
# /*eyedentificator*/

http://www.eyedentificator.org/

## Studienarbeit Informatik

### Content Based Information Retrieval
### Open-Source Framework

# *Projektvorstellung*


| | |
|---|---|
| Autor | Pascal Gadient |
| Betreuer | Prof. Dr. Josef M. Joller |


| | |
|---|---|
| Erstellt am | 22. September 2011 |
| Letzte Änderung am | 19. Dezember 2011 |
| Version | 1.0 |

# 0. Dokumentinformationen

## 0.1. Änderungsgeschichte

| *Datum* | *Version* | *Änderung* |
|---|---|---|
| 22.09.11 | 0.1 | Initiale Version aus Vorlage erstellt |
| 07.10.11 | 0.2 | Struktur erstellt |
| 09.10.11 | 0.3 | Kapitel 1 begonnen |
| 21.11.11 | 0.4 | Umfassende Restrukturierung, Kapitel 1.2 abgeschlossen |
| 22.11.11 | 0.5 | Kapitel 1.3 & 1.4 abgeschlossen, Kapitel 1.5 begonnen |
| 23.11.11 | 0.51 | Kapitel 1.5 & 1.6.1 abgeschlossen, Kapitel 1.6.2 begonnen |
| 30.11.11 | 0.6 | Kapitel 2.2 bearbeitet |
| 01.12.11 | 0.7 | Kapitel 2.2 bis und mit Kapitel 4.2 abgeschlossen, dies umfasst:<br>- Filtererklärungen<br>- Konkrete Filteranwendungen<br>- Beschrieb der Comparation Modules<br>- Testreihen der Comparation Modules<br>u.v.m. |
| 17.12.11 | 0.8 | - Titel geändert von „Software Architecture Document“ nach „Projektvorstellung“<br>- Überarbeitung des Dokuments (Korrekturvorschläge des Betreuers umgesetzt) |
| 18.12.11 | 0.9 | - Überarbeitung des Dokuments (Korrekturvorschläge des Betreuers umgesetzt) |
| 19.12.11 | 1.0 | - kleinere finale Anpassungen vorgenommen |

## 0.2. Übersicht & Zweck

Dieses Dokument zeigt das „WIE“, d. h. wie Entscheide für Lösungsansätze gefällt wurden, oder aber auch wie die Software strukturiert wurde, oder ganz allgemein: „Wie funktioniert diese Software?“.
Diese Antworten sollen einen Einblick in die Gedankenwelt dieses Softwareprojekts vermitteln. Somit ist dieses Dokument bestens geeignet um über das gesamte Projekt einen Überblick zu erhalten.

## 0.3. Definitionen und Abkürzungen

Das Glossar befindet sich unter /Dokumente/04_Konventionen/Glossary.docx.

## 0.4. Referenzen

Folgende externe Quellen wurden für die Erstellung der Software benötigt:


| | |
|---|---|
| **[01]** | ***JUnit 4 Tutorial,***<br>http://patricklam.ca/stqam/files/junit__tutorial.pdf<br>letzter Zugriff am 18.12.2011 |
| **[02]** | ***JUnit Tutorial „JUnit in 60 seconds“,***<br>http://www.cavdar.net/2008/07/21/junit-4-in-60-seconds/<br>letzter Zugriff am 18.12.2011 |
| **[03]** | ***Factory Pattern,***<br>Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994, ISBN 0-201-63361-2 |
| **[04]** | ***Pipes & Filters Pattern,***<br>Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, „Pattern-Oriented Software Architecture“, John Wiley & Sons Ltd., Juli, 1996, pp. 53 - 70 |
| **[05]** | ***Reflection Pattern,***<br>Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, „Pattern-Oriented Software Architecture“, John Wiley & Sons Ltd., Juli, 1996, pp. 211 - 237 |
| **[06]** | ***Ansätze der Bildverarbeitung und Auswertung,***<br>http://wwwmath.uni-muenster.de:8010/u/lammers/EDU/ws03/Landminen/Abgaben/Gruppe9/Thema09-ObjekterkennungInBilddaten-ChristianGrosseLordemann-MartinLambers.pdf<br>letzter Zugriff am 18.12.2011 |

| | |
|---|---|
| **[07]** | ***Ansätze der Bildverarbeitung und Auswertung,***<br>Niklas Peinecke, „Eigenwertspektren des Laplaceoperators in der Bilderkennung“, Books on Demand GmbH, 2006, ISBN 3-8334-5072-X |
| **[08]** | ***Beschreibung von diversen Bildverarbeitungsalgorithmen,***<br>http://www.jhlabs.com/ip/filters/index.html<br>letzter Zugriff am 18.12.2011 |
| **[09]** | ***Bildeditor, verwendet Technologien von [4],***<br>http://www.jhlabs.com/ie/ImageEditor.jar<br>letzter Zugriff am 18.12.2011 |
| **[10]** | ***Bildverarbeitungsfilter, verwenden Technologien von [4],***<br>http://www.jhlabs.com/ip/filters/Filters.zip<br>letzter Zugriff am 18.12.2011 |
| **[11]** | ***Gauss Algorithmus im 2D Koordinatenraum,***<br>R.A. Haddad and A.N. Akansu, "A Class of Fast Gaussian Binomial Filters for Speech and Image Processing", IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 39, March 1991, pp. 723-727 |
| **[12]** | ***Gauss Algorithmus im 2D Koordinatenraum,***<br>Shapiro, L. G. & Stockman, "Computer Vision", 2001, page 137 |
| **[13]** | ***Gauss Algorithmus im 2D Koordinatenraum,***<br>Mark S. Nixon, Alberto S. Aguado, "Feature Extraction and Image Processing", Academic Press, 2008, page 88 |
| **[14]** | ***Maple Befehlsübersicht,***<br>http://faculty.uca.edu/weijiul/MATH3331_DiffEq/Arrigo_Maple_Lect2.pdf<br>letzter Zugriff am 18.12.2011 |
| **[15]** | ***Oracle Java Dokumentation,***<br>http://docs.oracle.com/javase/1.4.2/docs/api/index.html<br>letzter Zugriff am 18.12.2011 |
| **[16]** | ***Beschrieb Windows Netzwerktreiberbug,***<br>http://support.uptimesoftware.com/article.php?id=280<br>letzter Zugriff am 18.12.2011 |


## 0.5. Interne Referenzen

Projektdokumente

- /Dokumente/00_Vorlagen/Vorlage Studienarbeit.dotm
- /Dokumente/01_Projektantrag/Projektantrag.docx
- /Dokumente/02_Projektrahmenbedingungen/Anforderungsspezifikationen.docx
- /Dokumente/02_Projektrahmenbedingungen/Projektplan.docx
- /Dokumente/04_Konventionen/Glossary.docx
- /Dokumente/04_Konventionen/StyleGuide.docx
- /Dokumente /13_JavaDoc (Dokumentation vom Programmcode in HTML)

## 0.6. Inhaltsverzeichnis


0. Dokumentinformationen ... 2

0.1. Änderungsgeschichte ... 2
0.2. Übersicht & Zweck ... 2
0.3. Definitionen und Abkürzungen ... 2
0.4. Referenzen ... 2
0.5. Interne Referenzen ... 3
0.6. Inhaltsverzeichnis ... 4

1. Abstract ... 6

1.1. Ausgangslage ... 6
1.2. Aufgabenstellung ... 6
1.3. Vorgehen ... 6
1.4. Das Konzept ... 6

2. Software-Systemarchitektur ... 7

2.1. Architekturübersicht ... 7
2.1.1. Einführung ... 7
2.1.2. Ziele ... 7
2.1.3. Einschränkungen ... 7
2.2. Systemstruktur ... 8
2.2.1. Physische Schicht ... 8
2.2.2. Logische Sicht ... 9
2.2.2.1. System eyedentificator ... 9
2.2.2.1. Package filter ... 11
2.2.2.2. Package data ... 13
2.2.2.3. Package gui ... 15
2.3. Angewandte Software-Patterns ... 17
2.3.1. Factory ... 17
2.3.2. Variation Pipe and Filters ... 17
2.3.3. Reflection ... 18
2.4. Process View ... 20
2.4.1. Handling eines asynchronen Requests via Netzwerk ... 20
2.5. Architekturkonzepte ... 21
2.5.1. Protokoll ... 21
2.5.2. Datenbank ... 25
2.5.3. Persistenz der Filterprofile ... 27
2.6. Sequenzdiagramme ... 29
2.6.1. Initialisierungspfad Starten des Servers ... 29
2.6.2. Comparation Prozess ... 30
2.6.3. Datenbank Update Prozess ... 31

3. Filterentwicklung und Testreihen ... 33

3.1. Einleitung ... 33
3.2. Fremde Filter ... 33
3.2.1. Sharpen ... 33
3.2.2. Level ... 33
3.2.3. DifferenceOfGaussians ... 34
3.2.4. GaussianBlur ... 34
3.3. Eigene Filter ... 37
3.3.1. Stretch ... 37
3.3.2. Crop ... 37
3.3.3. ColorDifferenceWithBacklog ... 37
3.3.4. FreeFormSelect ... 38
3.3.5. Grayscale ... 40
3.3.6. BlackAndWhiteOnly ... 40
3.3.7. BlackAreaRemover ... 40
3.3.8. Dot ... 40
3.3.9. InvertOnDemand ... 40

3.3.10. MoveBWImage ..... 41
3.3.11. ColorHistogramGraph ..... 41
3.3.12. HorizontalBlackGraph ..... 41
3.3.13. VerticalBlackGraph ..... 41
3.4. Eigene Comparation Modules ..... 41
3.4.1. SquareSplitCount ..... 41
3.4.2. ColorHistogram ..... 41
3.4.3. Size2D ..... 41
3.5. Konkrete Applizierung der Filter ..... 41
3.5.1. Filter - Sharpen ..... 41
3.5.2. Filter - Level ..... 42
3.5.3. Filter - DifferenceOfGaussians ..... 42
3.5.4. Filter - GaussianBlur ..... 43
3.5.5. Filter - Stretch ..... 44
3.5.6. Filter - Crop ..... 44
3.5.7. Filter - ColorDifferenceWithBacklog ..... 45
3.5.8. Filter - FreeFormSelect ..... 46
3.5.9. Filter - Grayscale ..... 47
3.5.10. Filter - BlackAndWhiteOnly ..... 47
3.5.11. Filter - BlackAreaRemover ..... 47
3.5.12. Filter - Dot ..... 48
3.5.13. Filter - InvertOnDemand ..... 49
3.5.14. Filter - MoveBWImage ..... 50
3.5.15. Filter - ColorHistogramGraph ..... 50
3.5.16. Filter - HorizontalBlackGraph ..... 51
3.5.17. Filter - VerticalBlackGraph ..... 51
3.6. Testreihen mit den vordefinierten Filterprofilen ..... 51
3.6.1. Profil „Color Comparison“ ..... 51
3.6.2. Profil „FreeForm With Color Comparison“ ..... 53
3.6.3. Profil „FreeForm With Size Comparison“ ..... 54

4. Abschliessende Erkenntnisse ..... 57

4.1. Rückblick & Erfahrungen ..... 57
4.2. Fazit ..... 57
4.3. Dankesworte ..... 57

5. Anhang ..... 58

5.1. Proof-of-Concepts ..... 58
5.2. Verwendete Plug-ins ..... 60
5.2.1. Video- & Kameraansteuerung ..... 60
5.2.2. Datenbank 1 ..... 60
5.2.3. Datenbank 2 ..... 60
5.2.4. Diagramme ..... 61
5.2.5. Pfadmanagement für VLC ..... 61
5.3. JavaDoc ..... 61

# 1. Abstract

## 1.1. Ausgangslage

Es gibt zurzeit nur sehr wenige Bibliotheken und Anwendungen (kommerziell ebenso wie open-source), welche eine Erkennung von Objekten ermöglichen. Weiterhin sind diese Lösungen oft nur sehr eingeschränkt einsetzbar, da sie ausschliesslich für einen bestimmten Einsatzzweck konstruiert worden sind.

## 1.2. Aufgabenstellung

Ziel dieser Arbeit ist ein Open-Source Framework zu erstellen, welches beliebig erweitert und auf die eigenen Bedürfnisse angepasst werden kann. Dazu wird neben einer Serverkomponente mit der entsprechenden Logik auch ein Client für unterschiedliche Plattformen (PC / Windows Phone / Apple iOS / Android) benötigt. Die mobilen Clients gehören jedoch nicht zu dieser Arbeit. Es soll in einem ersten Schritt möglich sein ein Bild als Bilddatei über ein Netzwerk dem Server zu übergeben, damit er Vergleiche starten und ein Resultat generieren kann. Die Resultate dieses Projekts sollen ausserdem sehr portabel sein, d. h. sich mit einem Doppelklick starten lassen, ohne langwierige Installation. Zusätzlich sollen verschiedene Grafikalgorithmen (= Filter) auf ein Bild angewendet werden können, um eine allfällige Erkennung verbessern zu können. Die verwendete Sprache ist Java 7.

## 1.3. Vorgehen

Ausgehend von zwölf Prototypen in allen möglichen Bereichen wurde ein Serverbackend erstellt. Die Filter wurden nach dem Trial & Error Prinzip erstellt. Als Basis für die Filter dienten meist bewährte Techniken, welche durch den pragmatisch / experimentellen Ansatz zu den unterschiedlichsten Filtern verholfen haben. Falls eine Komponente / Funktion während dem Projekt sich unvorhergesehen als unverzichtbar herausstellte, wurde sie bei der nächsten Gelegenheit implementiert, so geschehen beispielsweise beim eigens entwickelten Netzwerkprotokoll.

## 1.4. Das Konzept

Das Serverbackend nimmt Netzwerkverbindungen entgegen und arbeitet die Anfragen der Clients ab. Der Server erlaubt die dynamische Konfiguration von wesentlich mehr als einem Dutzend unterschiedlicher Filter. Die besten und universellsten Filterkombinationen wurden in insgesamt drei Filterprofilen hinterlegt, welche nach dem Programmstart einfach und unkompliziert geladen werden können. Ein Grossteil der Filter lässt sich zusätzlich mit verschiedenen Parametern weiter im Detail konfigurieren. Ausserdem lässt sich die Software entweder mit einer integrierten Datenbank, oder auf Wunsch auch mit einer externen Datenbank für die Datenspeicherung betreiben.

# 2. Software-Systemarchitektur

## 2.1. Architekturübersicht

### 2.1.1. Einführung

Die Anwendung „eyedentificator“ soll eine Erkennung von Objekten ermöglichen, mit Hilfe verschiedener Technologien, wie z. B. einer Helligkeitsverteilung, Histogrammkurve, Merkmalsvektoren, oder wenn möglich sogar OCR. Diese Anwendung setzt sich aus zwei Teildomänen zusammen: Einerseits diese Studienarbeit, welche das Serverbackend bereitstellen soll, andererseits den Client, welcher an dieser Stelle nicht behandelt wird und allenfalls für eine Bachelorarbeit dann verwendet werden kann.
Der Servermodus wird von einem Administrator betreut, welcher auch die Konfiguration der Applikation vornimmt.

