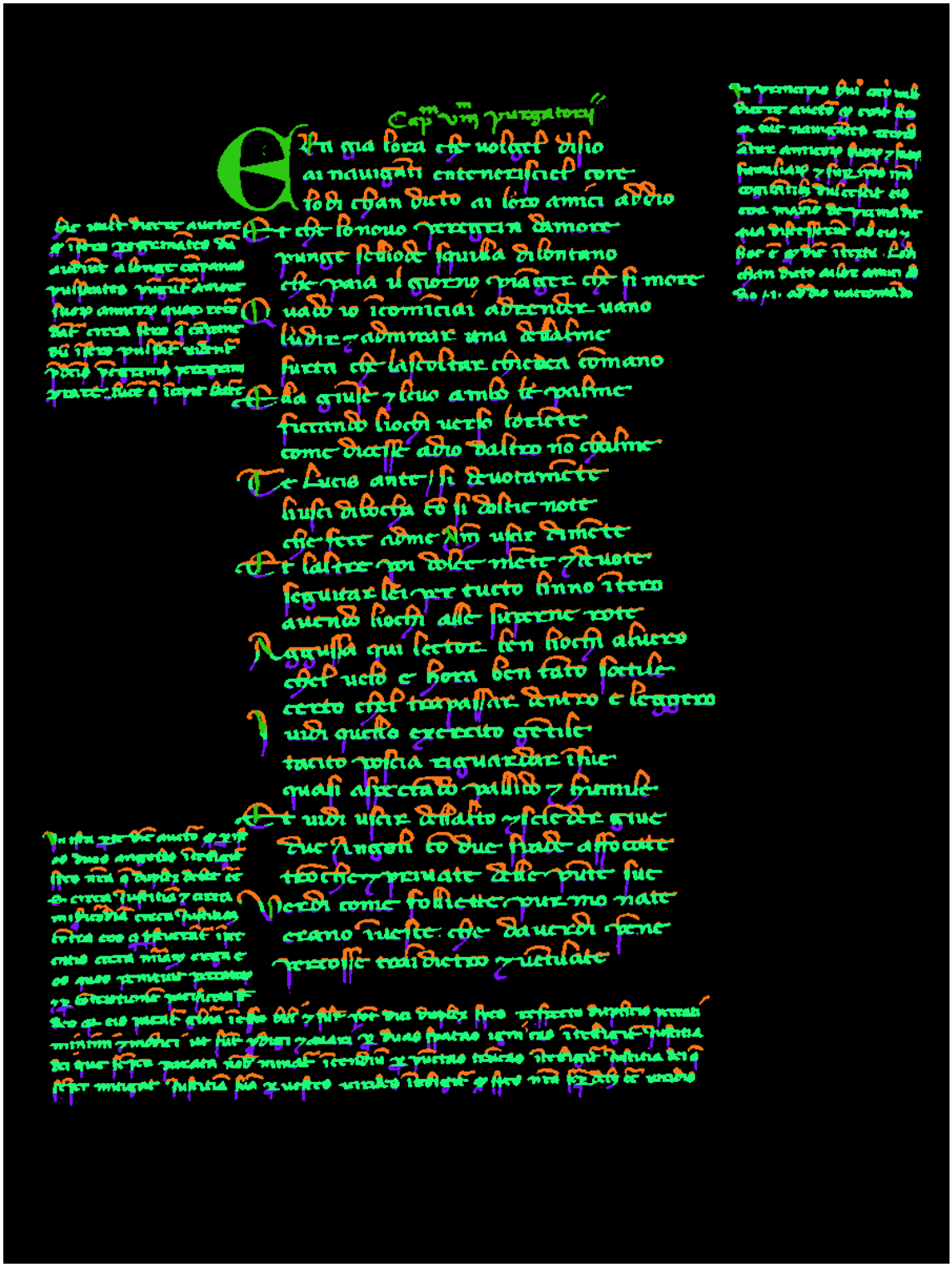


Abbildung 5.1: Ground-Truth mit gefärbten Ascenders und Descenders

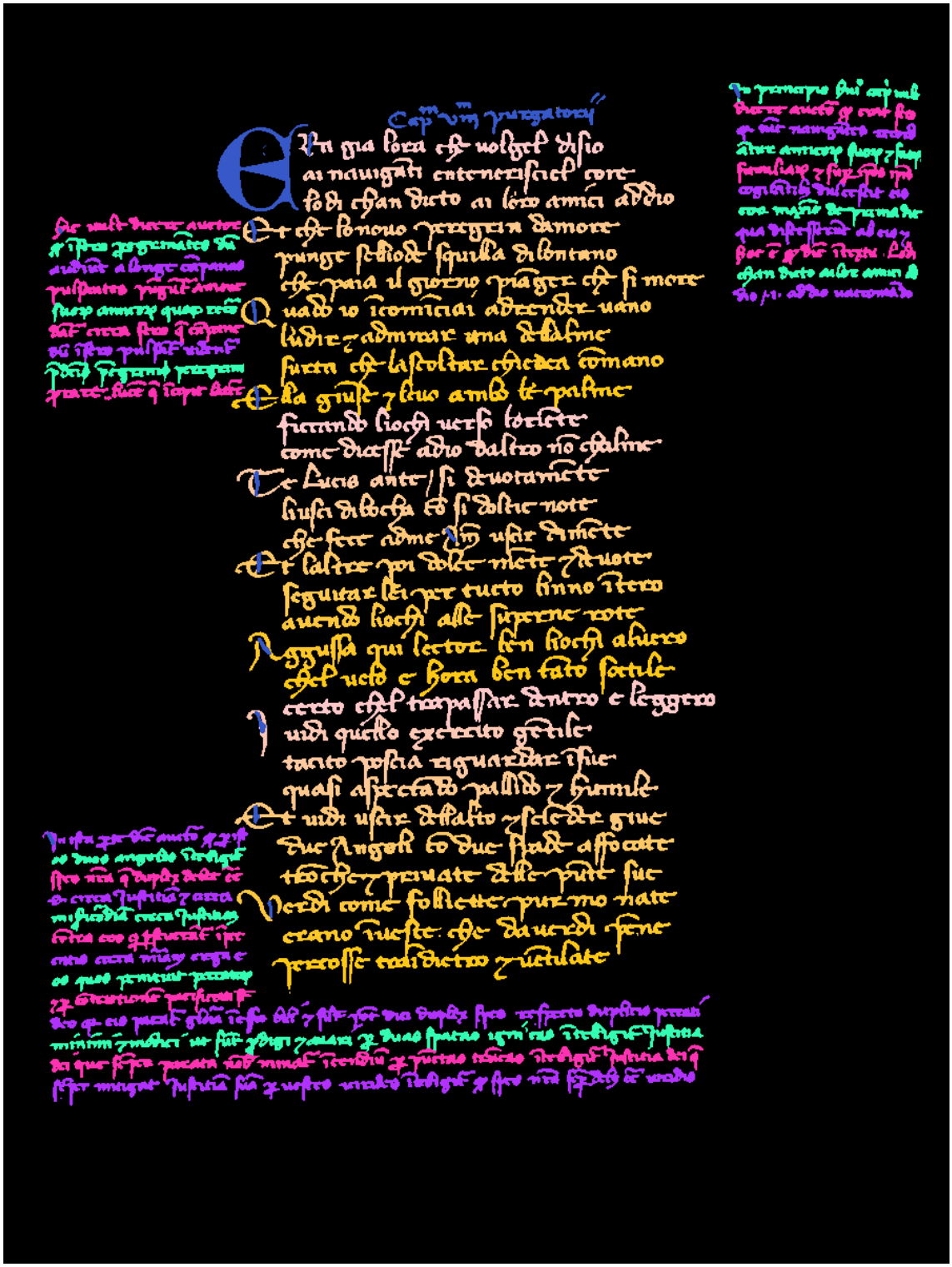


Abbildung 5.2: Ground-Truth mit iterierend gefärbten Kommentaren und Main-Text-Elementen.

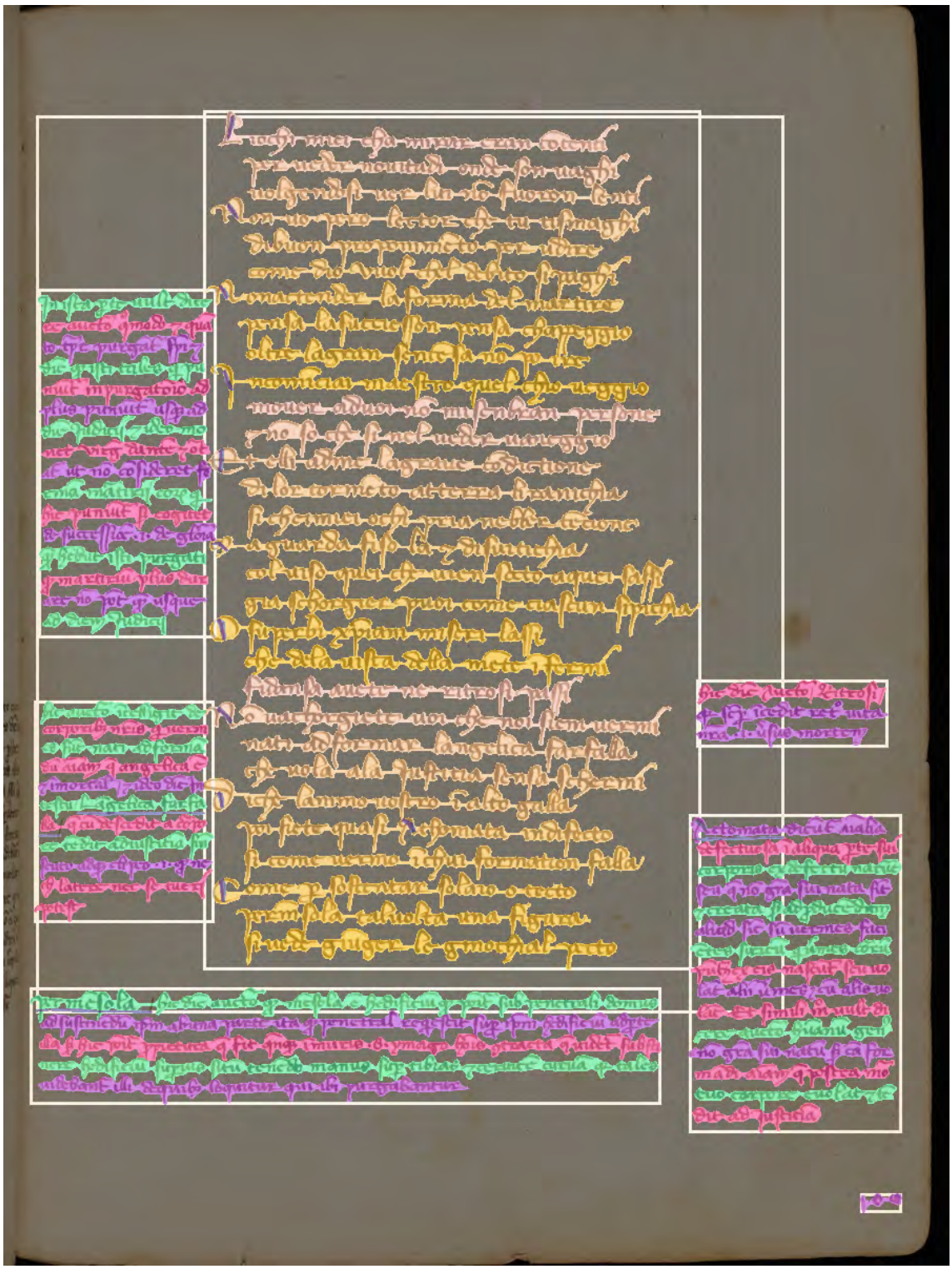


Abbildung 5.3: Zeigt ein vektorbasierter Ground-Truth, der über das Originalbild gelegt worden ist. Seine Kommentare sind mit dem `ThresholdGrouper` und `BlockGrouper` neuen Gruppen zugewiesen worden.