Im Folgenden wird der Prozessablauf grafisch dargestellt, wobei die Prozesse 3 und 4 iterativ über die ganze Datenbank ausgeführt werden:

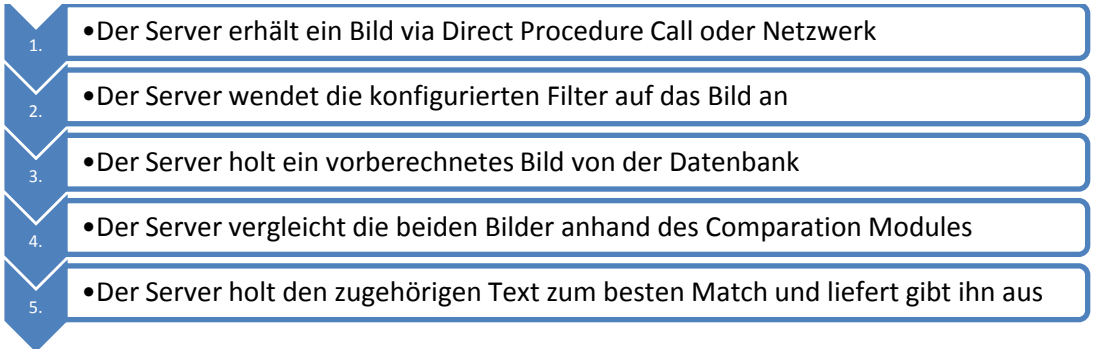



**Abbildung 1:**
**Ablauf der Verarbeitung eines eingehenden Bildes beim Server**

### 2.1.2. Ziele

- Der Server soll die ganze Bildverarbeitung realisieren, dies aus drei Gründen:
  a) Die Leistung der mobilen Geräte ist im Verhältnis zu dedizierten Computern relativ schwach.
  b) Die Portierung der Clients auf verschiedene Plattformen wird so um Grössenordnungen einfacher und schneller
  c) Die Verarbeitungskette vereinfacht sich, weil die speziell konfigurierten Filter nicht zuerst noch vom Server geholt werden müssen

- Der Server soll die Bilder zwecks Einfachheit in einer DB ablegen
- Die Filter in der Bildverarbeitungskette sollen dynamisch verwaltet werden können

### 2.1.3. Einschränkungen

- Der Fokus liegt zum jetzigen Zeitpunkt nicht auf Performance
- Die Ablage der Bilder in einer Datenbank kann bei grösseren Datenmengen suboptimal sein *(dieser Punkt wird unter Kapitel 1.5.2 noch ausführlicher behandelt)*

## 2.2. Systemstruktur

Für detailliertere Informationen über die Klassen schlagen Sie bitte unter ***[ 7 ] JavaDoc*** nach.

### 2.2.1. Physische Schicht

Der Server der Solution „eyedentificator“ kann in zwei verschiedenen Modi arbeiten:

- Regulärer Client / Server Betrieb mit Netzwerkintegration
- Server only zu Testzwecken, wobei Daten direkt per Methodenaufruf eingegeben werden

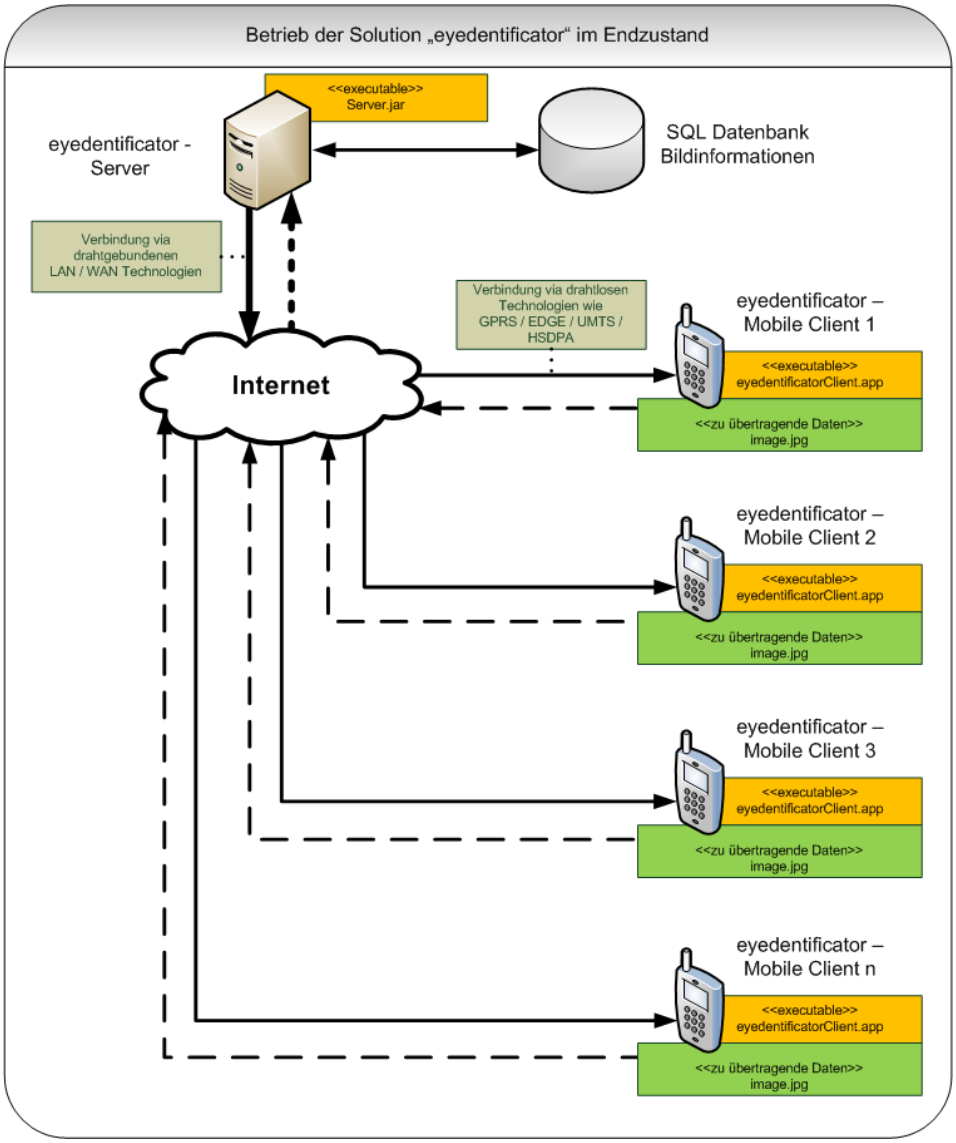



**Abbildung 2:**
**Deployment der Solution eyedentificator zu allen Aktoren**

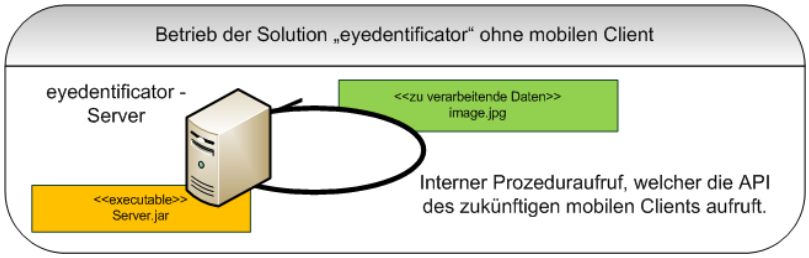



**Abbildung 3:**
**Deployment der Standalone-Solution eyedentificator**

### 2.2.2. Logische Sicht

#### *2.2.2.1. System eyedentificator*

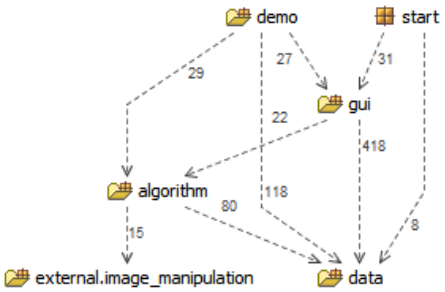



**Abbildung 4:**
**Paketstruktur von eyedentificator und deren Abhängigkeiten**

In der Abbildung 4 lassen sich die ineinander verzahnten Klassen gut erkennen. Das Package ***start*** verwendet das ***gui*** und einige Elemente aus dem ***data*** Bereich, wie zum Beispiel den Logger, oder den ResourceLoader zum Zugriff auf Dateien innerhalb des Projekts. Das ***gui*** verwendet die ***algorithm*** für die grafische Darstellung und ***data*** für das Laden von Icons und weiteren Elementen. Das Package ***algorithm*** verwendet noch einige externe Filter, welche unter dem Package ***external*** zu finden sind.

***Einige Beispiele zur Illustration:***

Dienste, welche vom Package ***gui*** angeboten werden:

- *Bildschirmauflösung liefern*
- *Fenster generieren und zeichnen*
- *Vorschau berechnen und anzeigen*

Dienste, welche vom Package ***data*** angeboten werden:

- *Zugriff auf Bild in Projekt*
- *Zugriff auf Datenbank*
- *Zugriff auf Konstanten und Textsnippets*

| Package | Inhalt |
|---|---|
| **start** | Starterklassen für JUnit Test Runner, Server & PC Client. Sie enthalten jeweils die Main-Methoden. ***[01] [02]***<br><br>*Enthaltene Klassen:*<br>*Server*<br>*Client_PC*<br>*JUnitTestRunner* |
| **gui** | Generiert GUI und verwaltet die Darstellung.<br><br>*Enthaltene packages:*<br>*generic*<br>*pc_client*<br>*server*<br><br>*siehe Kapitel: 1.2.2.3 für detailliertere Informationen* |

| | |
|---|---|
| **data** | Enthält alle Bereiche rund ums Datenmanagement.<br><br>*Enthaltene Packages:*<br>*access*<br>*adapter*<br>*annotation*<br>*collection*<br>*exception*<br>*factory*<br>*filterconfiguration*<br>*logging*<br><br>*siehe Kapitel 1.2.2.2 für detailliertere Informationen* |
| **demo** | Beinhaltet alle Demoprojekte. Sie sind in der Regel alle separat startbar; enthalten daher eine Mainmethode. Sie repräsentieren meist Proof-of-Concepts, d. h. sie wurden erstellt um zu testen, ob eine bestimmte Vorgehensweise funktioniert und ins Projekt eingegliedert werden kann. Sie werden nach der Übernahme in den Hauptcode nicht mehr aktiv weiter gepflegt. Es kann daher vorkommen, dass sie im Verlaufe des Projekts nicht mehr wie gewünscht funktionieren, da sich einige Schnittstellen und Klassen geändert haben. Diese Demos finden sich im Subpackage ***deprecated*** wieder.<br><br>*Enthaltene Packages:*<br>*comparation*<br>*comparation_camera*<br>*comparation_db*<br>*dbaccess*<br>*dbaccess_derby*<br><br>*#deprecated packages:*<br>*comparation_network*<br>*dbreader*<br>*dbwriter*<br>*dbviewer*<br>*fileaccess*<br>*filterchoosergui*<br>*servergui* |
| **algorithm** | Dieses Package beinhaltet alle Bildalgorithmen von eyedentificator. Dazu zählen die Filter, sowie die Comparation Modules, welche im Package ***comparator*** zu finden sind. Das Package ***generic*** enthält Helperklassen, welche z. B. BufferedImages komplett kopieren, oder etwa Alpharaster ausgeben bzw. sichern & wiederherstellen können. Das Package **filter** wird später noch detaillierter behandelt.<br><br>*Enthaltene Packages:*<br>*comparator*<br>*filter (siehe Kapitel 1.2.2.1 für detailliertere Informationen)*<br>*generic* |
| **external** | Dieses Package enthält alle externen Filter und Codefragmente, welche nicht selbst entwickelt, sondern kopiert wurden. Teilweise mussten dennoch einige Zeilen angepasst / ergänzt werden, damit sich die Klassen in diesem Projekt verwenden liessen. Im Grossen und Ganzen sind diese Anpassungen jedoch meist Kleinigkeiten. |
| **res** | Das Package res dient ausschliesslich der Bereitstellung von Ressourcen |

| | |
|---|---|
| | jeglicher Art. Beispielsweise werden Bilder, Profile, oder etwa Plugins darin gelagert. Das package ***images*** gliedert sich selbst wiederum in ***sample*** (= Beispielbilder zum Testen der Algorithmen), ***test*** (= Bilder für die JUnit-Tests) und in ***userinterface*** (= Bilder, welche vom UI verwendet werden). Das Package ***profiles*** enthält die vordefinierten Profile, welche zur Laufzeit geladen werden können. Das Package ***plugins*** enthält alle verwendeten Plugins und das Package ***xml*** enthält ein XML-Demofile zum Testen des XML-Parsers.<br><br>*Enthaltene Packages:*<br>*images*<br>*=> sample*<br>*=> test*<br>*=> userinterface*<br><br>*profiles*<br>*plugins*<br>*xml* |

Tabelle 1: Unterpakete des Paketstamms

#### *2.2.2.1. Package filter*

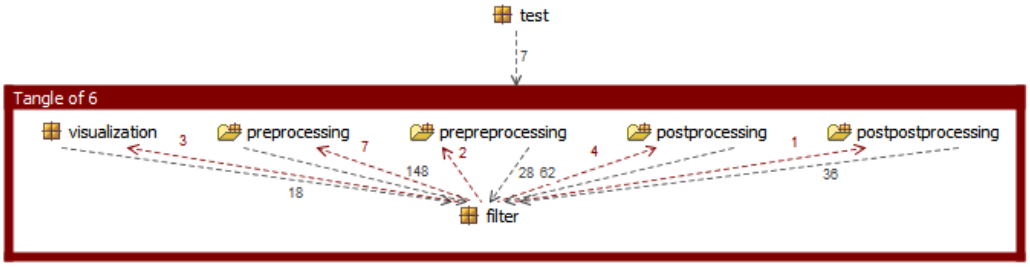



Abbildung 5:
Paketstruktur vom Package filter und deren Abhängigkeiten

Es scheint so, als ob eine umfangreiche Abhängigkeit zwischen den Filtern und der Filterbasisklasse besteht. Dem ist aber nicht so. Das Problem entsteht dadurch, dass das Package ***filter*** zwei Klassen enthält: Einerseits die eben erwähnte Basisklasse und andererseits noch die FilterType-Klasse, die den Typ eines Filters markiert. Alle konkreten Filterklassen verwenden natürlich ihre Basisklassen mit ihren Datenfeldern (beispielsweise den Pre- & Postfilterimages). Umgekehrt verwendet die Klasse FilterType – die im selben Package steckt wie die Basisklasse – jede Filterklasse, da sie die Fähigkeit hat, anhand ihres Types eine konkrete Klassenrepräsentation retour zu liefern. Somit kommt dieses Tangle zustande. Es wäre einfach zu beheben, in dem man die zwei Klassen in separate Packages auslagern würde (wurde überprüft und es würde funktionieren). Ich finde jedoch, dass die Übersichtlichkeit nicht unbedingt besser wird, wenn noch mehr versplitterte Packages entstehen.
Übrigens: Der gleiche Sachverhalt gilt auch für die Comparation Modules, welche technisch gesehen sehr ähnlich aufgebaut sind, sich jedoch in einigen wesentlichen Aspekten stark unterscheiden.