Acronyms

DIVA Document, Image, and Video Analysis Group

DIA Document Image Analysis

XML Extensible Markup Language

JSON JavaScript Object Notation

PNG Portable Network Graphic

JPEG Joint Photographic Experts Group

RGB Red-Green-Blue

GIF Graphics Interchange Format

RLSA Run Length Smoothing Algorithm

ARSLA Adaptive Run Length Smoothing Algorithm

Literaturverzeichnis

[1] Anaconda software distribution, 2020.

[2] Michele Alberti, Lars Vögtlin, Vinaychandran Pondenkandath, Mathias Seuret, Rolf Ingold, and Marcus Liwicki. Labeling, cutting, grouping: an efficient text line segmentation method for medieval manuscripts. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1200–1206, 2019.

[3] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[4] Ofer Biller, Abedelkadir Asi, Klara Kedem, Jihad El-Sana, and Itshak Dinstein. Webgt: An interactive web-based system for historical document ground truth generation. In *2013 12th International Conference on Document Analysis and Recognition*, pages 305–308. IEEE, 2013.

[5] Dan Bloomberg and Luc Vincent. Document image analysis (chapter 18). 2010.

[6] Jon Bosa, Tim Bray, CM Sperberg-McQueen, and James Clard. Xml (extensible markup language). 2014.

[7] Georg Brandl. Sphinx documentation. *URL http://sphinx-doc. org/sphinx. pdf*, 2021.

[8] Samuele Capobianco and Simone Marinai. Docemul: A toolkit to generate structured historical documents. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 1186–1191, 2017.

[9] Alex Clark. Pillow (pil fork) documentation, 2015.

[10] Christian Clausner, Stefan Pletschacher, and Apostolos Antonacopoulos. Aletheia-an advanced document layout and text ground-truthing system for production environments. In *2011 International Conference on Document Analysis and Recognition*, pages 48–52. IEEE, 2011.

[11] Douglas Crockford. The application/json media type for javascript object notation (json). RFC 4627, IETF, 2006.

[12] Martin Eigner, Florian Gerhardt, Torsten Gilz, and Fabrice Mogo Nem. *Informationstechnologie für Ingenieure*. Springer-Verlag, 2012.

[13] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, 1994.

[14] Angelika Garz, Mathias Seuret, Andreas Fischer, and Rolf Ingold. A user-centered segmentation method for complex historical manuscripts based on document graphs. *IEEE Transactions on Human-Machine Systems*, 47(2):181–193, 2016.

[15] Sean Gillies. Shapely: manipulation and analysis of geometric objects, 2007.

[16] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin

Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[17] Andreas Heuer and Gunter Saake. Datenbanken: Konzepte und sprache, 1995.

[18] Robert Hirsch. *Exploring colour photography: a complete guide*. Laurence King Publishing, 2004.

[19] Compuserve Incorporated. Gif graphics interchange format: A standard defining a mechanism for the storage and transmission of bitmap-based graphics information. Columbus, OH, USA, 1987.

[20] ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, 2001.

[21] ISO/IEC 25010. ISO/IEC 25010:2011, systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models, 2011.

[22] Stephen Johnson. *On Digital Photography*, volume 35. O'Reilly, 2006.

[23] Nicholas Journet, Muriel Visani, Boris Mansencal, Kieu Van-Cuong, and Antoine Billy. Doccreator: A new software for creating synthetic ground-truthed document images. *Journal of imaging*, 3(4):62, 2017.

[24] Joan Mas, Alicia Fornés, and Josep Lladós. An interactive transcription system of census records using word-spotting based information transfer. In *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*, pages 54–59. IEEE, 2016.

[25] Wayne Niblack. *An introduction to digital image processing*. Strandberg Publishing Company, 1985.

[26] Nikos Nikolaou, Michael Makridis, Basilis Gatos, Nikolaos Stamatopoulos, and Nikos Papamarkos. Segmentation of historical machine-printed documents using adaptive run length smoothing and skeleton segmentation paths. *Image and Vision Computing*, 28(4):590–604, 2010.

[27] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[28] Stefan Pletschacher and Apostolos Antonacopoulos. The page (page analysis and ground-truth elements) format framework. In *2010 20th International Conference on Pattern Recognition*, pages 257–260. IEEE, 2010.

[29] Vinaychandran Pondenkandath, Michele Alberti, Michaël Diatta, Rolf Ingold, and Marcus Liwicki. Historical document synthesis with generative adversarial networks. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, volume 5, pages 146–151, 2019.

[30] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[31] Jaakko Sauvola and Matti Pietikäinen. Adaptive document image binarization. *Pattern recognition*, 33(2):225–236, 2000.

[32] Zejiang Shen, Kaixuan Zhang, and Melissa Dell. A large dataset of historical japanese documents with complex layouts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 548–549, 2020.

[33] Foteini Simistira, Mathias Seuret, Nicole Eichenberger, Angelika Garz, Marcus Liwicki, and Rolf Ingold. Diva-hisdb: A precisely annotated large dataset of challenging medieval manuscripts. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 471–476. IEEE, 2016.

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[35] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. *arXiv preprint arXiv:1603.03417*, 2016.

[36] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[37] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[38] Lars Vögtlin, Manuel Drazyk, Vinaychandran Pondenkandath, Michele Alberti, and Rolf Ingold. Generating synthetic handwritten historical documents with ocr constrained gans. In *International Conference on Document Analysis and Recognition*, pages 610–625. Springer, 2021.

[39] Friedrich M Wahl, Kwan Y Wong, and Richard G Casey. Block segmentation and text extraction in mixed text/image documents. *Computer graphics and image processing*, 20(4):375–390, 1982.

Ground-Truth Refiner

Release 02.09.2022

Max Gregor Haller

CHAPTER

ONE

HISDB_GT_REFINEMENT

1.1 GTRefiner package

1.1.1 Subpackages

GTRefiner.Builder package

Submodules

GTRefiner.Builder.Builder module

class GTRefiner.Builder.Builder.**GTBuilder**

Bases: object

The builder provides an interface for manipulating an instance of :class: *Page* class. It serves to construct the ground truth and is based on the builder design pattern. It improves code readability, complexity, and thus safety by replacing large constructors with many, different, optional input parameters with an auxiliary class. The auxiliary class is called Director and serves the methods provided by the builder in the desired order. If a new ground truth is to be built, developers:inside can write their own directors. Methods of the Builder can be called iteratively and are therefore superior to large constructors from this perspective as well. For example, if page objects of the vector GT are to be regrouped first and then sorted, this can be instructed in a few lines of code thanks to the builder pattern, see ClientExamples: RegionIllustratorDirector.py.

abstract crop(*cropper:* Cropper)

Visit the Page with the :class: *Cropper* given. :param cropper: crop the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the cropper's implementation. :type cropper: Cropper

abstract decorate(*decorator:* TextLineDecorator)

Visit the Page with the :class: *TextLineDecorator* given. :param resizer: decorate the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the decorator's :class: *TextLineDecorator* implementation. :type resizer: TextLineDecorator

abstract get_GT() → *Page*

Get the new ground truth :return: Return the modified page :class: *Page* :rtype: Page

abstract group(*grouper:* Grouper)

Visit the Page with the :class: *Grouper* given. :param grouper: group the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the grouper's :class: *Grouper* implementation. :type grouper: Grouper

abstract illustrate(*illustrator:* Illustrator)

Visit the Page with the :class: *Illustrator* given. :param illustrator: illustrate the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the illustrator's :class: *Illustrator* implementation. :type illustrator: Illustrator

abstract layer(*layerer:* Layerer)

Visit the Page with the :class: *Layerer* given. :param layerer: combines the two ground-truths vector-gt :class: *VectorGT* and pixel-level-gt `PixelLevelGT` and stores the newly generated pixel-based ground-truth based on the Layerer's implementation in the page `Page`. :type layerer: Layerer

abstract read(*vector_gt_path: Path*, *px_gt_path: Path*, *orig_img: Path*, *vis_table: Path*, *col_table: Path*)

Constructor Method :param vector_gt_path: path to the vector ground truth :type vector_gt_path: Path :param px_gt_path: path to the pixel-level ground truth :type px_gt_path: Path :param orig_img: path to the original image ground truth :type orig_img: Path :param vis_table: path to the visibility table to be applied to the ground truth :class: *Page* :type vis_table: VisibilityTable :param col_table: path to the color table to be applied to the ground truth :class: *Page* :type col_table: ColorTable

resize(*resizer:* Resizer)

Visit the Page with the :class: *Resizer* given. :param resizer: resize the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the resizer's :class: *Resizer* implementation. :type resizer: Resizer

abstract set_color(*colorer: ~GTRefiner.BuildingTools.Visitors.Colorer.Colorer = <GTRefiner.BuildingTools.Visitors.Colorer.Colorer object>*)

Set the color of both the vector_gt objects and the vector_gt.

abstract set_visible()

abstract sort(*sorter:* Sorter)

Visit the Page with the :class: *Sorter* given. :param grouper: sort the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the sorter's :class: *Sorter* implementation. :type grouper: Sorter

abstract write(*output_path: Path*)

Visit the Page with the :class: *Writer* given. :param output_path: writes the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* to the :class: *Path* out_put_path to the format :class: *Writer*. :type output_path: Path

GTRefiner.Builder.Builder_v1 module

class GTRefiner.Builder.Builder_v1.**BuilderV1**(*vector_gt_path: Path*, *px_gt_path: Path*, *orig_img: Path*, *vis_table: Path*, *col_table: Path*)

Bases: *GTBuilder*

This class provides an example for concrete :class: *Builder*.

crop(*cropper:* Cropper)

Visit the Page with the :class: *Cropper* given. :param cropper: crop the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the cropper's implementation. :type cropper: Cropper

decorate(*decorator:* TextLineDecorator)

Visit the Page with the :class: *TextLineDecorator* given. :param resizer: decorate the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the decorator's :class: *TextLineDecorator* implementation. :type resizer: TextLineDecorator

get_GT() → *Page*

Get the new ground truth :return: Return the modified page :class: *Page* :rtype: Page

group(*grouper:* Grouper)

Visit the Page with the :class: *Grouper* given. :param grouper: group the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the grouper's :class: *Grouper* implementation. :type grouper: Grouper

illustrate(*illustrator:* Illustrator)

Visit the Page with the :class: *Illustrator* given. :param illustrator: illustrate the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the illustrator's :class: *Illustrator* implementation. :type illustrator: Illustrator

layer(*layerer:* Layerer)

Visit the Page with the :class: *Layerer* given. :param layerer: combines the two ground-truths vector-gt :class: *VectorGT* and pixel-level-gt `PixelLevelGT` and stores the newly generated pixel-based ground-truth based on the Layerer's implementation in the page `Page`. :type layerer: Layerer

read(*vector_gt_path: Path*, *px_gt_path: Path*, *orig_img: Path*, *vis_table: Path*, *col_table: Path*) → *Page*

Initialization method. Take all input paths :class: *Path* and returns a :class: *Page* :param vector_gt_path: path to the vector ground truth :type vector_gt_path: Path :param px_gt_path: path to the pixel-level ground truth :type px_gt_path: Path :param orig_img: path to the original image ground truth :type orig_img: Path :param vis_table: path to the visibility table to be applied to the ground truth :class: *Page* :type vis_table: VisibilityTable :param col_table: path to the color table to be applied to the ground truth :class: *Page* :type col_table: ColorTable :return: Returns a page :class: *Page* based on the paths :class: *Path*. :rtype: Page

resize(*resizer:* Resizer)

Visit the Page with the :class: *Resizer* given. :param resizer: resize the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the resizer's :class: *Resizer* implementation. :type resizer: Resizer

set_color(*colorer: ~GTRefiner.BuildingTools.Visitors.Colorer.Colorer =*
<GTRefiner.BuildingTools.Visitors.Colorer.Colorer object>)

Set the color of both the vector_gt objects and the vector_gt.

set_visible(*visibilitor: ~GTRefiner.BuildingTools.Visitors.VisibilityVisitor.VisibilityVisitor =*
<GTRefiner.BuildingTools.Visitors.VisibilityVisitor.VisibilityVisitor object>)

sort(*sorter:* Sorter)

Visit the Page with the :class: *Sorter* given. :param grouper: sort the pages image :class: *Image*, vector-gt :class: *VectorGT* and pixel-level-gt :class: *PixelLevelGT* according to the sorter's :class: *Sorter* implementation. :type grouper: Sorter

write(*output_path*)

Write :class: *Page* Page as JSON and GIF. :param output_path: :type output_path: :return: :rtype:

GTRefiner.Builder.Director module

class GTRefiner.Builder.Director.**Director**

Bases: object

abstract make()

class GTRefiner.Builder.Director.**Factory**(*builder: <module 'GTRefiner.Builder.Builder' from '/Users/loverboy99/PycharmProjects/BachelorThesis/HisDB_GT_Refinement/GTI*
color_table: ~GTRefiner.GTRepresentation.Table.ColorTable,
vis_table: ~GTRefiner.GTRepresentation.Table.VisibilityTable,
root_directory: ~pathlib.Path)

Bases: *Director*

Calls the given Builder on all the GT_files in the root directory.

class GTRefiner.Builder.Director.**Prototyper**(*builder: <module 'GTRefiner.Builder.Builder' from '/Users/loverboy99/PycharmProjects/BachelorThesis/HisDB_GT_Refinement/*
color_table:
~GTRefiner.GTRepresentation.Table.ColorTable, *vis_table:*
~GTRefiner.GTRepresentation.Table.VisibilityTable,
vector_gt: ~pathlib.Path, *px_gt: ~pathlib.Path*, *ori_img:*
~pathlib.Path)

Bases: *Director*

Creates prototypes of the given ColorTable, VisibilityTable, Paths, and Builder and shows it. Once pleased, the client can should use the factory :class: *Factory* class.

Module contents

GTRefiner.BuildingTools package

Subpackages

GTRefiner.BuildingTools.Visitors package

Submodules

GTRefiner.BuildingTools.Visitors.Colorer module

class GTRefiner.BuildingTools.Visitors.Colorer.**Colorer**(*color_table: Optional[*ColorTable*] = None*)

Bases: *Visitor*

Default implementation of Colorer sets the colors of the vector objects and the different levels of the pixel ground truth. Using a color table in c{JSON} format, a client can assign individual colors to the various vector objects of the VectorGT or layers of the PixelGT. The colorer takes the responsibility of reading this information and assigning it to the target elements. It supports three strategies: paint each TextElement of a layout class the same, paint each TextElement of a layout class differently, or give each a different color. In principle, any other coloring strategy is conceivable. To do this, the colors in the Color palette must be selected accordingly. If specifics groups should have a certain color (or iteration of colors), a new Colorer can be written, to introduce the desired rules. :param color_table: colors of color table are used to define the colors of the differenet page elements `PageElement`, layouts `Layout` and regions, `Region`. :type color_table: ColorTable

visit_page(*page:* Page)

Set the colors of the ground truth information from page. Supported strategies are alternating, all the same, all different or any other combination of colors given in the color table. This Colorer just iterates over the colors given. :param page: Is going to be colored according to the color table from the instance of self :type page: Page

GTRefiner.BuildingTools.Visitors.Cropper module

class GTRefiner.BuildingTools.Visitors.Cropper.**Cropper**(*target_dim:* ImageDimension)

Bases: *Visitor*

The Cropping Visitor crops the image to a desired dimension. It is possible to specify whether the page is left- or right-bound. The current use of the cropping function is to get rid of useless edge pixels, but it could also be used for illustration purposes or targeted cropping of text regions. The behavior is inherited through the Croppable interface.

visit_page(*page:* Page)

Default implementation of Cropper :param page: Crops all elements of page :class: *Page* to a given target dimension, where a page can be cut left or cut right. This is figured out algorithmically. :type page: Page

GTRefiner.BuildingTools.Visitors.Grouper module

class GTRefiner.BuildingTools.Visitors.Grouper.**BlockGrouper**(*layout_class:* LayoutClasses)

Bases: *Grouper*

Groups the elements of the :class: *Layout* given into different blocks depending on their minimal x-coordinate. Takes use of the np.histogram to group into different bins. :param layout_class: Layout-class to be grouped. :type layout_class: LayoutClasses

group(*region:* TextRegion, *bins: int = 6*) → *TextRegion*

Detect clusters based on x value of page elements. :param region: region to be sorted. :type region: TextRegion :param bins: number of x-oriented bins, defaults to 6 :type bins: int :return: returns the given amount of bins (or less) see numpy documentation :rtype: List[PageElement]

visit_page(*page:* Page)

Groups all the elements of the layout-class :class: *LayoutClasses* of this instance.

class GTRefiner.BuildingTools.Visitors.Grouper.**Grouper**

Bases: *Visitor*

The Grouping tool works at the level of text regions. It creates, divides and combines the text elements into logical (sub)groups depending on the algorithm. Currently, the Grouper module supports clustering text elements based on their smallest x-coordinate (Blockgrouper) and subdividing them into blocks of close, adjacent text elements (Textgrouper).

abstract group(*region:* TextRegion) → *TextRegion*

Groups all layouts class: *Layout* within a region class: *TextRegion*. :param region: Region to be (re-)grouped :type region: TextRegion :return: grouped Region :rtype: TextRegion

abstract visit_page(*page:* Page)

No default implementation available for the Grouper.

class GTRefiner.BuildingTools.Visitors.Grouper.**ThresholdGrouper**(*x_threshold: int*, *y_threshold: int*, *layout_class:* LayoutClasses)

Bases: *Grouper*

Threshold grouper splits regions if their elements are too far apart (only if both the x and y threshold are exceeded). :param x_threshold: x threshold in pixels to define determin if an element should be split off. :type x_threshold: int :param y_threshold: y threshold in pixels to define determin if an element should be split off. :type y_threshold: int :param layout_class: Layout-class to be grouped. :type layout_class: LayoutClasses

group(*region:* TextRegion) → *TextRegion*

Splits regions if their elements are too far apart (only if both the x and y threshold are exceeded). Warning: Make sure regions are sorted in either ascending or descending order the way a text is read. :param region: region to be grouped :type region: TextRegion

visit_page(*page:* Page)

Groups all the elements of the layout-class :class: *LayoutClasses* of this instance. :param page: page to be grouped :type page: Page

GTRefiner.BuildingTools.Visitors.IllustratorVisitor module

class GTRefiner.BuildingTools.Visitors.IllustratorVisitor.**Illustrator**(*background: <module 'PIL.Image' from '/opt/miniconda3/envs/BachelorThesis/lib/p packages/PIL/Image.py'> = None*, *color_table: ~GTRefiner.GTRepresentation.Table.ColorTable = None*, *vis_table: ~GTRefiner.GTRepresentation.Table.VisibilityTabl = None*, *outline: ~typing.Tuple = None*)

Bases: *Visitor*

Illustrator serves for visualizing processes. :param background: If you want the vector gt to be drawn on a background, specify the image :type background: Image :param color_table: If you want another color table than the quick and dirty specified in this class :type color_table: ColorTable :param vis_table: If you want another vis table than the quick and dirty specified in this class :type vis_table: VisibilityTable :param outline: Specify the outline color here, if None is given no outline will be drawn. :type outline: tuple

color_table = <GTRefiner.GTRepresentation.Table.ColorTable object>

comment_color = [(255, 20, 255)]

decoration_color = [(20, 255, 255)]

main_text_color = [(10, 20, 255)]

vis_table = <GTRefiner.GTRepresentation.Table.VisibilityTable object>

visit_page(*page:* Page)

Illustrate the page. :param page: page to illustrate :type page: Page :return: If a background is given its going to be blended. :rtype: Image

GTRefiner.BuildingTools.Visitors.Layerer module

class GTRefiner.BuildingTools.Visitors.Layerer.**Layerer**

Bases: *Visitor*

The Layerer Visitor is used to combine the two ground truths (vector gt and pixel-based gt). It paints the desired vector objects of the vector GT onto the layers of the pixel-level GT and combines them to form an RGB image. In doing so, it overlays the vector GT as a binary image on top of the combined layers of the pixel GT, keeping only pixels that are visible in both the vector GT and the pixel-level GT.

classmethod visit_page(*page:* Page)

The default implementation of Combiner combines the vector gt and pixel gt by drawing the vector objects on the according layer of the levels within the pixel gt. It takes use of the Layarable :class: *Layerable* interface. :param page: :type page: :return: :rtype:

GTRefiner.BuildingTools.Visitors.Resizer module

class GTRefiner.BuildingTools.Visitors.Resizer.**Resizer**(*target_dim:* ImageDimension)

Bases: *Visitor*

Resize a page (and all it's ground-truth information, including the original image) to a target dimension. The default implementation scales the PixelGT in four steps. As in the last presented strategy, first all relevant text pixels are set as visible True and all others as invisible False. In the next step, the image is blurred using Gaussian methods - the ground truth image is now in grayscale. Finally, the blurred image is bicubically interpolated and binarized again (according to Otsu, Niblack or Sauvola). Blurring leads to a thickening of the text elements. The more blurring is applied, the more the text elements merge into each other. :param target_dim: Target dimension :type target_dim: ImageDimension

visit_page(*page:* Page)

Resize a page (and all it's ground-truth information, including the original image) to a target dimension. :param target_dim: Target dimension :type target_dim: ImageDimension

GTRefiner.BuildingTools.Visitors.Sorter module

class GTRefiner.BuildingTools.Visitors.Sorter.**AscendingSorter**

Bases: *Sorter*

visit_page(*page:* Page)

Sort all elems of a region in ascending order (ascending = lowest y value first). :param page: page to be sorted. :type page: Page

class GTRefiner.BuildingTools.Visitors.Sorter.**DescendingSorter**

Bases: *Sorter*

visit_page(*page:* Page)

Sort all elems of a region in descending order (descending = highest y value first). :param page: page to be sorted. :type page: Page

class GTRefiner.BuildingTools.Visitors.Sorter.**Sorter**

Bases: *Visitor*

Sort a given container of objects. The text elements of the vectorized ground truth of the DIVA-HisDB are not consistently sorted, which is why the Sorter should always be used if the order of the text elements matters. We implement this function by having the layout and TextRegion classes both override __lt__() base-function of the

Python object. Thus they provide an interface for efficient sorting (thanks to Python's built-in sorting algorithms) of text elements and regions. The sorter tool can be used to invoke, add to, and modify this behavior as desired. The sorter goes hand in hand with the grouper tool, see module Grouper, and the alternating colorer, see module Colorer.

abstract visit_page(*page:* Page)

Visit the page and apply the new behaviour of the concrete implementation of this Visitor :class: *Visitor*.