| Package | Inhalt |
|---|---|
| **test** | Enthält JUnit Testklassen zum Testen der Filterbasisklasse.<br><br>*Enthaltene Klassen:*<br>*Test_Filter* |
| **filter** | Enthält die Basisklasse der Filter und eine Klasse die den Typ eines Filters |

| | |
|---|---|
| | repräsentieren kann.<br><br>*Enthaltene Klassen:*<br>*Filter*<br>*FilterType* |
| **visualization** | Enthält Filter für die Visualisierung von (Zwischen-) Resultaten. Sie werden gleich wie „ausgewachsene“ Filter verwendet, mit dem kleinen Unterschied, dass das Ausgabebild eine Statistik mit Diagrammen darstellt und nicht wie bei den regulären Filtern ein verändertes Eingabebild.<br><br>*Enthaltene Klassen:*<br>*ColorHistogramGraph*<br>*HorizontalBlackGraph*<br>*VerticalBlackGraph* |
| **prepreprocessing** | Enthält Filter zur Vorverarbeitung von Bildern. Die Vorverarbeitung umfasst das Skalieren und Zuschneiden von Bildern. Dieses Package repräsentiert die Phase 1 Filterklassen.<br><br>*Enthaltene Klassen:*<br>*Crop*<br>*Stretch* |
| **preprocessing** | Enthält die Filter für die weitere Bearbeitung nach der Vorverarbeitung. Dieses Package repräsentiert die Phase 2 Filterklassen.<br><br>*Enthaltene Klassen:*<br>*ColorDifferenceWithBacklog*<br>*DifferenceNormalCurveOfDistribution*<br>*FreeFormSelect*<br>*GaussianBlur*<br>*Grayscale*<br>*Level*<br>*Sharpen* |
| **postprocessing** | Enthält die Filter für die weitere Bearbeitung nach der Preprocessing-Stufe. Dieses Package repräsentiert die Phase 3 Filterklassen.<br><br>*Enthaltene Klassen:*<br>*BlackAndWhiteOnly*<br>*BlackAreaRemover*<br>*Dot*<br>*InvertOnDemand* |
| **postpostprocessing** | Enthält die Filter für die weitere Bearbeitung nach der Postprocessing-Stufe. Dieses Package repräsentiert die Phase 4 Filterklassen.<br><br>*Enthaltene Klassen:*<br>*MoveBWImage* |

**Tabelle 2: Paket filter**

#### *2.2.2.2. Package data*

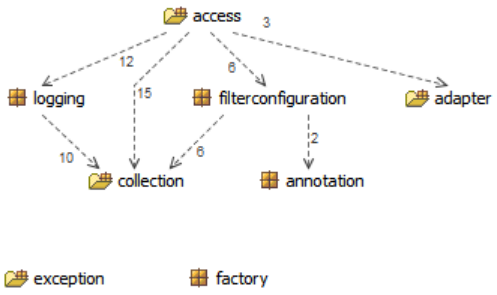



**Abbildung 6:**
**Paketstruktur vom Package data und deren Abhängigkeiten**

Das Package data bietet viele stark differenzierte Dienste an. Da das Projekt eyedentificator sehr breit aufgestellt ist, benötigt es auch viele unterschiedliche Technologien. Allen solchen Technologien gemein ist, dass sie Daten bereitstellen oder Daten weiterverarbeiten können. Daher wurde ein so abstraktes Konstrukt wie data eingeführt. Hier finden sich alle Klassen, welche nicht selbst aktiv werden und daher ausschliesslich von anderen Klassen als Diensterbringer benötigt werden. Dieses Package behandelt alle Anfragen zu Dateien, Datenbanken, Netzwerken, Collections und noch einige weitere mehr.

| Package | Inhalt |
|---|---|
| **access** | ***access*** enthält Zugriffsmöglichkeiten auf viele unterschiedliche Technologien. ***database*** erlaubt gewissermassen den Zugriff auf eine Datenbank (lesend & schreibend). ***filesystem*** ermöglicht das Laden von Dateien aus dem Projekt, sowie das Speichern von Dateien ausserhalb des Projekts. Dieses Speichern wird jedoch seit der Überarbeitung der Filterprofilverwaltung vorläufig nicht mehr verwendet. ***language*** enthält die Textschnipsel, sowie die Konstanten, welche im gesamten Projekt verwendet werden. Zu guter Letzt enthält das Package ***xml*** noch einen vollwertigen SAX XML-Parser. XML wurde anfangs in Betracht für die Speicherung von Profilen gezogen, jedoch wieder aufgrund der Komplexität verworfen. Es wird ebenso wie die Speicherung von Daten noch im aktiven Programm belassen, da in einer späteren Ausbaustufe diese Funktionen allenfalls eingesetzt werden (möglicherweise sinnvoll und professioneller wäre beispielsweise die Verwendung von XML zur Netzwerkkommunikation).<br><br>*Enthaltene Packages:*<br>*database*<br>*filesystem*<br>*language*<br>*network*<br>*xml* |
| **adapter** | Da die vorhandenen Reader / Writerklassen nicht ausreichten und die gesamte Verwaltung wesentlich erschwerten, wurden zwei Adapter geschaffen, welche jeweils für den Schreib- bzw. Leseprozess eingesetzt werden können. Die Besonderheit ist, dass sie sowohl zeilenweise Strings, als auch byte-weise schreiben / lesen können. Dies wird durch das für eyedentificator selbst definierte Protokoll gefordert.<br><br>*Enthaltene Klassen:*<br>*DataAndTextInputReader*<br>*DataAndTextOutputWriter* |

| | |
|---|---|
| **annotation** | Enthält die Annotation, welche jeder markierten Variable einen Wert zuweisen kann. Diese Annotation wird via Reflection ausgelesen und auch wieder gesetzt.<br><br>*Enthaltene Klassen:*<br>*VariableDefaultValue* |
| **collection** | Im Projekt werden an verschiedenen Orten Daten benötigt, welche logisch zusammengehören und auch immer zusammen benötigt werden. Um das Handling zu vereinfachen wurden Collection Klassen geschaffen, welche mehrere Elemente zu einem Objekt zusammenfassen. Die Generics Klassen speichern jeweils zwei (Tuple), oder drei (Triple) zusammengehörende Elemente. Die Typen der Elemente können via Java Generics beliebig definiert werden. Das DatabaseImagePacket enthält dagegen eine Zeile von Bilder in der Datenbank; das sind dann fünf Bilder (das Original und die vier Processingstufen).<br><br>*Enthaltene Klassen:*<br>*DatabaseImagePacket*<br>*GenericsDataTriple*<br>*GenericsDataTuple* |
| **exception** | Dieses Package enthält die Exceptionklasse, welche geworfen wird, falls das Eingabebild eine zu schlechte Qualität aufweisen sollte. Diese Funktionalität wird momentan nur vom FreeFormSelect-Filter implementiert.<br><br>*Enthaltene Klassen:*<br>*BadInputImageException* |
| **factory** | Dieses Package stellt Factory-Methoden für GridBagConstraints-Komponenten des GUI bereit.<br><br>*Enthaltene Klassen:*<br>*GridBagConstraintsFactory* |
| **exception** | Dieses Package enthält die Exceptionklasse, welche geworfen wird, falls das Eingabebild eine zu schlechte Qualität aufweisen sollte. Diese Funktionalität wird momentan nur vom FreeFormSelect-Filter implementiert.<br><br>*Enthaltene Klassen:*<br>*BadInputImageException* |
| **filterconfiguration** | ***filterconfiguration*** enthält die Logik für die Filterprofile. Ein Filterprofil besteht wiederum aus mehreren individuellen Filterkonfigurationen. Ausserdem hält sie eine Enumeration für die verschiedenen Filterphasen bereit.<br><br>*Enthaltene Klassen:*<br>*FilterConfiguration*<br>*FilterProfile*<br>*ProcessingType* |
| **logging** | |

| | |
|---|---|
| | Dieses Package stellt grundlegende Log-Mechanismen zur Verfügung. Einerseits via Konsolenausgabe, andererseits via Statistik im GUI.<br><br>*Enthaltene Klassen:*<br>*Logger*<br>*StatisticManager* |

**Tabelle 3: Paket data**

#### *2.2.2.3. Package gui*

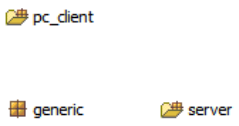



**Abbildung 7:**
**Paketstruktur vom Package gui und deren Abhängigkeiten**

Das Package ***gui*** scheint untereinander überhaupt keine Abhängigkeiten zu haben und das ist auch korrekt so, da die Elemente von ***gui*** durch die Starterklassen aufgerufen werden. Ausserdem sind die GUIs vom Client PC und Server strikte getrennt. Einzig der generische ScreenHelper kann beiderseits verwendet werden.

| Package | Inhalt |
|---|---|
| **pc_client** | Enthält alle Klassen die zum GUI des Clients gehören. Sie umfassen hauptsächlich die grafischen Klassen des JTabbedPane. Speziell ist das Serverdestination-JPanel, welches den Serverstandort für das VLC-Plugin in Erfahrung bringt.<br><br>*Enthaltene Klassen:*<br>*MainJTabbedPane*<br>*CameraImageAddJPanel*<br>*CameraImageComparisonJPanel*<br>*EntryCountJPanel*<br>*HandshakeJPanel*<br>*ImageAddJPanel*<br>*ImageComparisonJPanel*<br>*ImageRemoveJPanel*<br>*RetrieveImagesJPanel*<br>*RetrieveInfoJPanel*<br>*ServerDestinationJPanel*<br>*ValidEntriesJPanel* |
| **server** | Enthält alle Klassen die ausschliesslich zum GUI des Servers gehören. Das GUI des Servers besteht aus dem Hauptfenster, einem Filterauswahldialog, einem Filterkonfigurationsdialog und einem Statusbereich. Ausserdem sind die ***worker*** (bestehen aus ConnectionHandler und Analyzer) für die Verarbeitung von Bildern angegliedert, da die Bildverarbeitung und die Darstellung praktisch immer sehr stark korrelieren. Es wäre allenfalls dennoch sinnvoll die Worker auszulagern, jedoch gibt es keinen passenden Ort, und neue Packages mit nur wenigen Klassen führen wie zuvor schon erwähnt zu starker Zersplitterung, was auch suboptimal ist.<br><br>*Enthaltene packages:*<br>*filterconfiguration* |

| | |
|---|---|
| | *information*<br>*worker* |
| **generic** | Enthält eine Klasse für die Rückmeldung der Bildschirmauflösung.<br><br>*Enthaltene Klassen:*<br>*ScreenHelper* |

**Tabelle 4: Paket data**

## 2.3. Angewandte Software-Patterns

### 2.3.1. Factory *[03]*

Eine Factory wurde für die Erstellung von GridBagConstraints verwendet, da die Verwaltung derer Objekte relativ mühsam ist und schnell zu riesigen Parameterlisten führt. Sie wird nur an wenigen Stellen verwendet, weil an vielen Stellen sehr stark angepasste Constraints benötigt werden.

Anwendungsbeispiel:

```
GridBagConstraints c = GridBagConstraintsFactory.getInsettedNone(
        xPos, yPos, width, height,  xScale, yScale, fill);
```

Das hiermit generierte Constraints-Objekt enthält genullte Insets. Insets bestimmen den Randabstand zu den umgebenden Elementen im GUI.

### 2.3.2. Variation Pipe and Filters *[04]*

Die Filter, sowie die Comparation Modules können in diesem Projekt beliebig und mehrfach hintereinander gestellt werden. Die Filter werden logisch zu einer Pipe zusammengestellt, bevor sie dann seriell einer nach dem anderen abgearbeitet werden. Konkret holt die Klasse „Analyzer“ die Filter aus dem Profil und arbeitet sie dann Phase für Phase, Filter für Filter durch. Eine „reine“ Pipe and Filters Lösung kommt ohne externen Koordinator aus. In diesem Fall wurde die Lösung mit einem Iterator gelöst, welcher dann die geordneten Filter durchläuft. Der Grund dafür ist die gesteigerte Komplexität, welche von einer ausgewachsenen Pipe ausgeht: Jeder Übergang eines Filters zum nächsten würde wieder einen separaten Koordinator benötigen (welcher meist in einem separatem Thread asynchron liefe). Es wäre auch eine zweite Variante ohne Übergang mit direkten Methodenaufrufen denkbar. Diese Lösungen jedoch würden keinen allzu grossen Mehrwert bieten, da:

- nur ein Objektkopiervorgang eingespart würde und dieser aufgrund der vielen anderen nicht ins Gewicht fiele
- es wesentlich komplexer wäre die einzelnen Zwischenschritte zu beaufsichtigen, d. h. die erstellten Zwischenbilder auszulesen und geworfene Exceptions angemessen zu verarbeiten, da in diesem Fall den Filtern die entsprechende Logik / Übersicht fehlen würde

Generiertes Filterprofil im folgenden Anwendungsbeispiel:


**Abbildung 8:**
**Mögliche Filterpipeline eines Filterprofils**

Anwendungsbeispiel:

```
// Die essenziellen Aufrufe sind hervorgehoben, die restlichen Befehle könnten
// auch ohne grössere Einschränkungen weggelassen werden in diesem Fall.

// grüne Befehle gehören dem Prozess der Filterprofilkonfiguration an
// violette Befehle gehören dem Prozess des Bildvergleichs an

// Leeres Filterprofil wird erstellt
FilterProfile profile = new FilterProfile();

// Filter „Stretch“ wird hinzugefügt
```

```
profile.add(new
FilterConfiguration(FilterType.PREPREPROCESSING_STRETCH.concreteClass()));

// Filter „Sharpen“ wird hinzugefügt
profile.add(new
FilterConfiguration(FilterType.PREPROCESSING_SHARPEN.concreteClass()));

// Filter „FreeFormSelect“ wird hinzugefügt
profile.add(new
FilterConfiguration(FilterType.PREPROCESSING_FREEFORMSELECT.concreteClass()));

// Filter „BlackAndWhiteOnly“ wird hinzugefügt
profile.add(new
FilterConfiguration(FilterType.POSTPROCESSING_BLACK_AND_WHITE_ONLY.concreteCla
ss()));

// Comparation Module „Size2D“ wird hinzugefügt
profile.add(new FilterConfiguration(ComparatorType.SIZE_2D.concreteClass()));

// Analyzer wird erstellt für den vergleich, das Profil wird als Parameter
// mitgegeben. Das WaitingNull ist ein leeres (dummy) Status-JPanel.
Analyzer analyzer = new Analyzer(profile, (IWaiting) new WaitingNull());

// float Variable für den Übereinstimmungswert
float matchPercentage = 0.0f;

// Aufruf zum den Vergleich starten
matchPercentage = analyzer.compareImages(bufferedImage1, bufferedImage2);
```

Der hiermit vollzogene Vergleich skaliert zuerst ein Bild auf eine Einheitsgrösse, schärft es, stellt es vom Hintergrund frei, färbt es schwarz / weiss ein und vergleicht anschliessend die Fläche der Objekte in den Bildern. Die Filter werden - falls zusätzliche Angaben wie in diesem Beispiel fehlen - mit den Defaultkonfigurationen geladen. Die Reihenfolge der hinzugefügten Filter wird berücksichtigt, jedoch werden ungültige Kombinationen verhindert. Dies bedeutet, dass jede Kombination ohne grössere Konsequenzen durchlaufen kann, dies jedoch meist zu keinem sinnvollen Resultat führen wird. Beispielsweise macht es keinen Sinn ein Bild schwarz / weiss einzufärben, um anschliessend die Farben zu vergleichen. Andererseits können aber auch Filter, welche ein schwarz / weisses Eingabebild erwarten nicht viel an einem Farbbild ausrichten und reichen entweder das Bild unverändert durch, oder generieren ein komplett schwarzes bzw. weisses Bild.