GTRefiner.BuildingTools.Visitors.TextLineDecorator module

class GTRefiner.BuildingTools.Visitors.TextLineDecorator.**AscenderDescenderDecorator**(*x_height: int*)

Bases: *TextLineDecorator*

Parameters
x_height (*int*) – Based on this int value and a baseline provided by the TextLine element calculate Ascenders, Descenders and x-Height (Rectangles).

visit_page(*page:* Page)

Decorate all TextLine instaces elements of page.

class GTRefiner.BuildingTools.Visitors.TextLineDecorator.**HeadAndTailDecorator**

Bases: *TextLineDecorator*

"Example of another Decorator Class"

class GTRefiner.BuildingTools.Visitors.TextLineDecorator.**HistogramDecorator**

Bases: *TextLineDecorator*

"Example of another Decorator Class"

class GTRefiner.BuildingTools.Visitors.TextLineDecorator.**TextLineDecorator**

Bases: *Visitor*

abstract classmethod visit_page(*page:* Page)

Decorate textline elements of page.

GTRefiner.BuildingTools.Visitors.VisibilityVisitor module

class GTRefiner.BuildingTools.Visitors.VisibilityVisitor.**VisibilityVisitor**(*vis_table: Optional[*VisibilityTable*] = None*)

Bases: *Visitor*

Based on a visibility table, set all elements in the vector ground-truth `VectorGT` and all layers of the pixel level ground-truth:class:*PixelLevelGT* to the specified boolean value. Analogous to the Colorer, the Visibility-Visitor reads a visibility table that defines whether a layout class should be visible or not. If the user decides that only individual text regions or text elements are of interest, a new visitor can be written that implements the desired functionality. :param vis_table: visibility table. :type vis_table: VisibilityTable

visit_page(*page:* Page)

Based on a visibility table, set all elements in the vector ground-truth `VectorGT` and all layers of the pixel level ground-truth:class:*PixelLevelGT* to the specified boolean value. :param page: Page that should be set visible according to the visibility table provided within the page :class: *Page* or can be set Visible with a custom visibility table provided by the instance (at instantiation). :type page: Page

Module contents

Submodules

GTRefiner.BuildingTools.Visitor module

class GTRefiner.BuildingTools.Visitor.**Visitor**

Bases: object

Adds external behaviour to the page class.

abstract visit_page(*page:* Page)

Visit the page and apply the new behaviour of the concrete implementation of this Visitor :class: *Visitor*.

Module contents

GTRefiner.GTRepresentation package

Subpackages

GTRefiner.GTRepresentation.Interfaces package

Submodules

GTRefiner.GTRepresentation.Interfaces.GTInterfaces module

class GTRefiner.GTRepresentation.Interfaces.GTInterfaces.**Croppable**

Bases: object

abstract crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Introduces the behaviour of a croppable object. Based on a current dimension and a target dimension the object implementing this behaviour should be able to be cropped to the target dimension. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

class GTRefiner.GTRepresentation.Interfaces.GTInterfaces.**Dictionable**

Bases: object

abstract build() → Dict

Introduces the behaviour of a JSON compliant represenation. The returned dictionary is supposed to be stored in a JSON file. :return: Dictionary of all the vector objects of interest. :rtype: dict

class GTRefiner.GTRepresentation.Interfaces.GTInterfaces.**Drawable**

Bases: object

abstract draw(*drawer: ImageDraw*, *color: Optional[Tuple] = None*, *outline=None*)

Draw the vector object or collection of vector_objects. Useful for drawing on :class: *Layer*, for debugging purposes, and used by the :class: *Showable* interface.

class GTRefiner.GTRepresentation.Interfaces.GTInterfaces.**Scalable**

Bases: object

abstract resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Introduces the behaviour of a scalable object. Based on a current dimension and a target dimension the object implementing this behaviour should be able to scale to the target dimension. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

class GTRefiner.GTRepresentation.Interfaces.GTInterfaces.**Showable**

Bases: object

abstract show()

Draw the vector object or collection of vector_objects and show them on an Pillow Image. Useful to debug and make sure that the program is doing what it's supposed to. Closely related to the :class: *Showable* interface.

GTRefiner.GTRepresentation.Interfaces.Layarable module

class GTRefiner.GTRepresentation.Interfaces.Layarable.**Layarable**

Bases: object

abstract layer(***kwargs*)

Translates the vector_gt information into pixel information and store it on the corresponding :class: *LayoutClasses* layer.

Module contents

GTRefiner.GTRepresentation.PixelGTRepresentation package

Submodules

GTRefiner.GTRepresentation.PixelGTRepresentation.Layer module

class GTRefiner.GTRepresentation.PixelGTRepresentation.Layer.**Layer**(*layer: Optional[ndarray] = None*, *img_dim: Optional[*ImageDimension*] = None*, *color=None*)

Bases: object

The :class: *Layer* class is used to represent the different levels of the :class: *PixelLevelGT* class. It does not implement the :class: *Scalable* interface because Layer should only be instantiated once the resizing has been done. :param layer: binary array representing the layer :type layer: numpy.ndarayy :param img_dim: the size of the layer (static cannot be changed) :type img_dim: ImageDimension :param color: color of the layer :type color: tuple

classmethod bin_layer_from_rgb(*img: Image*) → *Layer*

Returns a binary layer where every pixel equal to (0,0,0) is set to '0', every other is set to '1'. :param img: Image to be turned into a binary layer where (0,0,0) is set to '0', every other is set to '1' :type img: Image :return: A layer corresponding to the binarized image where all black pixels (0,0,0) are set to 0 and the all the others are set to 1. :rtype: Layer

draw(*page_elem:* Drawable)

img_from_layer(*rgb: bool = False*) → Image

Returns an Image, either binary or in RGB-mode, from the layer. :param rgb: If True this class returns an RGB Pillow Image, defaults to binary :return: Return a binary image from it's mask by default

intersect(*other:* Layer) → *Layer*

XOR of each pixel of two layers. Keeps the color of the current layer. :param other: Other Layer. :type other: Layer :return: A new Layer. :rtype: Layer

intersect_this_layer_with_an_rgb_img(*img: Image*) → Image

Perform logical and of this layer with another image. For all pixels of this binary layer that are "True", keep the RGB pixel of the Image given. :param img: Image to be masked with this layer, for all pixels of this binary layer that are "True", keep the RGB pixel of the image given. :type img: Image :return: masked image. :rtype: Returns Image that has kept all its pixel wehre self.layer is "True" all others are set to (0,0,0) (black)

classmethod merge_and_draw(*layers: List[*Layer*]*, *img: Image = None*) → Image

Merges a list of layers onto a Pillow :class: *Image* Image while keeping their :param color: attribute. If only one layer is given, it will return it as a RGB Image. :param layers: The first layer in the list must be the base-layer. If only one layer is given. :type layers: List[Layer]The base-layer contains the base-object, the object that should be represented on the final Image. :param img: When given, it will draw the merged layers on this image. :type img: Image :return Image: return a Pillow :class: *Image* Image with moder RGB

paint_layer_on_img(*img: Image*, *color=None*) → Image

Takes a base_img of mode RGB and overlays it with the layer of the current instance. Pixels corresponding to 0 are set to black (0,0,0). Pixels corresponding to 1 are kept. :param img: Image to be masked. :return: Masked Image with (0,0,0) where layer is 0.

paint_layer_on_img_and_keep_colors(*img: Image*, *color: Tuple = None*)

Overlay img with colored layer. The values > 0 from self.layer will overwrite the img, the rest of the pixels are kept as are. Make sure that the layer has a color. Color of the layer can be set with set_color() or the color param. :param img: img for the layer to be painted on. :type img: Image :param color: color in which the layer should be painted on the Image :type color: Image :return: :rtype:

set_color(*color: Tuple*)

Set color of the layer. :param color: Set color of the layer. :type color: tuple

set_visible(*is_visible: bool*)

Set visibility of the layer. :param is_visible: Set visibility of the layer. :type is_visible: bool

show()

Display the image

unite(*other:* Layer) → *Layer*

Logical OR of each pixel of two layers. Keeps the color of the current layer. :param other: Other Layer. :type other: Layer :return: A new Layer. :rtype: Layer

GTRefiner.GTRepresentation.PixelGTRepresentation.PixelGT module

class GTRefiner.GTRepresentation.PixelGTRepresentation.PixelGT.**MyImage**(*img: <module 'PIL.Image' from '/opt/miniconda3/envs/BachelorThesis/lib/packages/PIL/Image.py'>*)

Bases: *GroundTruth*

Wraper class for the pillow image and functionalities to manipulate it. :param img: ground truth image :type img: Image

binarize(*img: <module 'PIL.Image' from '/opt/miniconda3/envs/BachelorThesis/lib/python3.8/site-packages/PIL/Image.py'> = None*, *bin_algo: str = 'otsu'*, ***kwargs*) → array

Binarize the given image to based on either otsu, sauvola or niblack. :param img: img to be binarized :type img: Image to binarize, defaults to self.img :param bin_algo: either otsu, sauvola or niblack. Otsu binarizes base on a global threshhold, sauvola and niblack on a local threshold (based on a window size). :type bin_algo: str :param kwargs: if other than default parameters of sauvola or niblack should be used (see scikit documentation) :type kwargs: Any :return: returns a numpy array :rtype: ndarray

crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Crops the image of this instance to a target dimension. Cuts the off the black part on either the right or left side. Designed to be used before resizing ground truth. Suggested dimension is ImageDimension(4500, 6000). :param current_dim: Not used in this method :type current_dim: ImageDimension :param target_dim: Target dimension the cropped image should have :type target_dim: ImageDimension :param cut_left: Whether or not the left or right part of the image should be cut off. :type cut_left: bool

resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Bicubic scaling of the ground-truth image based on a current dimension and a target dimension. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

show()

Show the image of the raw img (debugging purposes).

class GTRefiner.GTRepresentation.PixelGTRepresentation.PixelGT.**PixelLevelGT**(*img: <module 'PIL.Image' from '/opt/miniconda3/envs/BachelorThe packages/PIL/Image.py'> = None*, *img_dim: ~GTRefiner.GTRepresentation.ImageDime = None*)

Bases: *MyImage*

crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Crops the image of this instance to a target dimension. Cuts the off the black part on either the right or left side. Designed to be used before resizing ground truth. Suggested dimension is ImageDimension(4500, 6000). :param current_dim: Not used in this method :type current_dim: ImageDimension :param target_dim: Target dimension the cropped image should have :type target_dim: ImageDimension :param cut_left: Whether or not the left or right part of the image should be cut off. :type cut_left: bool

get_layer(*layout_class:* LayoutClasses) → *Layer*

Returns the layer with the given layout_class.

merged_levels(*visibility_table: Optional[*VisibilityTable*] = None*, *all_vis: bool = False*) → *Layer*

Only using this method will the client get the right pixel-gt. :param visibility_table: If a only certain layers are of interest they can be modified with a custom visibility table. :type visibility_table: VisibilityTable :param all_vis: given this parameter, it merges all layers. :type all_vis: bool :return: returns a binary layer :rtype: Layer

resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Resizes the pixel ground-truths image and all levels (layers) of the instance in four steps. First, it blurs the binary image. Then it uses the bicubic scaling algorithm of Pillow (provided by the super-class RawImage), followed by binarizing it. Same procedure is applied to every layer. Is supposed to be applied exactly once.