### 2.3.3. Reflection *[05]*

Dieses Projekt macht regen Gebrauch von Reflection während der Filterprofilinstanziierung. Ursprünglich waren mehrere Varianten in Evaluation:

- **Lösung mit internen Datenstrukturen**, d.h. man setzt in jedem Filter einige Werte in seinem Container. Dieser Container wird dann zur Laufzeit vom Filter ausgelesen, analysiert und dementsprechend verhält sich dann der Filter.
  - Vorteil: sehr einfach zu implementieren
  - Vorteil: Persistenz praktisch geschenkt
  - Nachteil: jede Filterklasse enthält individuellen Code zur Parameterverarbeitung
  - Nachteil: jede Klasse muss zugelassene Parameter dokumentiert haben
  - Nachteil: massiv duplicated Code
  - Nachteil: Konfiguration liegt im Filtercode

- **Lösung mit Reflection** ***(gewählter Ansatz)***, d. h. man markiert alle konfigurierbaren Variablen mit einer Annotation und definiert den default-Value in der Annotation. Nun werden zur Laufzeit via Reflection die markierten Variablen des Filters ausgelesen und zur Konfiguration angeboten. Nach dem Bestätigen der neuen Konfiguration werden die

Parameter wieder aus dem GUI ausgelesen und im Filterprofil in die jeweilige Filterkonfiguration geschrieben.

- Vorteil: alle Filter werden von genau einem Reflection-Codeblock bedient
- Vorteil: es können zur Laufzeit keine falschen Typen zugewiesen werden
- Vorteil: es muss keine Dokumentation erstellt werden; Eclipse zeigt sie an
- Vorteil: Konfiguration liegt ausserhalb des Filtercodes in der Konfiguration
- Nachteil: komplex zu implementieren
- Nachteil: Reflection einzusetzen ist in der Regel nicht elegant

Grafische Darstellung des Aufbaus eines Filterprofile-Objekts:

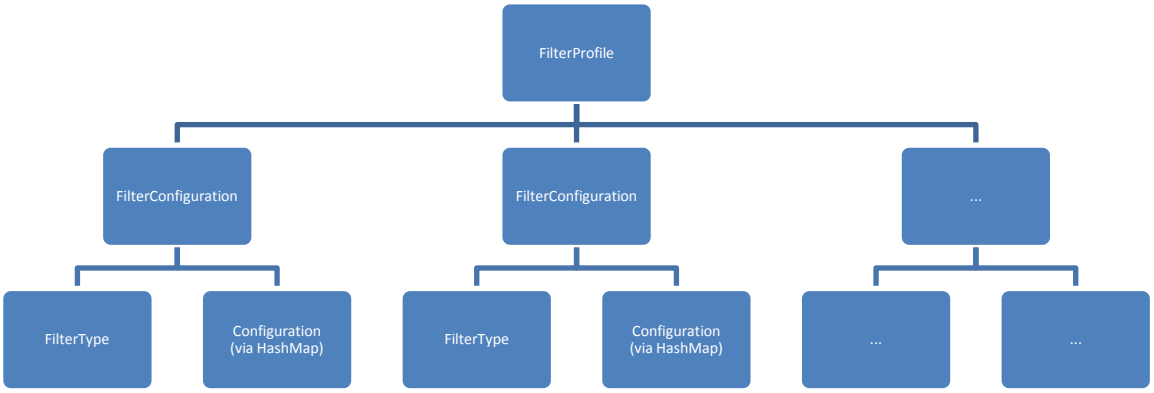



**Abbildung 9:**
**Interner Aufbau eines FilterProfile-Objekts**

Anwendungsbeispiel:

```
@VariableDefaultValue(value = "800")
public int _xSize;
@VariableDefaultValue(value = "600")
public int _ySize;
```

Mit diesem Beispiel wird der default-Value dieser zwei Variablen definiert, zusätzlich werden sie markiert für die Reflectionlogik. Somit erscheinen diese zwei Variablen wie von Geisterhand im Filterkonfigurationsdialog im GUI, ohne sonstiges zutun. Dies macht die Freigabe von Variablen äusserst einfach: Variablen deklarieren, obige Annotation darüber und fertig. Keine weiteren Änderungen sind dafür notwendig. Ausserdem funktioniert damit auch gleich die Persistenz, d. h. die so markierten Variablen werden automatisch gesichert beim Speichern eines Profils. Dieses Konzept der Persistierung wird später unter dem Kapitel Architekturkonzepte umfangreicher erläutert.

## 2.4. Process View

### 2.4.1. Handling eines asynchronen Requests via Netzwerk

Die Netzwerkanfragen werden über ein Serversocket entgegengenommen, welches für jede Anfrage einen neuen Thread mit der Antwortlogik generiert und startet. Diese Lösung sichert einerseits die Leistung bei vielen Verbindungen, andererseits führt ein Crash ausgelöst durch eine fehlerhafte Anfrage nur den betreffenden Thread in den Abgrund und der Server läuft trotzdem weiter. Es sei an dieser Stelle vermerkt, dass der Server während diesem Projekt noch nicht für den heavy-use gedacht ist. Dies lässt sich unter anderem am Umstand erkennen, dass in der momentanen Situation zwar wie beschrieben mehrere Requests gleichzeitig abgearbeitet werden können, dies jedoch technisch nicht durch das gesamte Projekt durchgezogen wurde. Konkret stellt die Klasse Analyzer den Flaschenhals dar, denn sie erledigt den gesamten Vergleich und wird für alle folgenden Vergleiche verwendet. Somit werden bei zwei oder mehr gleichzeitigen Zugriffen auf den Analyzer die Datenfelder innerhalb der Klasse Analyzer korrumpiert. Bei ausschliesslichen Datenbankabfragen oder lesenden Serverrequests tritt dieses Problem natürlich nicht auf. Es sind ausschliesslich die Funktionen „AddImage“ und „CompareImage“ betroffen.

Im Folgenden ist der Codepfad beim Server aufgezeigt, welcher beim Eingang eines „image_comparison_ack“-Requests stattfindet:

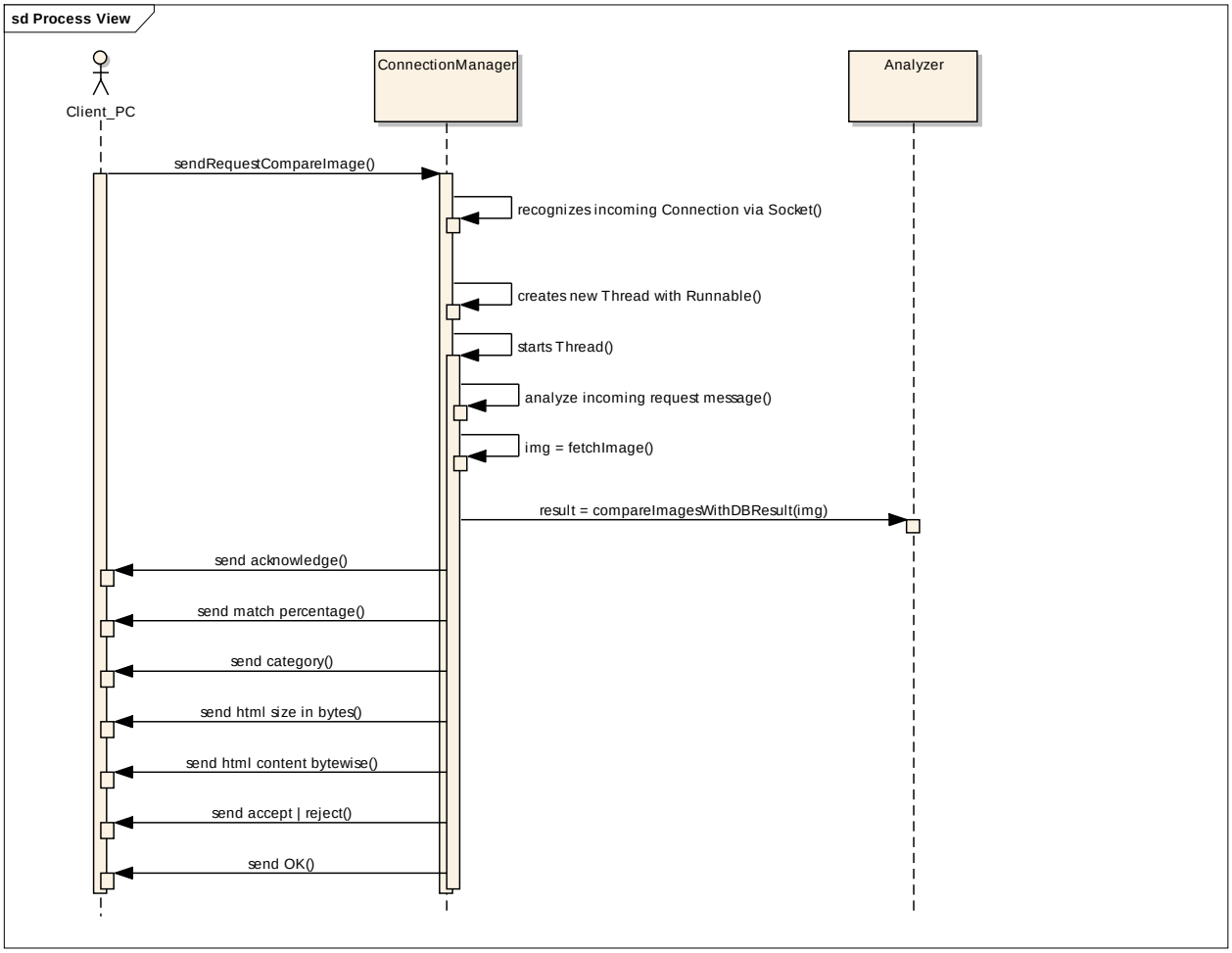
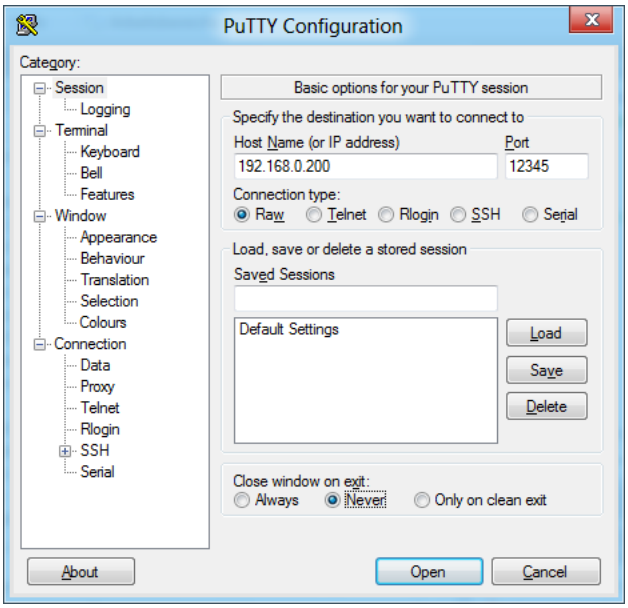


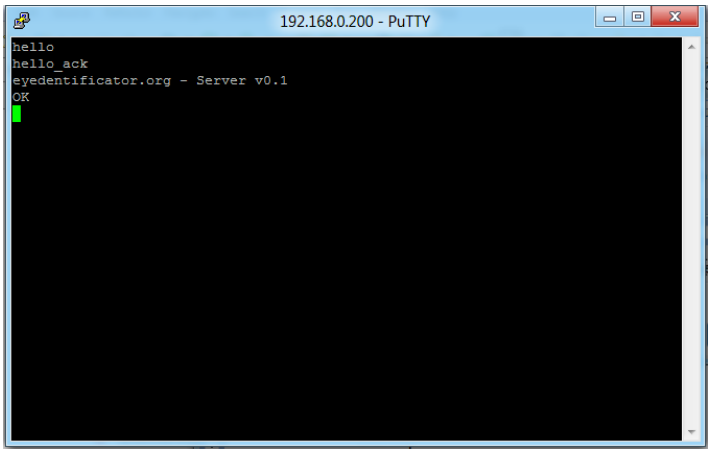

**Abbildung 10:**
**Process View des Netzwerkthreads**

## 2.5. Architekturkonzepte

### 2.5.1. Protokoll

Für dieses Projekt wurde ein eigenes Protokoll entwickelt. Auch hier wären Kommunikationsvarianten via XML, oder reinen binär Streams möglich gewesen, doch fiel der Entscheid für eine gemischte Daten / Textkommunikation. Diese bietet den Vorteil, dass sie sehr einfach zu debuggen ist (human-readable) und dennoch Binärdaten eines Bildes verwalten kann. Somit können einfache Requests ohne Binärdaten sogar mit einem Telnet-RAW Tool wie Putty aufgerufen werden (siehe Screenshots).