:param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

set_color(*color_table:* ColorTable)

Setter-method for the color.

set_visible(*vis_table:* VisibilityTable)

Setter-method for the is_visible field.

show()

Sho all the merged layers of the pixel_gt.

class GTRefiner.GTRepresentation.PixelGTRepresentation.PixelGT.**RawImage**(*img: <module 'PIL.Image' from '/opt/miniconda3/envs/BachelorThesis/lib packages/PIL/Image.py'>*)

Bases: *MyImage*

Raw image of the dataset.

get_cut_side() → bool

Returns the orientation of the Page. If the page is left-oriented return true, if it's right-oriented return false. :return: bool

Module contents

GTRefiner.GTRepresentation.VectorGTRepresentation package

Submodules

GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements module

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**AscenderDescenderRegion**(*text_line:* TextLine, *x_height: int*)

Bases: *TextLineDecoration*

Decorator of Textline which adds an x-region, ascender- and descender-region. :param text_line: Textline to be decorated. :param top_line: Represents the top line of x_region. :type top_line: Line :param x_region: Denotes the region for little x. :type x_region: Quadrilateral :param ascender_region: Denotes the region of ascenders. :type ascender_region: Quadrilateral :param descender_region: Denotes the region of descenders. :type descender_region: Quadrilateral The rest of the parameters are used to give custom colors to each of the `Polygon` vectorobjects for debugging purposes the different fields have.

build() → Dict

Helper function to build the json file.

crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Crop the page element to a target dimension. Due to the nature of the ground truth document :param cut_left: must be provided. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension :param cut_left: Whether or not the page is cut_left or not. :type cut_left: bool

draw(*drawer: ImageDraw*, *color: Tuple = None*, *outline=None*)

Draw the page element :class: *PageElement* polygon with the instance color if no other color is given. :param drawer: image to be drawn upon. :type drawer: ImageDraw.ImageDraw :param color: fill of the object. Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type color: tuple :param outline: outline of the object. Should only be used for illustration purposes! Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type outline: tuple

layer(*img: Image*, *layers: List[*Layer*] = None*)

Returns a list of layers for every VectorObject of a PageElement according to the order defined chosen with the implementation. (We suggest, you change it to a dict of LayoutClasses and Layers for a safer implementation.) :param img: base img to be drawn upon. :type img: Image :param layers: layers of vector objects each drawn on a layer. This PageElement will be drawn on a new layer and added to the list. :type layers: List[Layer] :return: layers: layers of vectorobjects (including this instance) each drawn on a layer. :rtype:layers: List[Layer]

resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Resizes the current :class: *PageElement* to a given target dimension. As this class doesn't possess a image dimension parameter, both the current dimension (of the page) and the target dimension (the size to be scaled to) should be given. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

set_color(*color: Optional[Tuple] = None*, *color_table: Optional[*ColorTable*] = None*)

Setter-method for the color. :param color: What color to use when drawn. :type color: Tuple

set_visible(*is_visible: Optional[bool] = None*, *vis_table: Optional[*VisibilityTable*] = None*)

Setter-method for the is_visible field. :param is_visible: Whether or not is should be recognized as visible :type is_visible: bool

show()

Displays the polygon of the PageElement for debugging purposes.

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**AscenderRegion**(*polygon:* Polygon, *top_line:* TopLine, *id: int = 0*)

Bases: *TextLineElements*

Region to represent ascenders. It is built based on a top_line and the polygon of the text_line :param polygon: Textline polygon. :type polygon: Polygon :param top_line: Top line (base line - some y value representing the x-height) :type top_line: TopLine :param id: To identify the region. Helpful for coloring, sorting, etc. Not mandatory, default value is 0. :type id: int

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**BaseLine**(*base_line:* Line, *id: int = 0*)

Bases: *TextLineElements*

set_visible(*is_visible: Optional[bool] = None*, *vis_table: Optional[*VisibilityTable*] = None*)

Setter-method for the is_visible field. :param is_visible: Whether or not is should be recognized as visible :type is_visible: bool

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**CommentTextLine**(*polygon:* Polygon, *base_line:* BaseLine, *color: Tuple = (255, 255, 255)*, *is_visible: bool = True*, *id: int = 0*)

Bases: *TextLine*

The :class: *CommentTextLine* class has no further characteristic than it's name. It instantiates the abstract super class :class: *PageElement*. :param polygon: represents the contour of the text line. :type polygon: Polygon :param base_line: Every :class: *TextLine* class comes with a default base_line provided by the original xml_gt. :type base_line: Line :param color: color of the line, defaults to white :type color: tuple :param is_visible: set visible or invisible, defaults to true :type is_visible: bool :param id: To identify the text region. Helpful for coloring, sorting, etc. Not mandatory, default value is 0. :type id: int

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**DecorationElement**(*polygon:* Polygon, *id: int = 0*)

Bases: *PageElement*

The :class: *DecorationElement* class has no further characteristic than it's name. It instantiates the abstract super class :class: *PageElement*.

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**DescenderRegion**(*polygon:* Polygon, *base_line:* BaseLine, *id: int = 0*)

Bases: *TextLineElements*

Region to represent descenders. It is built based on a base_line and the polygon of the text_line :param polygon: Text line polygon. :type polygon: Polygon :param top_line: Base line (base line - some y value representing the x-height) :type top_line: BaseLine :param id: To identify the region. Helpful for coloring, sorting, etc. Not mandatory, default value is 0. :type id: int

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**HeadAndTailRegion**(*text_line:* TextLine)

Bases: *TextLineDecoration*

Example for another decoration. A line could be devided into a head part (with initial) and tail part.

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**MainTextLine**(*polygon:* Polygon, *base_line:* BaseLine, *color: Tuple = (255, 255, 255)*, *is_visible: bool = True*, *id: int = 0*)

Bases: *TextLine*

The :class: *MainTextLine* class has no further characteristic than it's name. It instantiates the abstract super class :class: *PageElement*. :param polygon: represents the contour of the text line. :type polygon: Polygon :param base_line: Every :class: *TextLine* class comes with a default base_line provided by the original xml_gt. :type base_line: Line :param color: color of the line, defaults to white :type color: tuple :param is_visible: set visible or invisible, defaults to true :type is_visible: bool :param id: To identify the text region. Helpful for coloring, sorting, etc. Not mandatory, default value is 0. :type id: int

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**PageElement**(*polygon:* Polygon, *color: Optional[Tuple] = None*, *is_visible: bool = False*, *id: int = 0*)

Bases: *Scalable*, *Drawable*, *Showable*, *Croppable*, *Layarable*, *Dictionable*

Super class for all page elements, such as decorations & textlines. Every PageElement holds a polygon :class: *Polygon`at its core. If further characteristics shall be added, we suggest to use the decoration pattern. :param polygon: Represents the contour of the page element. :type polygon: Polygon :param color: What color to use when drawn. :type color: Tuple :param id: To identify the text_line. Helpful for coloring, sorting, etc. Not mandatory, default value is 0. :type id: int :param is_visible: Whether or not is should be recognized as visible to the :class: `Writer*. :type is_visible: bool

build() → Dict

Helper function to build the json file.

crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Crop the page element to a target dimension. Due to the nature of the ground truth document :param cut_left: must be provided. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension :param cut_left: Whether or not the page is cut_left or not. :type cut_left: bool

draw(*drawer: ImageDraw*, *color: Tuple = None*, *outline: Tuple = None*)

Draw the page element :class: *PageElement* polygon with the instance color if no other color is given. :param drawer: image to be drawn upon. :type drawer: ImageDraw.ImageDraw :param color: fill of the object. Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type color: tuple :param outline: outline of the object. Should only be used for illustration purposes! Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type outline: tuple

get_color()

Returns the page element's color :return: the page element's color :rtype: tuple

get_id()

get_min_y()

Getter method for the minimum y coordinate of this PageElement.

is_visible() → bool

Getter-method for the is_visible field. :return: Whether or not is should be recognized as visible

layer(*img: Image*, *layers: List[*Layer*] = None*) → List[*Layer*]

Returns a list of layers for every VectorObject of a PageElement according to the order defined chosen with the implementation. (We suggest, you change it to a dict of LayoutClasses and Layers for a safer implementation.) :param img: base img to be drawn upon. :type img: Image :param layers: layers of vector objects each drawn on a layer. This PageElement will be drawn on a new layer and added to the list. :type layers: List[Layer] :return: layers: layers of vectorobjects (including this instance) each drawn on a layer. :rtype:layers: List[Layer]

resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Resizes the current :class: *PageElement* to a given target dimension. As this class doesn't possess a image dimension parameter, both the current dimension (of the page) and the target dimension (the size to be scaled to) should be given. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

set_color(*color: Optional[Tuple] = None*, *color_table: Optional[*ColorTable*] = None*)

Setter-method for the color. :param color: What color to use when drawn. :type color: Tuple

set_visible(*is_visible: Optional[bool] = None*, *vis_table: Optional[*VisibilityTable*] = None*)

Setter-method for the is_visible field. :param is_visible: Whether or not is should be recognized as visible :type is_visible: bool

show()

Displays the polygon of the PageElement for debugging purposes.