**Abbildungen 11, 12:**
**Putty Konfiguration (links), Antwort des Servers zu einem „Hello“-Request (rechts)**

***Protokolldefinition***

| ***Handshake*** | | |
|---|---|---|
| **Request** | | |
| *Nachricht* | *Typ* | *Zweck* |
| hello | String-Zeile | informiert Server über Anfrage-Typ |
| **Response** | | |
| *Nachricht* | *Typ* | *Zweck* |
| hello_ack | String-Zeile | informiert Client über Antwort-Typ |
| eyedentificator.org - Server v0.1 | String-Zeile | zeigt Server Version |
| OK | String-Zeile | beendet Kommunikation |

| ***Comparison*** | | |
|---|---|---|
| **Request** | | |
| *Nachricht* | *Typ* | *Zweck* |
| image_comparison | String-Zeile | informiert Server über Anfrage-Typ |

| | | |
|---|---|---|
| 42 | String-Zeile | teilt dem Server mit, wie viele Bytes das folgende Datenpaket umfasst (ein int-Wert)[1] |
| 42 24 42 24 42 … | byte[ ] | Bild als JPEG / PNG / BMP als byte[ ] byte-wise übertragen |
| **Response** | | |
| *Nachricht* | *Typ* | *Zweck* |
| image_comparison_ack | String-Zeile | informiert Client über Antwort-Typ |
| 5.0f | String-Zeile | zeigt Matchpercentage des Matches |
| Computer | String-Zeile | zeigt Kategorie des Matches |
| 42 | String-Zeile | teilt dem Client mit, wie viele Bytes das folgende Datenpaket umfasst (ein int-Wert) |
| <html><head>my content</head>… | byte[ ] | HTML-Resultat des Matches als byte[ ] byte-wise übertragen |
| accept | String-Zeile (accept \| reject) | teilt dem Client mit, ob das eingesandte Bild eine ausreichende Qualität aufwies |
| OK | String-Zeile | beendet Kommunikation |

| ***Add Image*** | | |
|---|---|---|
| **Request** | | |
| *Nachricht* | *Typ* | *Zweck* |
| image_add | String-Zeile | informiert Server über Anfrage-Typ |
| Computer | String-Zeile | zeigt Kategorie des Entries |
| 42 | String-Zeile | teilt dem Server mit, wie viele Bytes das folgende Datenpaket umfasst (ein int-Wert) |
| 42 24 42 24 42 … | byte[ ] | HTML des Entries als byte[ ] byte-wise übertragen |
| 42 | String-Zeile | teilt dem Server mit, wie viele Bytes das folgende Datenpaket umfasst (ein int-Wert) |
| 42 24 42 24 42 … | byte[ ] | Bild als JPEG / PNG / BMP als byte[ ] byte-wise übertragen |
| **Response** | | |

[1] Auf den ersten Blick mag es erstaunlich wirken, warum dem Server die nachfolgend gesendete Rohdatengrösse mitgeteilt werden muss. Es wird jedoch schnell klar, dass der Server das Ende der Rohdaten nicht selbst erkennen kann, weil diese ja einen beliebigen Inhalt annehmen können. Es ist somit unmöglich mit einem Marker zu arbeiten. Diese „best-practice“ mit Countern wird auch von vielen aktuellen Datenprotokollen und -containern verwendet (Ethernet, ZIP-Komprimierung, u.v.m.).
Dieses Verfahren mit den Bytecounts wird in diesem Projekt zur binären Übermittlung von HTML und Bildern verwendet.

| *Nachricht* | *Typ* | *Zweck* |
|---|---|---|
| image_add_ack | String-Zeile | informiert Client über Antwort-Typ |
| accept | String-Zeile (accept \| reject) | teilt dem Client mit, ob das eingesandte Bild eine ausreichende Qualität aufwies |
| OK | String-Zeile | beendet Kommunikation |

| ***Remove Image*** | | |
|---|---|---|
| **Request** | | |
| *Nachricht* | *Typ* | *Zweck* |
| image_remove | String-Zeile | informiert Server über Anfrage-Typ |
| 42 | String-Zeile | zeigt Entry-ID des zu löschenden Datenbankeintrags |
| **Response** | | |
| *Nachricht* | *Typ* | *Zweck* |
| image_remove_ack | String-Zeile | informiert Client über Antwort-Typ |
| OK | String-Zeile | beendet Kommunikation |

| ***Number of Entries in Database*** | | |
|---|---|---|
| **Request** | | |
| *Nachricht* | *Typ* | *Zweck* |
| image_entry_count | String-Zeile | informiert Server über Anfrage-Typ |
| 42 | String-Zeile | zeigt Entry-ID des zu löschenden Datenbankeintrags |
| **Response** | | |
| *Nachricht* | *Typ* | *Zweck* |
| image_entry_count_ack | String-Zeile | informiert Client über Antwort-Typ |
| 42 | String-Zeile | repräsentiert die Anzahl der Einträge in der Datenbank |
| OK | String-Zeile | beendet Kommunikation |

| ***View Image-Entry from Database*** | | |
|---|---|---|
| **Request** | | |
| *Nachricht* | *Typ* | *Zweck* |
| image_retrieve_id | String-Zeile | informiert Server über Anfrage-Typ |
| 42 | String-Zeile | zeigt Entry-ID des gesuchten |

| | | Datenbankeintrags |
|---|---|---|
| **Response** | | |
| *Nachricht* | *Typ* | *Zweck* |
| image_retrieve_id_ack | String-Zeile | informiert Client über Antwort-Typ |
| 42 | String-Zeile | teilt dem Server mit, wie viele Bytes das folgende Datenpaket umfasst (ein int-Wert) |
| 42 24 42 24 42 … | byte[ ] | Bild #1 als JPEG / PNG / BMP als byte[ ] byte-wise übertragen (original) |
| 42 | String-Zeile | teilt dem Server mit, wie viele Bytes das folgende Datenpaket umfasst (ein int-Wert) |
| 42 24 42 24 42 … | byte[ ] | Bild #2 als JPEG / PNG / BMP als byte[ ] byte-wise übertragen (nach Phase 1) |
| 42 | String-Zeile | teilt dem Server mit, wie viele Bytes das folgende Datenpaket umfasst (ein int-Wert) |
| 42 24 42 24 42 … | byte[ ] | Bild #3 als JPEG / PNG / BMP als byte[ ] byte-wise übertragen (nach Phase 2) |
| 42 | String-Zeile | teilt dem Server mit, wie viele Bytes das folgende Datenpaket umfasst (ein int-Wert) |
| 42 24 42 24 42 … | byte[ ] | Bild #4 als JPEG / PNG / BMP als byte[ ] byte-wise übertragen (nach Phase 3) |
| 42 | String-Zeile | teilt dem Server mit, wie viele Bytes das folgende Datenpaket umfasst (ein int-Wert) |
| 42 24 42 24 42 … | byte[ ] | Bild #5 als JPEG / PNG / BMP als byte[ ] byte-wise übertragen (nach Phase 4) |
| OK | String-Zeile | beendet Kommunikation |

| ***Valid Entry-IDs in Database*** | | |
|---|---|---|
| **Request** | | |
| *Nachricht* | *Typ* | *Zweck* |
| valid_entries | String-Zeile | informiert Server über Anfrage-Typ |
| **Response** | | |
| *Nachricht* | *Typ* | *Zweck* |
| valid_entries_ack | String-Zeile | informiert Client über |

| | | |
|---|---|---|
| | | Antwort-Typ |
| [1, 42, 105, … , 213] | String-Zeile | repräsentiert alle gültigen Entry-IDs in der Datenbank, separiert durch ein Komma |
| OK | String-Zeile | beendet Kommunikation |

| ***View Specific Entry from Database*** | | |
|---|---|---|
| **Request** | | |
| *Nachricht* | *Typ* | *Zweck* |
| retrieve_info | String-Zeile | informiert Server über Anfrage-Typ |
| 42 | String-Zeile | teilt dem Server mit, zu welcher Entry-ID die Informationen beschafft werden sollen |
| **Response** | | |
| *Nachricht* | *Typ* | *Zweck* |
| retrieve_info_ack | String-Zeile | informiert Client über Antwort-Typ |
| Computer | String-Zeile | zeigt Kategorie des Entries |
| 192.168.0.42 | String-Zeile | zeigt IP, von der das Bild hinzugefügt wurde |
| 12:42:56 | String-Zeile | zeigt Uhrzeit, an der das Bild hinzugefügt wurde |
| 26.12.2011 | String-Zeile | zeigt Datum, an der das Bild hinzugefügt wurde |
| 42 | String-Zeile | teilt dem Client mit, wie viele Bytes das folgende Datenpaket umfasst (ein int-Wert) |
| 42 24 42 24 42 … | byte[ ] | HTML des Entries als byte[ ] byte-wise übertragen |
| OK | String-Zeile | beendet Kommunikation |

Tabellen 5 - 12: Protokolldetails

### 2.5.2. Datenbank

Die Datenbankanbindung hat noch ausserordentlich viel Optimierungspotential. Die gewählte Implementierung fusst auf BLOBs[2], welche weder portabel noch besonders schnell in der Verarbeitung sind. Der wesentliche Vorteil besteht in der Flexibilität und der Einfachheit: Es lassen sich sämtliche Filterzusammenstellungen dynamisch anwenden, man ist also nicht auf ein bestimmtes Filterprofil angewiesen. Ausserdem müssen keine Intermediateformate verwendet werden, um die Portabilität gewährleisten zu können und dies führt automatisch zu schlankerem Code. Allgemein stellt sich hier die Frage wie portabel ein Server überhaupt sein muss, denn ein Server wird ja schliesslich nicht jeden Tag gezügelt.
Wenn man nun den Gedanken etwas weiter spinnt, wird einem schnell klar, warum die Performance von dieser Generik sehr stark leidet, ich würde sogar so weit gehen und sagen, dass generische Methoden die zur Laufzeit auch noch auf Typenänderungen reagieren müssen im Allgemeinen einen der stärkst möglichen Impacts auf die Performance aufweisen. Dies wird nicht nur durch die entsprechend notwendige, sehr umfangreiche Logik zurück zu führen sein, sondern auch durch das teilweise Umwandeln der Typen.

[2] Binary Large Objects, das sind serialisierte (Java) Objekte.

Nun aber zurück zum Thema. Angenommen man konstruiert ein Profil mit einem Farbvergleich. Für diesen Vergleich müsste eigentlich einzig und allein ein int-Wert gespeichert werden, bzw. eine int[ ]-Repräsentation, welche die Farbverteilung im Bild handhabt. Das Original, sowie alle Zwischenbilder müssten nicht gespeichert werden. Es würden nur noch einige zusätzliche Metadaten benötigt, damit auch eine Antwort retour geliefert werden kann. Im Datenbankcode wäre nun also eine Abfrage hard-coded, welche genau nach einem int-Wert oder dessen Range sucht und dann die Metadaten retour liefert. Es müsste somit kein Bild mehr in der Datenbank abgelegt mehr werden (keine BLOBs) und der Vergleich könnte aufgrund von Zahlenvergleichen stattfinden, wäre so gesehen also sehr schnell.
Falls in dieser Situation nun der Benutzer des Servers ein anderes Profil zusammenstellt, vorzugsweise mit einer anderen Vergleichsmethode wie der Farbverteilung in bestimmten Regionen, müssten alle Originalbilder in der Datenbank nochmals neu durchgearbeitet werden. Dies wäre nun auf Basis der in der Datenbank vorhandenen Informationen gar nicht mehr möglich.
Somit fiel der Entscheid auf ein etwas generischeren Ansatz, welcher sehr dynamisch eingesetzt werden kann und ausserdem verhältnismässig leicht zu implementieren ist. Dies führt zu folgender Datenstruktur für jeden Entry:

- Eindeutiger Index als Zahl (DB-intern wahrscheinlich Long)
- Originalbild als BufferedImage BLOB
- Bild nach Phase 1 als BufferedImage BLOB
- Bild nach Phase 2 als BufferedImage BLOB
- Bild nach Phase 3 als BufferedImage BLOB
- Bild nach Phase 4 als BufferedImage BLOB
- Kategorie als String
- IP des Bilderzeugers als CIDR
- Zeit zu der das Bild damals hochgeladen wurde als Time
- Datum zu dem das Bild damals hochgeladen wurde als Date
- HTML Antwort, welche zum Bild gehört als TEXT (= „grosser String“)

Diese Tabelle wird durch das folgende SQL-Skript erzeugt beim initialen Start des Servers.

```
CREATE TABLE images (
  id SERIAL PRIMARY KEY,
  img1 bytea,
  img2 bytea,
  img3 bytea,
  img4 bytea,
  img5 bytea,
  kategorie VARCHAR(30),
  von_ip cidr,
  initialTime time DEFAULT 'now',
  initialDate date DEFAULT 'now',
  html_response TEXT
);
```

Alternativ das Skript für den Aufbau der Apache Derby Datenbank:

```
CREATE TABLE IMAGES (
  ID INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY
      (START WITH 1, INCREMENT BY 1),
  IMG1 BLOB(32M),
  IMG2 BLOB(32M),
  IMG3 BLOB(32M),
  IMG4 BLOB(32M),
  IMG5 BLOB(32M),
```

```
  CATEGORY VARCHAR(30),
  FROM_IP VARCHAR(18),
  INITIAL_TIME TIME DEFAULT CURRENT_TIME,
  INITIAL_DATE DATE DEFAULT CURRENT_DATE,
  HTML_RESPONSE VARCHAR(10000)
)
```

*Das Skript für Apache Derby musste leicht angepasst werden, da*

- *nur Grossschreibung bei Tabellen- und Spaltenbezeichnungen akzeptiert wird*
- *die Syntax für automatisch generierte Zahlenreihen unterschiedlich ist*
- *mehrere Datentypen von PostgreSQL nicht existieren*

Durch diese Lösung wird es jederzeit möglich das zugrundeliegende Filterprofil zu ändern, da alle benötigten Informationen für die Neuverarbeitung vorliegen. Es wird also bei einem Wechsel das Originalbild jedes Entries geholt, die Filter angewendet und das Resultat wieder in die Datenbank als BLOB gespeichert. Zwei Fragen bleiben noch offen:

*Warum sind vier Bilder in der Datenbank, wogegen eigentlich mindestens zwei reichen müssten?*

-und-

*Warum wird das Resultat wieder als Bild und nicht als Zahlenwert, respektive primitiver Datentyp gespeichert?*

Nun die Fragen lassen sich wie folgt beantworten: Es sind alle Bilder in der Datenbank gespeichert, weil gewisse Vergleichsmodule Zwischenergebnisse benötigen. Ausserdem können mehrere Filter miteinander kombiniert werden. Das bedeutet, dass es möglich ist einen Farbvergleich zusammen mit einem Vergleich der Kanten zu starten. Der Farbvergleich braucht noch alle Farbinformationen (= Bild nach Ende der Phase 1), wohingegen der Kantenvergleich die Kanteninformationen benötigt und diese sind erst nach der Phase 2 verfügbar. Somit ist es zwingend notwendig alle Zwischenbilder in der Datenbank abzulegen, damit diese nicht bei jedem Vergleich hergeleitet werden müssen.

Zur zweiten Frage lässt sich die Generik als Antwort herholen. Ein Kantenvergleich braucht eben Kanten und keinen int-Wert der die Farbenrepräsentiert. Es lässt sich natürlich auch hier anmerken, dass Kanten auch durch int’s und Co. ausgedrückt werden können, dies wird in grossen Produktivsystemen sicher auch so eingesetzt. Die sinnvolle Abbildung der Kanten auf einen aussagekräftigen Integerwert ist jedoch eine Wissenschaft für sich und lässt sich daher so relativ elegant umschiffen. Dies geht natürlich zu Lasten der Performance, aber wie gesagt, dieser Server soll eher zum Experimentieren einladen, als heavy-use verantworten können. Denn falls er später doch einmal unter heavy-use eingesetzt würde, müsste entweder ein Grossteil der dynamischen Funktionalität hard-coded werden, dann bliebe die Komplexität wahrscheinlich in etwa gleich, oder die Dynamik beibehalten werden und das Framework massiv erweitert werden. Dies würde aber zu einer um Faktoren höheren Komplexität führen und dies wäre in diesem Projekt untragbar.

### 2.5.3. Persistenz der Filterprofile

Die Filterprofile in diesem Projekt können auf die Festplatte gespeichert, bzw. davon geladen werden. Um dieses Verfahren zu implementieren wurden zwei verschiedene Lösungsansätze bedacht:

- **Lösung mit XML**: Die Konfiguration wird in ein XML-File gespeichert und dann auch wieder ausgelesen. Dies wäre eine sehr schöne Lösung, welche aber viel Overhead durch den XML-Parser und deren Ansteuerung mit sich bringt.
  - Vorteil: standardisiert, human-readable
  - Nachteil: sehr komplex, da mehrere Filter in einem File gespeichert wären

- **Lösung mit Serialisierung *(gewählter Ansatz)***: Die Konfiguration wird einfach als Java Objekt serialisiert auf die Festplatte geschrieben.
  - Vorteil: sehr einfach zu implementieren
  - Vorteil: Persistenz praktisch geschenkt
  - Vorteil: kann nicht einfach extern modifiziert werden
  - Nachteil: fehleranfällig bei häufigen Codeänderungen
  - Nachteil: Java-Profile nicht portabel auf andere Plattformen wie .Net

Es wurde die Variante mit der Serialisierung ausgewählt, da sie relativ einfach zu implementieren ist und aus Benutzersicht das genau gewünschte Layout bietet. Es wird somit bei Speichern einfach das aktuell verwendete FilterProfile Objekt serialisiert gespeichert und auch wieder geladen.

## 2.6. Sequenzdiagramme

### 2.6.1. Initialisierungspfad Starten des Servers

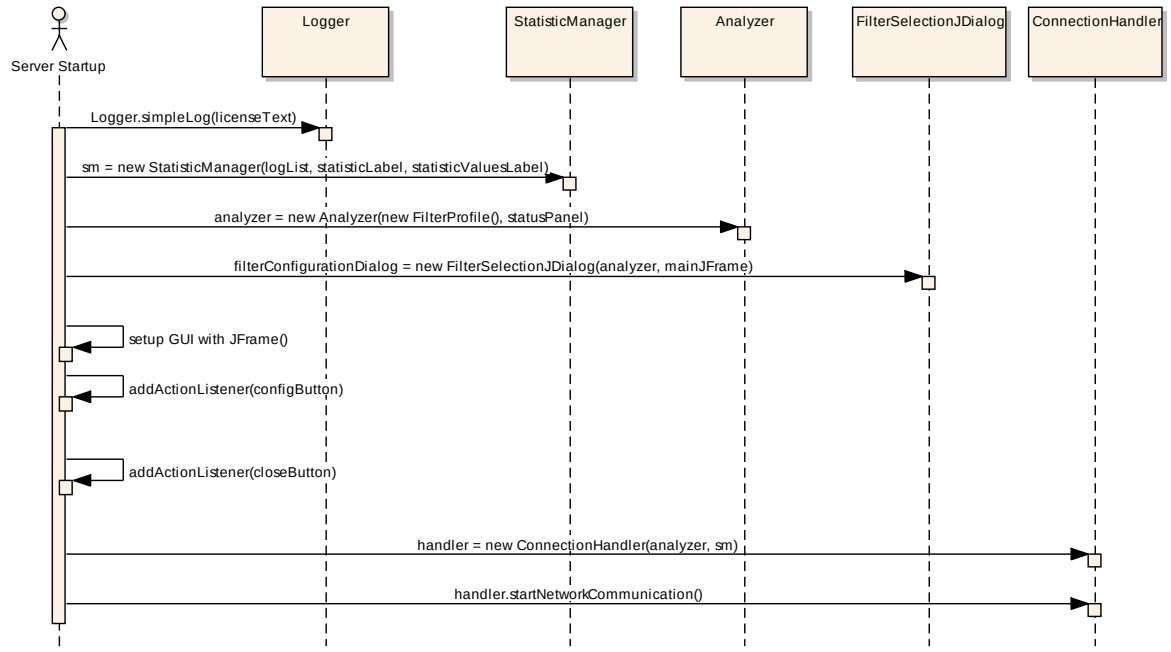



**Abbildung 13:**
**Initialisierungspfad Startvorgang des Servers**

Der Startvorgang lässt sich in folgende Bereiche unterteilen:

1) ***Logger für Konsolenoutput verwenden***
   Der Logger wird verwendet um die Lizenzbedingungen auf der Konsole auszugeben. Er wird statisch angesteuert.

2) ***StatisticManager initialisieren***
   Der StatisticManager kümmert sich um die Darstellung von Statistiken. Die Statistiken werden im Hauptfenster des Servers auf der rechten Seite angezeigt. Bei jeder Verbindungsanfrage werden die Werte neu berechnet und angezeigt.

3) ***Analyzer initialisieren***
   Der Analyzer erhält beim Start ein leeres Filterprofil. Der Analyzer bekommt weiter Zugriff auf das Statuspanel im unteren Bereich des Server-Hauptfensters. Der Analyzer verwendet dieses Panel um es mit neuen Informationen zu versorgen, denn der Analyzer ist Information Expert über ablaufenden Bildverarbeitungsprozesse.

4) ***FilterSelectionJDialog initialisieren***
   Dieser JDialog ermöglicht die Zusammenstellung von Filtern, welche dann zusammen ein Filterprofil ergeben. Er erhält einen Analyzer, welcher in sich ein Filterprofil enthält. Somit wird vom ganzen Server ein einziges Filterprofil verwendet. Der Grund für die Verwendung eines zentralen Filterprofils liegt darin, dass Änderungen im Konfigurator sich gleich unmittelbar überall im Server durchschlagen und somit keine „unterschiedlichen“ Filterprofile im Server rumgeistern. Durch diese Implementierung führt der Konfigurator nachteiligerweise zu einem kleinen Nadelöhr in der asynchronen Verarbeitung. Dies wird aber in einem kleineren LAN mit einigen Dutzend Bildern in der Datenbank nur in den seltensten Fällen bemerkt, da das Filterprofil nur als Referenzkonfiguration geholt wird, um die zu verwendende Filterpipeline im Analyzer aufzusetzen. Dies ist ein im Vergleich zur Bildverarbeitung sehr schlanker Prozess. Alternativ könnten threadsafe Versionen erstellt werden, dafür müsste aber eine zentrale

Instanz diese verschiedenen Profile managen und dies ist für das Projekt im derzeitigen Status overkill.

5) ***GUI zusammenstellen inkl. Listener anhängen***
Anschliessend wird das GUI des Hauptfensters zusammen gewürfelt. Es enthält mehrere Knöpfe und Labels. Ausserdem zeigt es Meldungen über erfolgreiche Verbindungen von Clients. Auch Fenster-Icons, Grösse, sowie weitere Eigenschaften werden definiert.

6) ***ConnectionHandler initialisieren und starten***
Zum Schluss - wenn die ganze Serverlogik betriebsbereit ist - wird noch der Netzwerklistener initialisiert. Dieser erhält dann einerseits den StatisticManager, welcher bei jeder Verbindung mit aktuellen Informationen versorgen kann. Ausserdem erhält er den Analyzer, welcher dann für Vergleiche verwendet wird. Nach dem Instanziieren des Handlers wird er gestartet. Ab diesem Zeitpunkt wird dann auf dem Netzwerkport (TCP) 12345 nach Anfragen von Clients gehorcht.

### 2.6.2. Comparation Prozess

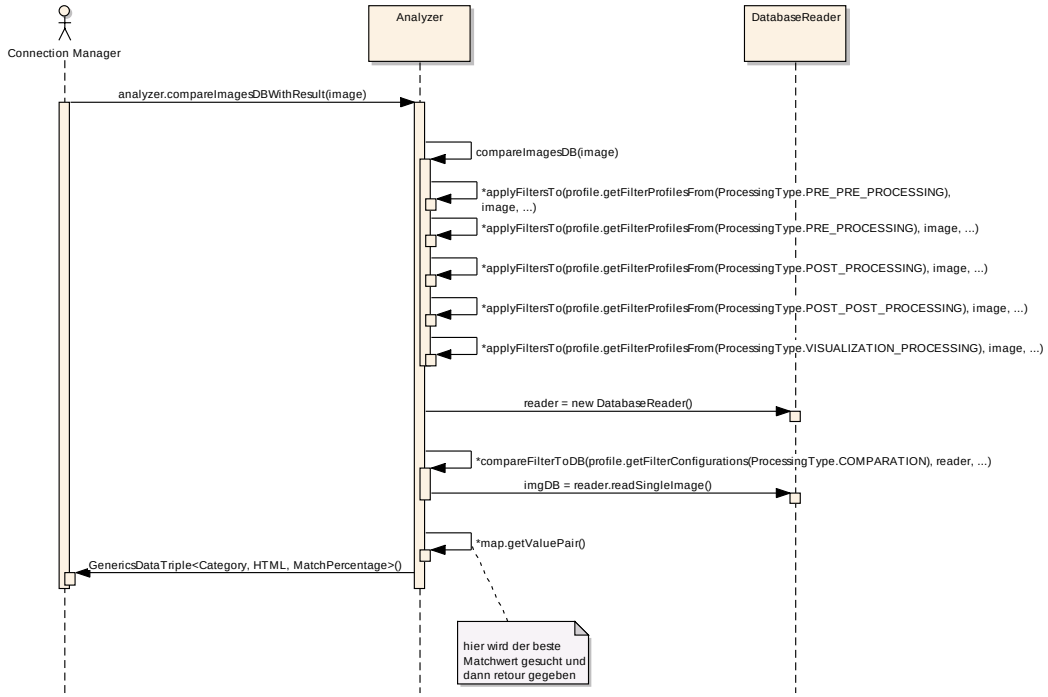



**Abbildung 14:**
**Programmfluss Comparationprozess**

Randnotiz: Der Programmfluss verwendet nicht mehrere Threads innerhalb des Analyzers. Es wurde im Diagramm lediglich der parallele Programmfluss gewählt, um die Zuständigkeiten der Methoden besser visualisieren zu können. Der Programmfluss lässt sich nun in folgende Bereiche unterteilen:

1) ***Konfiguriertes Filterprofil auf eingehendes Bild anwenden***
Es werden der Reihe nach die Filter der Phasen 1, 2, 3, 4 und Visualization angewendet. Visualization ist nur für Debugging-Zwecke, daher wird dieser Output im normalen Betrieb nicht verwendet. Bei den Phasen wird jeweils das Endresultat zwischengespeichert.

2) ***Bilder von Datenbank holen***
Es wird in der Datenbank das dem Comparation Module entsprechende Bild aus den Phasen 1 - 4 geholt. Dieser Schritt wird zusammen mit 3) für jeden Eintrag in der Datenbank

wiederholt. Ein Eintrag der Datenbank enthält immer auch eine eindeutige und zugewiesene ID und entspricht einer Zeile in der Tabelle der Datenbank.

3) ***Beide Bilder aufgrund des Filterprofils vergleichen***
Der Analyzer vergleicht nun die beiden in Relation stehenden Bilder auf Ähnlichkeiten aufgrund der im Filterprofil definierten Comparation Modules.

4) ***Resultat einer Map hinzufügen und auswerten***
Das Ergebnis dieses Vergleichs wird anschliessend einer Map hinzugefügt, welche dann nach Abschluss des Vergleichsprozesses die Wahrscheinlichkeiten aller in der Datenbank vorhandenen Bilder enthält. Nun wird der am bestpassendste Wert gesucht und dem Aufrufer mit den zugehörigen Metainformationen retour geliefert.

### 2.6.3. Datenbank Update Prozess

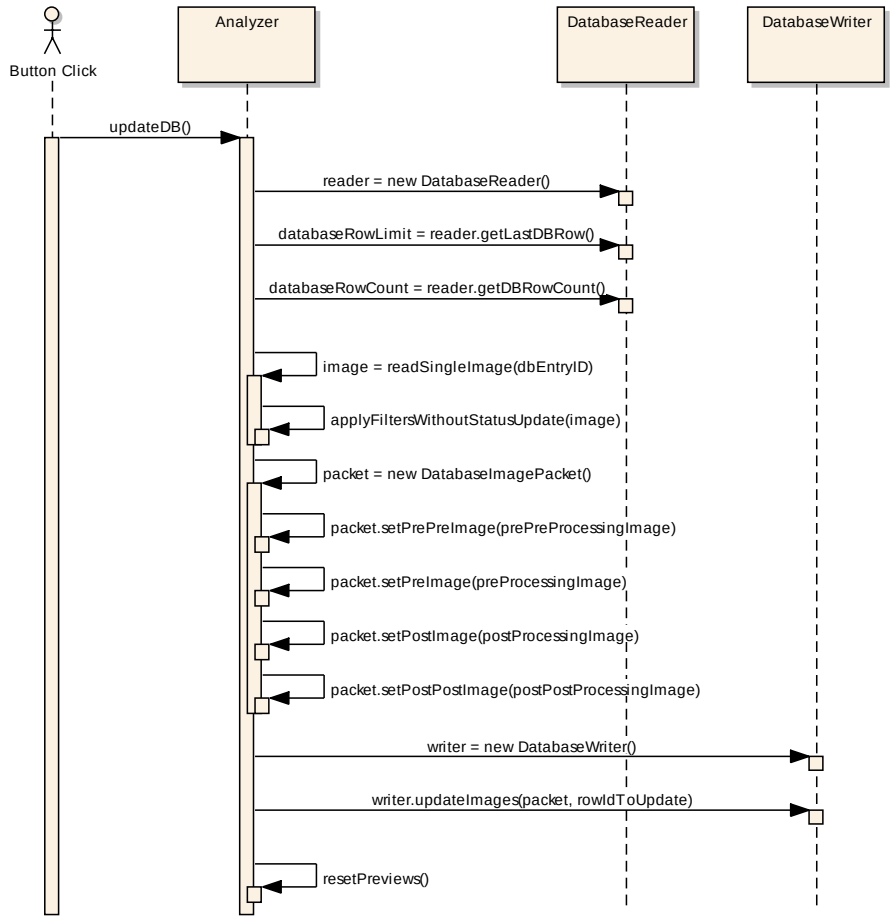



**Abbildung 15:**
**Programmfluss Update eines Datenbankentries**

Randnotiz: Der Programmfluss verwendet nicht mehrere Threads innerhalb des Analyzers. Es wurde im Diagramm lediglich der parallele Programmfluss gewählt, um die Zuständigkeiten der Methoden besser visualisieren zu können. Der Programmfluss lässt sich nun in folgende Bereiche unterteilen:

1) ***Filter auf das eingehende Bild anwenden***
Es werden der Reihe nach die Filter der Phasen 1, 2, 3 und 4 angewendet. Bei den Phasen wird jeweils das Endresultat zwischengespeichert.

2) ***Bilder packen und in Datenbank schreiben***

Die unter Schritt 1 erzeugten Zwischenbilder werden anschliessend in ein DatabaseImagePacket eingefügt. Diese Paket wird dann einem DatabaseWriter übergeben, welcher dann den Inhalt in die Datenbank schreibt.