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**TextLine**(*polygon:* Polygon, *base_line:* BaseLine, *color: Tuple = (255, 255, 255)*, *is_visible: bool = True*, *id: int = 0*)

Bases: *PageElement*

Represents all PageElements with text line characteristics. Examples are comment lines and main-text lines. :param polygon: represents the contour of the text line. :type polygon: Polygon :param base_line: Every :class: *TextLine* class comes with a default base_line provided by the original xml_gt. :type base_line: Line :param

color: color of the line, defaults to white :type color: tuple :param is_visible: set visible or invisible, defaults to true :type is_visible: bool :param id: To identify the text region. Helpful for coloring, sorting, etc. Not mandatory, default value is 0. :type id: int

build() → Dict

Helper function to build the json file.

crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Crop the page element to a target dimension. Due to the nature of the ground truth document :param cut_left: must be provided. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension :param cut_left: Whether or not the page is cut_left or not. :type cut_left: bool

draw(*drawer: ImageDraw*, *color: Tuple = None*, *outline=None*)

Draw the page element :class: *PageElement* polygon with the instance color if no other color is given. :param drawer: image to be drawn upon. :type drawer: ImageDraw.ImageDraw :param color: fill of the object. Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type color: tuple :param outline: outline of the object. Should only be used for illustration purposes! Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type outline: tuple

layer(*img: Image*, *layers: List[*Layer*] = None*) → List[*Layer*]

Returns a list of layers for every VectorObject of a PageElement according to the order defined chosen with the implementation. (We suggest, you change it to a dict of LayoutClasses and Layers for a safer implementation.) :param img: base img to be drawn upon. :type img: Image :param layers: layers of vector objects each drawn on a layer. This PageElement will be drawn on a new layer and added to the list. :type layers: List[Layer] :return: layers: layers of vectorobjects (including this instance) each drawn on a layer. :rtype:layers: List[Layer]

resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Resizes the current :class: *PageElement* to a given target dimension. As this class doesn't possess a image dimension parameter, both the current dimension (of the page) and the target dimension (the size to be scaled to) should be given. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

set_color(*color: Optional[Tuple] = None*, *color_table: Optional[*ColorTable*] = None*)

Setter-method for the color. :param color: What color to use when drawn. :type color: Tuple

set_visible(*is_visible: Optional[bool] = None*, *vis_table: Optional[*VisibilityTable*] = None*)

Setter-method for the is_visible field. :param is_visible: Whether or not is should be recognized as visible :type is_visible: bool

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**TextLineDecoration**(*text_line:* TextLine)

Bases: *TextLine*

Decorator class used to add further functionalities to :class: *TextLine* page elements. :param polygon: represents the contour of the text line. :type polygon: Polygon :param base_line: Every :class: *TextLine* class comes with a default base_line provided by the original xml_gt. :type base_line: Line :param color: color of the line, defaults to white :type color: tuple :param is_visible: set visible or invisible, defaults to true :type is_visible: bool :param id: To identify the text region. Helpful for coloring, sorting, etc. Not mandatory, default value is 0. :type id: int

abstract crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Crop the page element to a target dimension. Due to the nature of the ground truth document :param cut_left: must be provided. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension :param cut_left: Whether or not the page is cut_left or not. :type cut_left: bool

abstract draw(*drawer: ImageDraw*, *color: Tuple = None*, *outline=None*)

Draw the page element :class: *PageElement* polygon with the instance color if no other color is given. :param drawer: image to be drawn upon. :type drawer: ImageDraw.ImageDraw :param color: fill of the object. Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type color: tuple :param outline: outline of the object. Should only be used for illustration purposes! Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type outline: tuple

abstract resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Resizes the current :class: *PageElement* to a given target dimension. As this class doesn't possess a image dimension parameter, both the current dimension (of the page) and the target dimension (the size to be scaled to) should be given. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

abstract set_color(*color: Optional[Tuple] = None*, *color_table: Optional[*ColorTable*] = None*)

Setter-method for the color. :param color: What color to use when drawn. :type color: Tuple

abstract set_visible(*is_visible: Optional[bool] = None*, *vis_table: Optional[*VisibilityTable*] = None*)

Setter-method for the is_visible field. :param is_visible: Whether or not is should be recognized as visible :type is_visible: bool

abstract show()

Displays the polygon of the PageElement for debugging purposes.

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**TextLineElements**(*polygon:* Polygon, *id: int = 0*)

Bases: *PageElement*

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**TextRegionElement**(*bounding_box:* Rectangle, *id: int = 0*)

Bases: *TextLineElements*

Represents the bounding box of a Text-Region. Helps illustrate and the structure of the vector_gt. :param bounding_box: bounding box of the Text-Region. :type bounding_box: bounding box of the Text-Region. :param id: To identify the text region. Helpful for coloring, sorting, etc. Not mandatory, default value is 0. :type id: int

draw(*drawer: ImageDraw*, *color: Tuple = None*, *outline: Tuple = None*)

Draw the page element polygon with the instance color if no other coller is given. Fill it, if and only if the set_filled parameter is true.

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**TopLine**(*base_line:* BaseLine, *x_height: int*, *id: int = 0*)

Bases: *TextLineElements*

set_visible(*is_visible: Optional[bool] = None*, *vis_table: Optional[*VisibilityTable*] = None*)

Setter-method for the is_visible field. :param is_visible: Whether or not is should be recognized as visible :type is_visible: bool

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageElements.**XRegion**(*base_line:* BaseLine, *top_line:* TopLine, *id: int = 0*)

Bases: *TextLineElements*

Region to represent x-Height (or XRegion as it's called in this program) based on a top_line and the polygon of the text_line :param base_line: Base line (base line - some y value representing the x-height) :type base_line: BaseLine :param top_line: top_line: Top line (base line - some y value representing the x-height) :type top_line: TopLine :param id: To identify the region. Helpful for coloring, sorting, etc. Not mandatory, default value is 0. :type id: int

GTRefiner.GTRepresentation.VectorGTRepresentation.PageLayout module

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageLayout.**Layout**(*page_elements: Optional[List[*PageElement*]] = None*, *layout_class: Optional[*LayoutClasses*] = None*)

Bases: *Scalable*, *Drawable*, *Croppable*, *Dictionable*, *Layarable*

Parameters

- **page_elements** (*List[*PageElement*]*) – List of all page elements of a layout (e.g. comment text lines or decorations).
- **layout_class** (LayoutClasses) – The class of the layout (e.g. COMMENT or DECORATION)

add_elem(*elem:* PageElement)

Append an element to this layout. Can also be done with the built-in function append, however this method is is safer. :param elem: :type elem: :return: :rtype:

build() → Dict

Introduces the behaviour of a JSON compliant represenation. The returned dictionary is supposed to be stored in a JSON file. :return: Dictionary of all the vector objects of interest. :rtype: dict

crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Crop all page elements of this layout to a target dimension. Due to the nature of the ground truth document :param cut_left: must be provided. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension :param cut_left: Whether or not the page is cut_left or not. :type cut_left: bool

draw(*drawer: ImageDraw*, *color: Optional[Tuple] = None*, *outline=None*)

If layout is visible, draw all page elements of this layout on the given image in a certain color and outline. :param drawer: image to be drawn upon. :type drawer: ImageDraw.ImageDraw :param color: fill of the object. Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type color: tuple :param outline: outline of the object. Should only be used for illustration purposes! Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type outline: tuple

get_region_bbox() → *TextRegionElement*

Dynamically returns a TextRegion element.

layer(*img: Image*)

Returns a image of all layers for every PageElement according to the order chosen with the implementation. (We suggest, you change it to a dict of LayoutClasses and Layers for a safer implementation.) :param img: base img to be drawn upon. :type img: Image TODO: Make safer with dict.

merge(*other:* Layout)

Merge this layout with another one. Doesn't do any layout_class checks. Client must be sure that they are both of the same layout class.

resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Resizes all page elements :class: *List[PageElement]* of the current :class: *LayoutClass* to a given target dimension. As this class doesn't possess a image dimension parameter, both the current dimension (of the page) and the target dimension (the size to be scaled to) must be given. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

set_color(*color: Optional[Tuple] = None*, *color_table: Optional[*ColorTable*] = None*)

Setter-method for the color. :param color: What color to use when drawn. :type color: Tuple

set_visible(*is_visible: Optional[bool] = None*, *vis_table: Optional[*VisibilityTable*] = None*)

Setter-method for the is_visible field. :param is_visible: Whether or not is should be recognized as visible :type is_visible: bool

class GTRefiner.GTRepresentation.VectorGTRepresentation.PageLayout.**TextRegion**(*layout: Optional[*Layout*] = None*, *text_regions: Optional[List[*Layout*]] = None*)

Bases: *Layout*

add_region(*layout:* Layout)

Add a layout to the region. Doesn't do any layout_class checks. Client must be sure that they are both of the same layout class. :param layout: Layout to be added to the region. :type layout: Layout

build() → Dict

Introduces the behaviour of a JSON compliant represenation. The returned dictionary is supposed to be stored in a JSON file. :return: Dictionary of all the vector objects of interest. :rtype: dict

get_region_bbox() → *TextRegionElement*

Dynamically returns a TextRegion element.

layer(*img: Image*)

Returns a image of all layers for every layout on the corresponding layer according to the order chosen with the implementation. :param img: base img to be drawn upon. :type img: Image

sort_layout_elements(*reverse: bool = False*)

Sorts the page_elements of a text_region by the min y value of the page elements.

GTRefiner.GTRepresentation.VectorGTRepresentation.VectorGT module

class GTRefiner.GTRepresentation.VectorGTRepresentation.VectorGT.**VectorGT**(*regions:* *List[*TextRegion*]*, *img_dim:* ImageDimension)

Bases: *GroundTruth*, *Dictionable*, *Scalable*, *Croppable*, *Drawable*

build() → Dict

Introduces the behaviour of a JSON compliant represenation. The returned dictionary is supposed to be stored in a JSON file. :return: Dictionary of all the vector objects of interest. :rtype: dict

crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Crop all page elements of this vector gt to a target dimension. Due to the nature of the ground truth document :param cut_left: must be provided. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension :param cut_left: Whether or not the page is cut_left or not. :type cut_left: bool

draw(*drawer: <module 'PIL.ImageDraw' from '/opt/miniconda3/envs/BachelorThesis/lib/python3.8/site-packages/PIL/ImageDraw.py'>*, *color: ~typing.Optional[~typing.Tuple] = None*, *outline=None*)

For all visible layouts and pagelements, draw them on given image in a certain color and outline. TODO: What's the difference of this method to the layer method? :param drawer: image to be drawn upon. :type drawer: ImageDraw.ImageDraw :param color: fill of the object. Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type color: tuple :param outline: outline of the object. Should only be used for illustration purposes! Can be of any mode that pillow understands (e.g. RGBA, 1, L, RGB). :type outline: tuple

get_dim()

Returns
Return the dimension of this vector_gt

Return type
ImageDimension

layer(*img*)

Draw all visible page_elements of all visible layers of all visible regions on an image.

resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Resizes all page elements :class: *List[PageElement]* of the current :class: *LayoutClass* to a given target dimension. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

show(*base_img: <module 'PIL.Image' from '/opt/miniconda3/envs/BachelorThesis/lib/python3.8/site-packages/PIL/Image.py'> = None*, *transparency=0.5*, *color: ~typing.Tuple = None*, *outline: ~typing.Tuple = None*)

Illustrate the vector ground truth according to the clients needs (especially useful for debugging). :param base_img: If given, this image will be the base image where the vector objects are drawn upon. :type base_img: Image :param transparency: The factor to which the layers should be overlayed. A value smaller than 0.5 prioritizes the base img. A value bigger than 05. prioritizes the vector objects. :type transparency: int :param color: color the page elements should be drawn in. We suggest you use a color table and :class: *Colorer* to color the elements individually. :type color: tuple :param outline: Outline of the vector objects to be drawn (illustrated). :type outline: tuple :return: :rtype:

GTRefiner.GTRepresentation.VectorGTRepresentation.VectorObjects module

class GTRefiner.GTRepresentation.VectorGTRepresentation.VectorObjects.**Line**(*xy*)

Bases: *Polygon*

Represents a line of two coordinates.

draw(*drawer: ImageDraw*, *outline=None*, *color: Tuple = None*)

Draw the vector object on a given object. :param drawer: Pillow drawer, can be of any mode (example "1", "RGB", "RGBA"). The color parameter must be corresponding format. :type drawer: ImageDraw.ImageDraw :param color: Single element tuple for binary, triple tuple for RGB, quadruple tuple for RGBA (consult PILLOW Documenation for more information). :type color: tuple :param outline: Only use this parameter for illustration purposes. Using it for ground truth generation is dangerous. :type outline: tuple :return: :rtype:

get_max_x_coord()

get_min_x_coord()

class GTRefiner.GTRepresentation.VectorGTRepresentation.VectorObjects.**Polygon**(*xy: List[Tuple]*)