***3) Analyzer reseten***

Zum Schluss wird der Analyzer geresetet, damit wieder ein reiner Tisch für Nachfolgeoperationen geschaffen wird.

# 3. Filterentwicklung und Testreihen

## 3.1. Einleitung

Die Filterentwicklung an sich ist bereits ein sehr komplexes und weitreichendes Gebiet. Es existieren wahrscheinlich tausende von Forschungsarbeiten weltweit zu diesem Thema, welche einen Lösungsansatz zu exakt einem Problem suchen, die dann erst noch oft von vielen Fachspezialisten betreut werden. Dass in diesem Projekt kein solches Ausmass an Entwicklungsarbeit investiert werden kann ist offensichtlich. Daher wurde meist ein pragmatischer Ansatz gewählt.
Meistens wurde durch Fachliteratur ein Konzept aufgegriffen ***[06] [07]***, dies nach Tauglichkeit bewertet und dann in einem ersten Prototyp implementiert. Falls etwas noch nicht so funktionierte wie es sollte, wurde die Logik entsprechend angepasst, bis sie zu ansprechenden Ergebnissen führte. Der Nachteil eines solchen Prozesses wiederspiegelt sich darin, dass in solchen Situationen immer ein Spagat zwischen generischer, schlechter Erkennung und sehr spezifischer, guter Erkennung gemeistert werden muss. Aus diesem Grund liefern Filter in gewissen Situationen massiv schlechtere Ergebnisse, wenn ein Input ausserhalb des „gedachten“ Andwendungsprofils verwendet wird. Dies äussert sich zum Beispiel beim Filter „FreeForm“-Filter sehr stark, welcher ein Objekt von einem Hintergrund freistellen soll.
Im Grossen und Ganzen lässt sich dennoch sagen, dass der trial & error Ansatz ausgehend von einem bewährten Grundkonzept relativ gute Ergebnisse innerhalb der vorhandenen Zeitperiode lieferte. So lassen sich Bilderansammlungen bis und mit etwa 50 Bildern ohne grössere Probleme verarbeiten, wobei es in einem solchen Umfang entsprechend höhere Mismatch-Raten gibt, als mit einer wesentlich kleineren Datenbank.

## 3.2. Fremde Filter *[08] [09] [10]*

### 3.2.1. Sharpen

Dieser Filter schärft das Bild. Er arbeitet mit einer vorgegebenen und in diesem Fall unveränderlichen Matrix zum Schärfen. Anhand der unten abgebildeten Matrix ist gut zu erkennen, wie der Schärfeprozess funktioniert: Es werden ausgehend von einem Pixel die Farbwerte rundherum analysiert und anschliessend vom Pixel im Zentrum abgezogen zu den Teilen, welche in der Matrix aufgeführt sind. Der Zentrumswert wird zuerst mit dem Faktor 1.8 multipliziert, damit er die gleiche Intensität nach der Operation aufweist, da durch die umgebenden Pixel ein Faktor von insgesamt 0.8 wieder abgezogen wird. Andernfalls wäre das resultierende Pixel viel zu dunkel.

$$m_{Sharpen} = \begin{bmatrix} 0.0f & -0.2f & 0.0f \\ -0.2f & 1.8f & -0.2f \\ 0.0f & -0.2f & 0.0f \end{bmatrix}$$

Konkret bedeutet dies für die gegebene Matrix, dass sich der Wert des Zentrumspixels wie folgt ermitteln lässt, wobei dieser Prozess bei einem Farbbild für jeden Farbkanal (rot | grün | blau) separat wiederholt werden müsste:

$$RGB_{center} = 1.8 * RGB_{Center} - 0.2 * RGB_{upper} - 0.2 * RGB_{lower} - 0.2 * RGB_{left} - 0.2 * RGB_{right}$$

### 3.2.2. Level

Dieser Filter verstärkt einen definierten Farbbereich innerhalb eines Bildes. Dies wird realisiert, in dem zuerst die Histogrammkurve des Bildes ermittelt wird, anhand derer man den unteren und den oberen Schwellwert definiert. Es können bei dieser Implementation keine einzelnen Farbkanäle verändert werden, sondern nur der gesamte RGB-Farbraum. Anschliessend definiert man die SOLL-Intensität des zuvor definierten Schwellwertbandes. Nun werden die Farbwerte innerhalb dieses Schwellwertbandes um die entsprechenden Werte erhöht, bzw. erniedrigt. Falls die Werte zu stark erniedrigt werden entsteht als Resultat ein invertiertes Bild. In der folgenden Abbildung aus dem freien Grafikprogramm Gimp[3] wird dargestellt, wie dies konkret funktioniert. Die Abbildung entstammt nicht aus dem eigenen Projekt, weil die Graphen im Hintergrund generiert werden und daher unsichtbar sind.
Es ist gut erkennbar, dass man zuerst den Bereich der verwendeten Quellwerte definieren muss. Dieser Bereich bestimmt - wie schon zuvor erwähnt - die zu verwendenden Farben

[3] Gimp, GNU Image Manipulation Program, freie Photoshop-Alternative, welche leider nicht so professionell wie das Vorbild verwendet werden kann, da es immer wieder Abstürze und weitere unannehmlichen Vorfälle erleidet. Download unter http://www.gimp.org/

für die Operation. Anschliessend definiert man einen Zielwertbereich. Die Farben, welche sich innerhalb des Quellwertbereichs aufhalten, werden nun verhältnismässig korrekt auf den Zielwertbereich abgebildet. Dies lässt sich einfach am Resultat erkennen: Ein kleines Farbspektrum des Bildes (ganz dunkle Töne zwischen 25 - 32 RGB) werden auf 127 - 255 RGB verschoben / transformiert. Im Bild ist nun gut erkennbar, dass die besonders dunklen Töne noch vorhanden sind und diese relativ hell, bzw. gräulich dargestellt werden. Dies ist so, weil der untere Schwellwert des Zielwertbereichs bei 127 liegt. Dies ist exakt die Mitte des RGB Farbspektrums, also 50% schwarz (= grau).

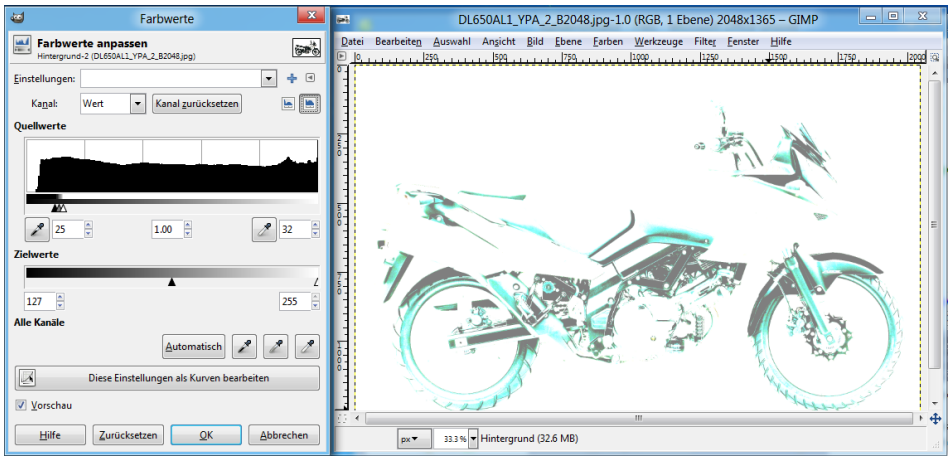



**Abbildung 16:**
**Schwellwertbeispiel aus dem Programm Gimp**

### 3.2.3. DifferenceOfGaussians

Dieser Filter hebt Kanten hervor indem er zuerst zwei Bilder mit unterschiedlichen Gauss-Unschärfen (ausgehend vom Originalbild) generiert und vergleicht. Der Radius dieser zwei Unschärfen kann beliebig definiert werden. Nun werden die Unterschiede dieser zwei Gaussbilder verglichen, je grösser die Unterschiede, desto stärker die Kante. Die Wirkungsweise ist relativ einfach erklärt: Ein Unschäfefilter holt Pixel aus der Umgebung ins Zentrum, je stärker eben dieser Filter ist, desto weiter entfernte „Umgebungen“ werden ins Zentrum geholt. Nun hat man also zwei unscharfe Bilder, welche unterschiedlich grosse Bereiche ins Zentrum holen. Die Differenzen in den unscharfen Bildern zeigen nun die Unterschiede der Lokalitäten der Farben auf, d. h. je grösser der Farbunterschied der Umgebung ist, desto stärker wird der betreffende Zentrumspixel markiert. Schlussendlich werden also die Farbübergänge betrachtet; je stärker ein Farbübergang, desto stärker die hypothetisch erkannte Kante. Dieses Prinzip wird bei fast allen Kantenerkennungsalgorithmen verwendet. Teilweise etwas ausgefeilter als in diesem Exemplar. Beispielsweise liessen sich durch die erkannten Kanten in der Umgebung weitere Rückschlüsse auf die so ermittelten Kanten ziehen, die das Ergebnis weiter verbessern könnten. Weiter steht die Frage im Raum, ob es wirklich notwendig ist, den Umweg über die Unschärfe zu gehen, damit man ansprechende Resultate bekommt. Dies ist definitiv nicht zwingend notwendig, wie die eigene Implementation des Filters ColorDifferenceWithBacklog (Kapitel 2.3.3) beweist.

### 3.2.4. GaussianBlur

Dieser Filter generiert eine Unschärfe im Bild. Dies wird erreicht, indem der Vorgang des Schärfens umgekehrt wird, d. h. anstelle vom Subtrahieren der umgebenden Bildwerte, werden die Bildwerte addiert. Somit kommt ein Teil der Farbinformation zum Zentrum und führt zu einer leichten Blur-Bildung. Im Gegensatz zum Schärfefilter kann dieser Filter mit einer frei definierbaren Intensität verwendet werden. Die Blur-Matrix wird aufgebaut durch die Anwendung der Gaussfunktion auf die zuvor definierten Filterparameter. Der Gaussalgorithmuss bestimmt somit die Wertigkeit eines spezifischen Punktes P, der um ein Zentrum Z liegt. Die Wertigkeit wird bestimmt durch die Gaussformel ***[11] [12] [13]***

$$f(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

wobei

$$\sigma = Standardabweichung\ (ein\ float - Value)$$

$$e \approx 2.7182818\ (Euler - Zahl)$$
$$x = X - Koordinate, relativ\ zum\ Zentrumspixel$$
$$y = Y - Koordinate, relativ\ zum\ Zentrumspixel$$

Wenn man nun diesen Algorithmus für jeden umgebenden Pixel (z. B. innerhalb einer 4x4 Matrix) anwendet, erhält man nun eine Matrix, in welcher die Wertigkeiten nach Gauss verteilt sind.
Die ominöse Standardabweichung definiert nun die Steilheit der Gausskurve (siehe Bild unten). Je kleiner die Abweichung, desto spitziger wird die Kurve und dementsprechend grösser fallen die Wertigkeitsunterschiede von Zeile zu Zeile, bzw. Spalte zu Spalte aus. Die Gaussfunktion verwendet in der Regel maximal eine 7x7 Matrix, da andernfalls die Werte, welche noch weiter vom Zentrum entfernt sind (gegeben $\sigma = \sim 0.84$), eine so kleine Rolle spielen, dass sie auch gleich komplett vernachlässigt werden können.
In den folgenden mit Maple geplotteten Graphen (auf der nächsten Seite) kann man die Gaussfunktionen mit einigen unterschiedlichen Standardabweichungen erkennen. Man kann sich die dazugehörigen Matrizen relativ einfach pragmatisch herleiten: Von oben auf den Graphen sehen, das Zentrum der Matrix wird durch den höchsten Punkt im Graphen repräsentiert. Nun können einfach die Z-Achsen Werte auf den jeweiligen Ort in der Matrix abgebildet werden.
Für eine etwas bessere Visualisierung ist in der folgenden Grafik mit Standardabweichung $\sigma = 0.8$ die Ausprägung der Höhe unterschiedlich eingefärbt. Es ist somit klar erkennbar, dass der grösste Informationsgehalt vom Zentrum geholt wird und gegen aussen immer weiter abgeschwächt wird. Beispielsweise könnte nun für die Matrix an der Stelle ***(-1, -1)*** konkret der Z-Wert des Graphen an der ***bezeichneten*** Stelle ausgelesen und eingesetzt werden ***[14]***.

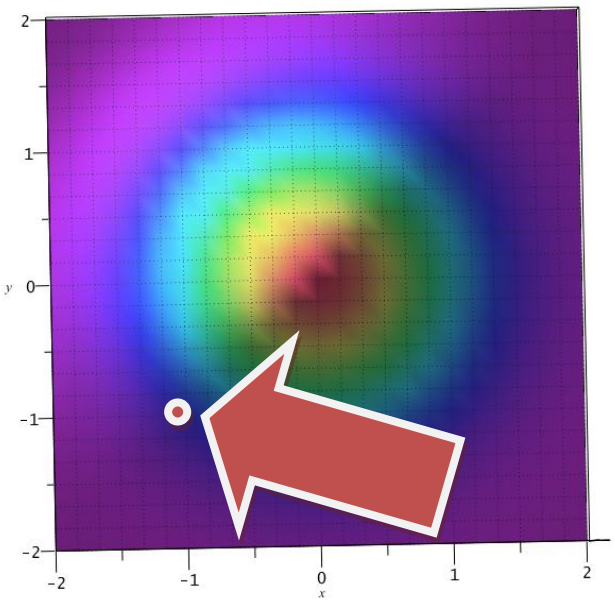
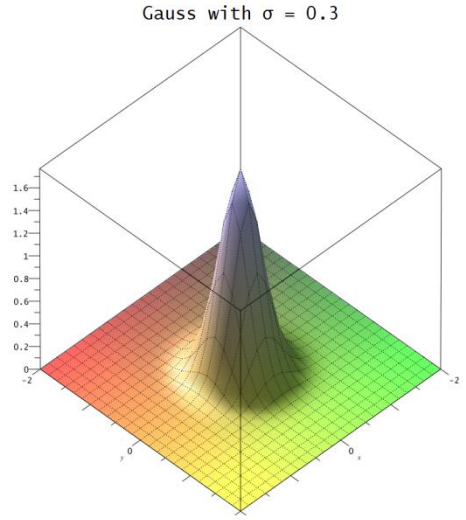
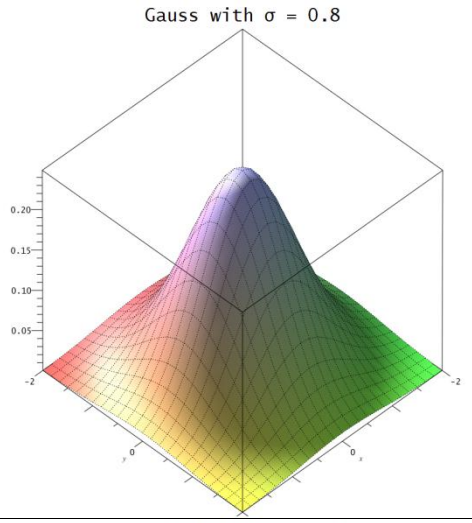