Bases: *Scalable*, *Drawable*, *Croppable*, *Dictionable*

This class is the parent class of all vector objects used to represent the vector ground truth. :param xy: is a polygon of 2 or more coordinates. :type xy: represents the coordinates [(x1,x1),(x2,y2),...] of the polygon. It's important the polygon is sorted. No checks are made for that. It doesn't matter if the polygon is closed or open (the last coordinate doesn't need to be the first coordinate).

crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Crop this vector object to a target dimension. Due to the nature of the ground truth document :param cut_left: must be provided. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension :param cut_left: Whether or not the page is cut_left or not. :type cut_left: bool

draw(*drawer: ImageDraw*, *color: Tuple = None*, *outline: Tuple = None*)

Draw the vector object on a given object. :param drawer: Pillow drawer, can be of any mode (example "1", "RGB", "RGBA"). The color parameter must be corresponding format. :type drawer: ImageDraw.ImageDraw :param color: Single element tuple for binary, triple tuple for RGB, quadruple tuple for RGBA (consult PILLOW Documenation for more information). :type color: tuple :param outline: Only use this parameter for illustration purposes. Using it for ground truth generation is dangerous. :type outline: tuple :return: :rtype:

get_bbox() → *Rectangle*

Get bounding box of the vector object. :return: returns a rectangle defining the bounding box. :rtype: Rectangle

get_max_x()

Get max x coordinate of all the coordinates of this vector object. :return: max x :rtype: int

get_max_y()

Get max y coordinate of all the coordinates of this vector object. :return: max y :rtype: int

get_min_x()

Get min x coordinate of all the coordinates of this vector object. :return: min x :rtype: int

get_min_y()

Get min y coordinate of all the coordinates of this vector object. :return: min y :rtype: int

resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Resizes the vector object to a given target dimension. As this class doesn't possess a image dimension parameter, both the current dimension (of the page) and the target dimension (the size to be scaled to) must be given. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

class GTRefiner.GTRepresentation.VectorGTRepresentation.VectorObjects.**Quadrilateral**(*xy: List[Tuple]*)

Bases: *Polygon*

Parameters

xy – 4 coords that represent a quadrilateral. which are sorted upon initialisation to ensure a useful representation with the draw method.

crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Crop this vector object to a target dimension. Due to the nature of the ground truth document :param cut_left: must be provided. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension :param cut_left: Whether or not the page is cut_left or not. :type cut_left: bool

draw(*drawer: ImageDraw*, *color=None*, *outline=None*)

Draw the vector object on a given object. :param drawer: Pillow drawer, can be of any mode (example "1", "RGB", "RGBA"). The color parameter must be corresponding format. :type drawer: ImageDraw.ImageDraw :param color: Single element tuple for binary, triple tuple for RGB, quadruple tuple for RGBA (consult PILLOW Documenation for more information). :type color: tuple :param outline: Only use this parameter for illustration purposes. Using it for ground truth generation is dangerous. :type outline: tuple :return: :rtype:

is_sorted()

resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Resizes the vector object to a given target dimension. As this class doesn't possess a image dimension parameter, both the current dimension (of the page) and the target dimension (the size to be scaled to) must be given. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

class GTRefiner.GTRepresentation.VectorGTRepresentation.VectorObjects.**Rectangle**(*xy*)

Bases: *Polygon*

Represents a rectangle :param xy: 4 coords that represent a quadrilateral. which are sorted upon initialisation to ensure a useful representation with the draw method. :type tuple

draw(*drawer: ImageDraw*, *color: Tuple = None*, *outline=None*, *width=None*)

Draw the vector object on a given object. :param drawer: Pillow drawer, can be of any mode (example "1", "RGB", "RGBA"). The color parameter must be corresponding format. :type drawer: ImageDraw.ImageDraw :param color: Single element tuple for binary, triple tuple for RGB, quadruple tuple for RGBA (consult PILLOW Documenation for more information). :type color: tuple :param outline: Only use this parameter for illustration purposes. Using it for ground truth generation is dangerous. :type outline: tuple :return: :rtype:

Module contents

Submodules

GTRefiner.GTRepresentation.GroundTruth module

class GTRefiner.GTRepresentation.GroundTruth.**GroundTruth**(*img_dim:* ImageDimension)

Bases: *Scalable*, *Showable*, *Croppable*

GroundTruth is an abstract class that provides the provides the common interface for ground-truth classes.

crop(*current_dim:* ImageDimension, *target_dim:* ImageDimension, *cut_left: bool*)

Crop all ground truth information of this ground truth to a target dimension. Due to the nature of the ground truth document. :param cut_left: must be provided. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension :param cut_left: Whether or not the page is cut_left or not. :type cut_left: bool

get_dim()

Get the ground truth's dimension. :return: dimension of the ground truth. :rtype: ImageDimension

abstract resize(*current_dim:* ImageDimension, *target_dim:* ImageDimension)

Resizes all ground-truth information of the current :class: *LayoutClass* to a given target dimension. :param current_dim: Current dimension :type current_dim: ImageDimension :param target_dim: Target dimension :type target_dim: ImageDimension

abstract show()

Illustrate the ground truth.

GTRefiner.GTRepresentation.ImageDimension module

class GTRefiner.GTRepresentation.ImageDimension.**ImageDimension**(*width: 'int' = 0*, *height: 'int' = 0*)

Bases: object

difference(*other:* ImageDimension) → Tuple[float, float]

Method used solely for cropping. Calculates the difference for which a vector object has to be moved given another target dimension. :param other: other dimension :type other: ImageDimension :return: the difference in x,y as float tuple. :rtype: tuple

height: int = 0

scale_factor(*other:* ImageDimension)

Helper method for resizing. Gives the scale factor for which a given ground truth object should be resized. :param other: target dimension. :type other: ImageDimension :return: list of width scale factor and height scale factor :rtype:

to_tuple()

Get the dimension as a tuple :return: Width and height :rtype: tuple

width: int = 0

GTRefiner.GTRepresentation.LayoutClasses module

class GTRefiner.GTRepresentation.LayoutClasses.**LayoutClasses**(*value*)

Bases: Enum

Represents the different levels of the pixel level ground truth. Essential to the program. Must changed if further classes should to be added.

ASCENDER = 16

BACKGROUND = 1

BASELINE = 22

COMMENT = 2

COMMENT_AND_DECORATION = 6

COMMENT_AND_MAINTEXT = 10

DECORATION = 4

DESCENDER = 20

HEAD = 26

MAINTEXT = 8

MAINTEXT_AND_DECORATION = 12

MAINTEXT_AND_DECORATION_AND_COMMENT = 14

TAIL = 28

TEXT_REGION = 255

TOPLINE = 24

XREGION = 18

classmethod get_layout_classes_containing(*layout_class:* LayoutClasses)

Get all layout classes containing another one (based on the string). :param layout_class: target layout class :type layout_class: LayoutClasses :return: list of layout classes containing the target layou class as a substring. :rtype: list[LayoutClasses]

get_name() → str

Return the name of a enum instance. E.g. MAINTEXT :return: Return the name :rtype: str

classmethod str_to_enum(*enum_name: str*) → *LayoutClasses*

Return the matching enum based on a string. :param enum_name: :type enum_name: :return: :rtype:

GTRefiner.GTRepresentation.Page module

class GTRefiner.GTRepresentation.Page.**Page**(*vector_gt:* VectorGT, *px_gt:* PixelLevelGT, *raw_img:* RawImage, *vis_table:* VisibilityTable, *col_table:* ColorTable)

Bases: *Scalable*, *Croppable*, *Showable*

Stores all ground truth information of an image (including the original image, the vector gt and the pixel-based ground truth) and provides basic functionality for its manipulation such as scalability, croppability and showability. :param vector_gt: Stores polyogons of the various text elements (maintextline, decoration, textregion, etc.) :type vector_gt: VectorGT :param px_gt: Stores the original pixel level ground truth and its different information levels as :class: *Layers*. :type px_gt: PixelLevelGT :param raw_img: Stores the original image. :type raw_img: RawImage :param vis_table: Defines the visibility for the ground truth objects (texelements of the vector gt, levels of the pixel gt) :type vis_table: VisibilityTable :param col_table: Defines the colors for the ground truth objects (texelements of the vector gt, levels of the pixel gt) :type col_table: ColorTable

get_img_dim()

Get the ground truth's dimension. :return: dimension of the ground truth. :rtype: ImageDimension

visit_page(*visitor*)

Accepts visitors to manipulate the page and add further functionalities. :param visitor: manipulates the page (or invokes it's methods) to add specific behaviour :type visitor: Visitor

GTRefiner.GTRepresentation.Table module

class GTRefiner.GTRepresentation.Table.**ColorTable**(*table: Dict[*LayoutClasses*, List[Tuple]]*)

Bases: *Table*

Defines the colors for the ground truth objects (texelements of the vector gt, levels of the pixel gt). If you want to implement iterating or all different colors you need to provide a list of colors for the target LayoutClass. Compare with the color tables in the Ressources/ColorTables/ directory. :param table: Color table :type table: Dict[LayoutClasses, List[Tuple]

class GTRefiner.GTRepresentation.Table.**Table**(*table: Dict[*LayoutClasses*, Any]*)

Bases: object

Provides the base class for layout class :class: *LayoutClasses* based dictionaries. Can be dumped to JSON file. :param table: Dictionary of layout classes :type table: Dict[LayoutClasses, Any]

to_json(*file_path: Path = PosixPath('../../Resources/some_table.json')*)

Dumps the the table to json file with a given file path. :param file_path: Output file path :type file_path: Path

class GTRefiner.GTRepresentation.Table.**VisibilityTable**(*table: Dict[*LayoutClasses*, bool]*)

Bases: *Table*

Defines the visibility for the ground truth objects (texelements of the vector gt, levels of the pixel gt). :param table: Visibility table :type table: Dict[LayoutClasses, bool]

Module contents

GTRefiner.IO package

Submodules

GTRefiner.IO.Reader module

class GTRefiner.IO.Reader.**AbstractReader**

Bases: object

Interface or all reader like classes.

abstract classmethod read(*path: Path*) → Any

Read the file at given location (path).

class GTRefiner.IO.Reader.**ColorTableReader**

Bases: *TableReader*

classmethod read(*path: Path*) → *ColorTable*

Read a given json-based color table. :param path: path to json-based color table :type path: Path :return: color table :rtype: ColorTable

class GTRefiner.IO.Reader.**GTReader**

Bases: *AbstractReader*

abstract classmethod read(*path: Path*) → *GroundTruth*

Read the ground truth.

class GTRefiner.IO.Reader.**ImageReader**

Bases: *GTReader*

classmethod read(*path: Path*) → *RawImage*

Read the ground truth.

class GTRefiner.IO.Reader.**JSONReader**

Bases: *GTReader*

classmethod read(*path: Path*) → *VectorGT*

Method to read the json file and create a vector ground truth. :param xml_path: File path to PAGE XML. :type xml_path: Path :return: Vector ground truth :rtype: VectorGT

class GTRefiner.IO.Reader.**PxGTReader**

Bases: *GTReader*

classmethod read(*path: Path*) → *PixelLevelGT*

Read the ground truth.

class GTRefiner.IO.Reader.**TableReader**

Bases: *AbstractReader*

abstract classmethod read(*path: Path*) → Any

Read the file at given location (path).

class GTRefiner.IO.Reader.**VisibilityTableReader**

Bases: *TableReader*

classmethod read(*path: Path*) → *VisibilityTable*

Read a given json-based visibility table. :param path: path to json-based visibility table :type path: Path :return: visibility table :rtype: VisibilityTable

class GTRefiner.IO.Reader.**XMLReader**

Bases: *GTReader*

classmethod read(*path: Path*) → *VectorGT*

Reade the XML base PAGE and return a vector ground truth. :param path: File path to PAGE XML. :type path: Path :return: Vector ground truth :rtype: VectorGT

GTRefiner.IO.Writer module

class GTRefiner.IO.Writer.**AbstractWriter**

Bases: object

abstract classmethod write(*ground_truth: <module 'GTRefiner.GTRepresentation.GroundTruth' from '/Users/loverboy99/PycharmProjects/BachelorThesis/HisDB_GT_Refinement/GTRefiner/GTR*, *path: ~pathlib.Path*)

class GTRefiner.IO.Writer.**GIFWriter**

Bases: *ImageWriter*

classmethod write(*ground_truth: Image*, *path: Path*)

Store an image as GIF. :param ground_truth: raw image to be stored :type ground_truth: Image :param path: Output path :type path: Path

class GTRefiner.IO.Writer.**ImageWriter**

Bases: *AbstractWriter*

abstract classmethod write(*ground_truth:* MyImage, *path: Path*)

class GTRefiner.IO.Writer.**JSONWriter**

Bases: *VectorGTWriter*

classmethod write(*ground_truth:* VectorGT, *path: Path*)

Write the vector based ground truth with the build()-method provided by the vector ground truth. :param ground_truth: vector ground truth to be dumped to json :type ground_truth: VectorGT :param path: Output path :type path: Path

class GTRefiner.IO.Writer.**PNGWriter**

Bases: *ImageWriter*

classmethod write(*ground_truth: Image*, *path: Path*)

Store an image as GIF. :param ground_truth: raw image to be stored :type ground_truth: Image :param path: Output path :type path: Path

class GTRefiner.IO.Writer.**PXGTWriter**

Bases: *ImageWriter*

classmethod write(*ground_truth: Image*, *path: Path*)

Write px-based ground truth image. :param ground_truth: pixel based ground truth image to be stored :type ground_truth: Image :param path: Output path :type path: Path

class GTRefiner.IO.Writer.**RawImageWriter**

Bases: *ImageWriter*

classmethod write(*ground_truth:* RawImage, *path: Path*, *format: str = '.gif'*)

Write raw image (original image that may haven been resized and/or cropped). :param ground_truth: raw image to be stored, defaults to GIF. :type ground_truth: Image :param path: Output path :type path: Path

class GTRefiner.IO.Writer.**VectorGTWriter**

Bases: *AbstractWriter*

abstract classmethod write(*ground_truth:* VectorGT, *path: Path*)

class GTRefiner.IO.Writer.**XMLWriter**

Bases: *VectorGTWriter*

classmethod write(*ground_truth:* VectorGT, *path: Path*)

Module contents

GTRefiner.Resources package

Subpackages

GTRefiner.Resources.ColorTables package

Module contents

GTRefiner.Resources.IO_JSON package

Module contents

GTRefiner.Resources.NewGTs package

Module contents

GTRefiner.Resources.ResizedGT package

Module contents

GTRefiner.Resources.VisibilityTables package

Module contents

Module contents

GTRefiner.TestingAndClientExamples package

Subpackages

GTRefiner.TestingAndClientExamples.ClientExamples package

Subpackages

GTRefiner.TestingAndClientExamples.ClientExamples.Director package

Submodules

GTRefiner.TestingAndClientExamples.ClientExamples.Director.GroundTruthRefinerDemo1 module

GTRefiner.TestingAndClientExamples.ClientExamples.Director.GroundTruthRefinerDemo2 module

GTRefiner.TestingAndClientExamples.ClientExamples.Director.GroundTruthRefinerDemo3 module

GTRefiner.TestingAndClientExamples.ClientExamples.Director.RegionIllustratorDirector module

GTRefiner.TestingAndClientExamples.ClientExamples.Director.ResizingIllustratorDirector module

Module contents

GTRefiner.TestingAndClientExamples.ClientExamples.Visitor package

Submodules

GTRefiner.TestingAndClientExamples.ClientExamples.Visitor.BlurAndScaleResizerVisitor module

class GTRefiner.TestingAndClientExamples.ClientExamples.Visitor.BlurAndScaleResizerVisitor.**BlurAndScaleR**

Bases: *Resizer*

visit_page(*page:* Page)

Resize a page (and all it's ground-truth information, including the original image) to a target dimension. :param target_dim: Target dimension :type target_dim: ImageDimension

class GTRefiner.TestingAndClientExamples.ClientExamples.Visitor.BlurAndScaleResizerVisitor.**BlurAndScaleR**

Bases: *Resizer*

visit_page(*page:* Page)

Resize a page (and all it's ground-truth information, including the original image) to a target dimension. :param target_dim: Target dimension :type target_dim: ImageDimension

Module contents

Submodules

GTRefiner.TestingAndClientExamples.ClientExamples.MinorityAndMajorityWins module

class GTRefiner.TestingAndClientExamples.ClientExamples.MinorityAndMajorityWins.**CustomResizer**(*img: <module 'PIL.Image' from '/opt/minicon packages/PI strategy: ~GTRefiner.Testing*

Bases: *Resizer*

class GTRefiner.TestingAndClientExamples.ClientExamples.MinorityAndMajorityWins.**FlexibleResizer**

Bases: *ResizingStrategy*

flexibel_resize(*im: <module 'PIL.Image' from '/opt/miniconda3/envs/BachelorThesis/lib/python3.8/site-packages/PIL/Image.py'>*, *number_rows: int*, *number_col: int*)

resize(*img_as_array*, *resize_factor*)

class GTRefiner.TestingAndClientExamples.ClientExamples.MinorityAndMajorityWins.**MajorityWins**

Bases: *ResizingStrategy*

resize(*img_as_array*, *resize_factor*)

class GTRefiner.TestingAndClientExamples.ClientExamples.MinorityAndMajorityWins.**MeanWins**

Bases: *ResizingStrategy*

resize(*img_as_array*, *resize_factor*)

class GTRefiner.TestingAndClientExamples.ClientExamples.MinorityAndMajorityWins.**MinorityWins**

Bases: *ResizingStrategy*

resize(*img_as_array*, *resize_factor*)

class GTRefiner.TestingAndClientExamples.ClientExamples.MinorityAndMajorityWins.**NaiveResizer**(*img: <module 'PIL.Image' from '/opt/minicon packages/PIL strategy*)

Bases: *Resizer*

resize(*scaling_factor*) → array

class GTRefiner.TestingAndClientExamples.ClientExamples.MinorityAndMajorityWins.**Resizer**(*img: <module 'PIL.Image' from '/opt/miniconda3/en packages/PIL/Image strategy: ~GTRefiner.TestingAndClie*

Bases: object

abstract resize(*scaling_factor*) → array

class GTRefiner.TestingAndClientExamples.ClientExamples.MinorityAndMajorityWins.**ResizingStrategy**

Bases: object

abstract resize(*img_as_array*, *resize_factor*)

class GTRefiner.TestingAndClientExamples.ClientExamples.MinorityAndMajorityWins.**Strategy**(*value*)

Bases: Enum

An enumeration.

MAJORITY_WINS = 0

MINORITY_WINS = 1

RANDOM_WINS = 2

Module contents

GTRefiner.TestingAndClientExamples.Testing package

Subpackages

GTRefiner.TestingAndClientExamples.Testing.Algorithms package

Subpackages

GTRefiner.TestingAndClientExamples.Testing.Algorithms.Combining package

Submodules

GTRefiner.TestingAndClientExamples.Testing.Algorithms.Combining.LayeringTester module

GTRefiner.TestingAndClientExamples.Testing.Algorithms.Combining.LayeringTester.**center_polygon**(*polygon:* Polygon, *min_x=None* *min_y=None*

GTRefiner.TestingAndClientExamples.Testing.Algorithms.Combining.LayeringTester.**test_merge_and_draw**()

GTRefiner.TestingAndClientExamples.Testing.Algorithms.Combining.LayeringTester.**test_on_text_line**()

GTRefiner.TestingAndClientExamples.Testing.Algorithms.Combining.LayeringTester.**test_real_gt**()

GTRefiner.TestingAndClientExamples.Testing.Algorithms.Combining.LayeringTester.**test_two_text_lines**()

GTRefiner.TestingAndClientExamples.Testing.Algorithms.Combining.bin_layer_from_rgb_tester module

class GTRefiner.TestingAndClientExamples.Testing.Algorithms.Combining.bin_layer_from_rgb_tester.**TestLay**

Bases: TestCase

test__bin_layer_from_rgb()

Module contents

Module contents

GTRefiner.TestingAndClientExamples.Testing.GTRepresentation package

Subpackages

GTRefiner.TestingAndClientExamples.Testing.GTRepresentation.ScalableTester package

Submodules

GTRefiner.TestingAndClientExamples.Testing.GTRepresentation.ScalableTester.CroppingTester module

GTRefiner.TestingAndClientExamples.Testing.GTRepresentation.ScalableTester.QuadrilateralSortingTester module

GTRefiner.TestingAndClientExamples.Testing.GTRepresentation.ScalableTester.QuadrilateralSortingTester.t

GTRefiner.TestingAndClientExamples.Testing.GTRepresentation.ScalableTester.TestResizing module

Module contents

Submodules

GTRefiner.TestingAndClientExamples.Testing.GTRepresentation.EnumTester module

GTRefiner.TestingAndClientExamples.Testing.GTRepresentation.PageElementTester module

Module contents

GTRefiner.TestingAndClientExamples.Testing.IO package

Submodules

GTRefiner.TestingAndClientExamples.Testing.IO.GIFReaderTester module

GTRefiner.TestingAndClientExamples.Testing.IO.GIFReaderTester.**get_raw_palette**(*img: <module 'PIL.Image' from '/opt/miniconda3/envs/BachelorT packages/PIL/Image.py'>*)

GTRefiner.TestingAndClientExamples.Testing.IO.ReadAndWritingTester module

GTRefiner.TestingAndClientExamples.Testing.IO.TableWriter module

GTRefiner.TestingAndClientExamples.Testing.IO.XMLReaderTester module

Module contents

Module contents

Module contents

1.1.2 Module contents

UNI
FR

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Faculté des sciences économiques et sociales
Wirtschafts- und sozialwissenschaftliche Fakultät
Boulevard de Pérolles 90
CH-1700 Fribourg

E R K L Ä R U N G

Ich bestätige mit meiner Unterschrift, dass ich die Arbeit persönlich erstellt und dabei nur die aufgeführten Quellen und Hilfsmittel verwendet sowie wörtliche Zitate und Paraphrasen als solche gekennzeichnet habe.

Es ist mir bekannt, dass andernfalls die Fakultät gemäss der Entscheidung des Fakultätsrats vom 09.11.2004 das Recht hat, den auf Grund dieser Arbeit verliehenen Titel zu entziehen.

Ich erkläre hiermit weiterhin, dass diese Arbeit bzw. Teile daraus noch nicht in dieser Form an anderer Stelle als Prüfungsleistung eingereicht worden sind, gemäss der Entscheidung des Fakultätsrats vom 18.11.2013.

……, den ..... 20.....

.....

(Unterschrift)