**Abbildung 17:**
**Gaussalgorithmus geplottet mit einer Standardabweichung von 0.8 (Sicht von „oben“)**



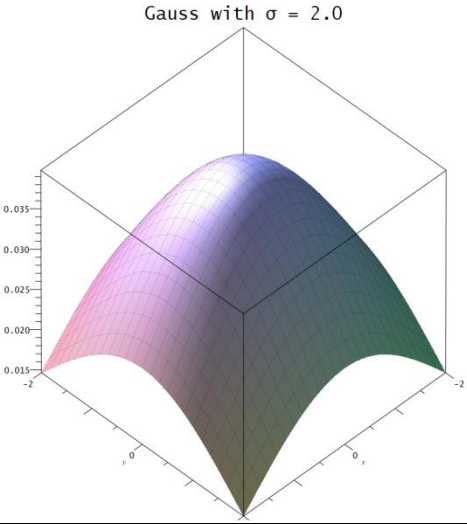

$$M := \begin{bmatrix} .00000000 & .00000000 & .00000000 & .00000000 & .00000000 & .00000000 & .00000000 \\ .00000000 & .00000000 & .00000000 & .00000000 & .00000000 & .00000000 & .00000000 \\ .00000000 & .00000000 & .00002643 & .00683645 & .00002643 & .00000000 & .00000000 \\ .00000000 & .00000000 & .00683645 & 1.76838826 & .00683645 & .00000000 & .00000000 \\ .00000000 & .00000000 & .00002643 & .00683645 & .00002643 & .00000000 & .00000000 \\ .00000000 & .00000000 & .00000000 & .00000000 & .00000000 & .00000000 & .00000000 \\ .00000000 & .00000000 & .00000000 & .00000000 & .00000000 & .00000000 & .00000000 \end{bmatrix}$$

Es ensteht eine sehr steile und schmale Glockenform. Es werden daher nur die Werte im Zentrum berücksichtigt, die Werte weiter entfernt davon werden ignoriert.
In diesem Fall wäre eine 3x3 Matrix sinnvoll.
Es entsteht eine schwache und konturstarke Unschärfe.




$$M := \begin{bmatrix} .00000019 & .00000966 & .00010063 & .00021979 & .00010063 & .00000966 & .00000019 \\ .00000966 & .00048006 & .00500239 & .01092622 & .00500239 & .00048006 & .00000966 \\ .00010063 & .00500239 & .05212608 & .11385382 & .05212608 & .00500239 & .00010063 \\ .00021979 & .01092622 & .11385382 & .24867960 & .11385382 & .01092622 & .00021979 \\ .00010063 & .00500239 & .05212608 & .11385382 & .05212608 & .00500239 & .00010063 \\ .00000966 & .00048006 & .00500239 & .01092622 & .00500239 & .00048006 & .00000966 \\ .00000019 & .00000966 & .00010063 & .00021979 & .00010063 & .00000966 & .00000019 \end{bmatrix}$$

Es ensteht eine steile und etwas wulstige Glockenform. Es werden daher die Werte im Zentrum und die Werte etwas weiter entfernt davon berücksichtigt. Nur Werte ganz am Rand sind wieder relativ klein.
In diesem Fall wäre eine 7x7 Matrix sinnvoll.
Es entsteht eine durchschnittliche Unschärfe.




$$M := \begin{bmatrix} .00444154 & .00490867 & .00521220 & .00531749 & .00521220 & .00490867 & .00444154 \\ .00490867 & .00542492 & .00576037 & .00587674 & .00576037 & .00542492 & .00490867 \\ .00521220 & .00576037 & .00611657 & .00624014 & .00611657 & .00576037 & .00521220 \\ .00531749 & .00587674 & .00624014 & .00636620 & .00624014 & .00587674 & .00531749 \\ .00521220 & .00576037 & .00611657 & .00624014 & .00611657 & .00576037 & .00521220 \\ .00490867 & .00542492 & .00576037 & .00587674 & .00576037 & .00542492 & .00490867 \\ .00444154 & .00490867 & .00521220 & .00531749 & .00521220 & .00490867 & .00444154 \end{bmatrix}$$

Es ensteht eine abgeflachte und stark wulstige Glockenform. Es werden daher praktisch alle Werte auch im grösseren Umkreis berücksichtigt.
In diesem Fall wäre eine Matrix sinnvoll, welche mehr als 7x7 Elemente umfassen würde.
Es entsteht eine aussergewöhnlich starke Unschärfe.

**Tabelle 13:**
**Gaussalgorithmus geplottet mit unterschiedlichen Standardabweichungen inkl. der zugehörigen Matrix**

## 3.3. Eigene Filter

### 3.3.1. Stretch

Dieser Filter skaliert das Bild (Vergrösserung oder Verkleinerung) auf eine definierte Auflösung. Er verwendet standardmässig die SCALE_SMOOTH Option, in welcher das Bild weichgewaschen wird und damit eine Verpixelung verhindert wird. Diese Option liefert insgesamt bessere Resultate, verbraucht aber um Faktoren mehr CPU-Leistung als BufferedImage.SCALE_FAST. Dies wäre daher eine Option für die spätere Optimierung der Software.

### 3.3.2. Crop

Dieser Filter schneidet das Bild zu. Als Parameter werden zwei Punkte verwendet, welche die Startkoordinaten und die Zielkoordinaten darstellen.


Abbildung 18:
Parametererklärung des Crop-Filters

### 3.3.3. ColorDifferenceWithBacklog

Dieser Filter markiert Kanten und verwendet zur Analyse derer die Verteilung der Farben innerhalb des Bildes. Einfach erklärt verwendet er ein Schiebefenster, welches immer den aktuellen Farbwert eines Pixels mit dem des Pixels zwei Positionen vorher vergleicht. Es lässt sich ein Schwellwert definieren, ab welcher Differenz des RGB-Mittelwerts eine Kante erkannt werden soll. Um das Resultat weiter zu verbessern wird ein Durchgang von oben nach unten, sowie ein zweiter Durchgang von unten nach oben gestartet. Die Resultate dieser beiden Durchgänge werden dann kombiniert und führen dann zum ausgegebenen Bild. Es wäre denkbar nicht nur spaltenweise, sondern auch zeilenweise (links nach rechts und umgekehrt) einen Durchgang zu starten. Dies wurde nicht realisiert, weil die Resultate so schon sehr ansprechend waren und die weiteren Durchgänge zu einer übermässigen Erkennung von Kanten geführt hätten. Als Experiment wäre es dennoch sinnvoll und interessant dieses Verhalten zu untersuchen.
In der folgenden Grafik ist der Ablauf schematisch dargestellt (Variante von oben nach unten). Jedes Rechteck mit einem RGB-Value stellt einen einzelnen Pixel dar. Der Mittelwert bildet sich aus den drei Farbwerten rot, grün und blau. Die Differenz bezieht sich ausschliesslich auf den Mittelwert. Falls die Differenz mehr als 25 beträgt, wird das betreffende Pixel schwarz markiert.

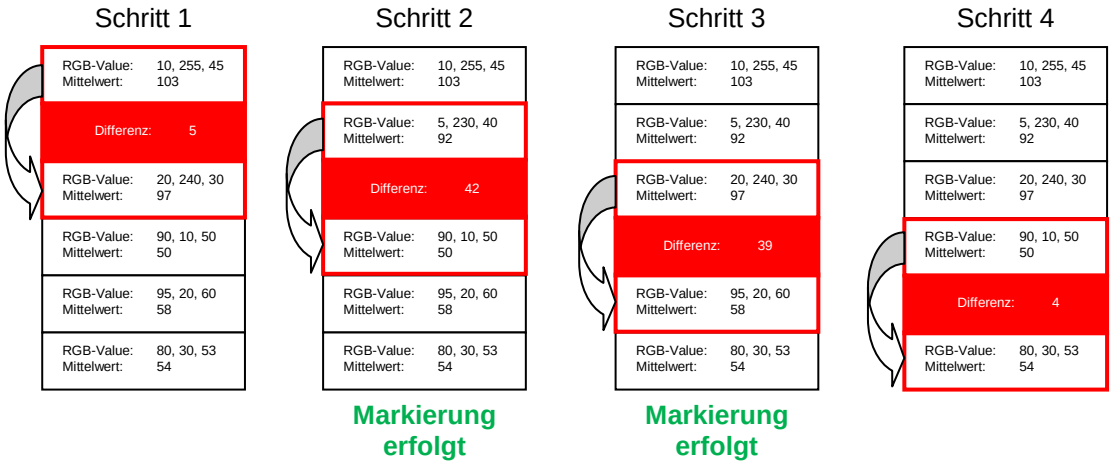



Abbildung 19:
Ablaufgrafik des ColorDifferenceWithBacklog Filters

### 3.3.4. FreeFormSelect

Dieser Filter stellt ein Bild frei vom Hintergrund. Dies wird erreicht durch den Einsatz einer Kantenhervorhebung (mit Filter von Kapitel 2.3.3), dem Entfernen von Rauschen und abschliessend einem Freistellprozess. Im resultierenden Bild nach dem Einsatz des ColorDifferenceWithBacklog-Filters werden zuerst Punkte zusammengefasst betrachtet und dann je nach Intensität entfernt, oder eben nicht. Dieser Vorgang wird zweimal ausgeführt; einmal mit einem kleinen Raster und einmal mit einem grösseren Raster. Anschliessend werden freistehende Punkte gelöscht. Dieser Prozess ist in den folgenden Bildern dargestellt.
Eine Besonderheit weist dieser Filter noch auf: Er kann die Qualität der eingehenden Bilder beurteilen. Falls sich die ermittelten Freistellkonturen oft überschneiden setzt er ein Flag, welches auf ein schlechtes Bild deutet. Der Server hat nun die Möglichkeit, die weitere Verarbeitung des Bildes abzubrechen und den Sender des Bildes umfassend zu informieren. Dies ist übrigens im Server bereits bis auf Protokollebene hinunter implementiert und betriebsbereit. Falls also ein schlechtes Bild zur Verarbeitung geschickt wird, erscheint beim Client eine MessageBox mit dem Hinweis, das Bild entspreche nicht den Qualitätsanforderungen, er solle dieses Bild bitte noch einmal erstellen und senden. Der Grund für diese Lösung liegt in der sehr stark vom Inhalt abhängigen Freistellfunktion, die auf diese Weise konstantere Resultate liefern und schlechte zumindest grob filtern kann.

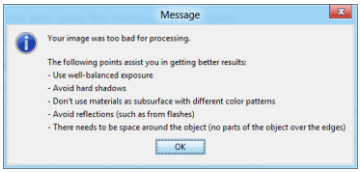
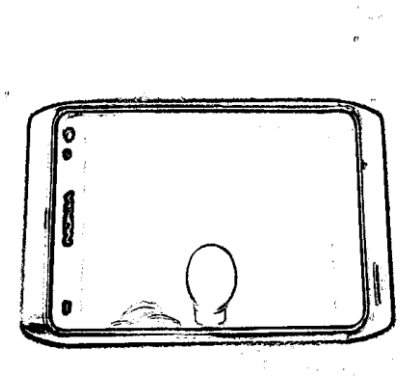
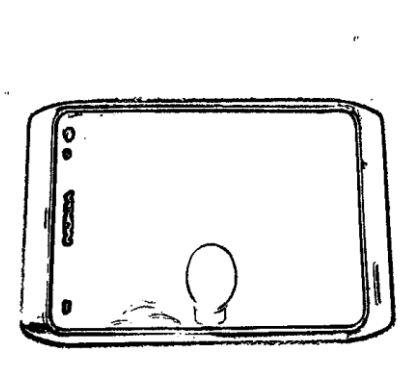
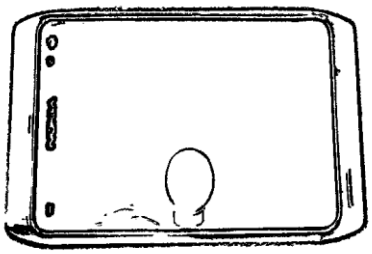
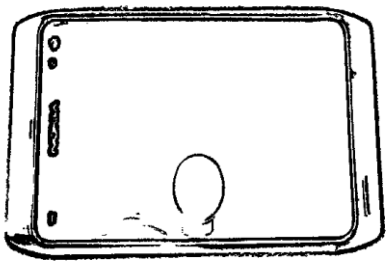


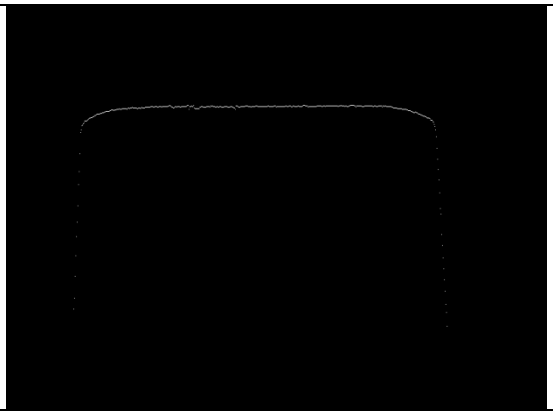
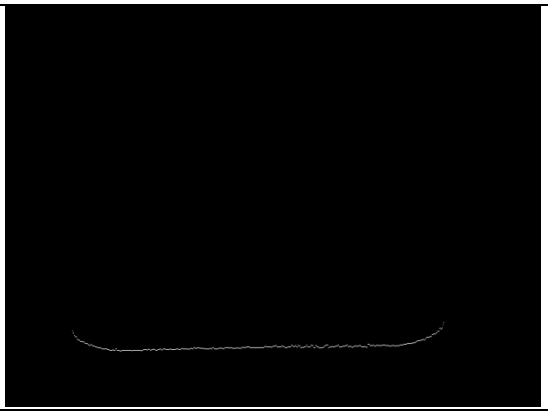

**Abbildung 20:**
**Fehlermeldung bei einem schlechten Bild**

| | |
|---|---|
|  | 1) Dieses Bild ist das Ausgangsprodukt. Es wurde mit einer handelsüblichen Webcam aufgenommen. Es befindet sich in der Mitte ein Objekt (Mobiltelefon) auf einer grauen und leicht verschmutzten Unterlage. Ziel soll nun sein, dieses Objekt möglichst korrekt frei zu stellen vom Hintergrund, damit nur die Farben des Mobiltelefons für die mögliche Weiterverarbeitung berücksichtigt werden. |

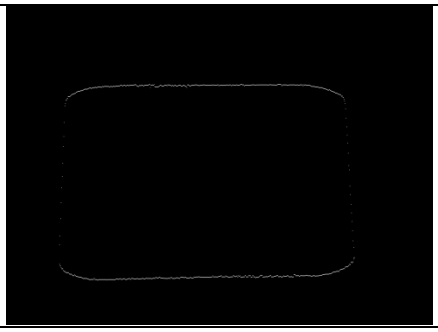
| | |
|---|---|
|  |  |
| 2) Dies ist das resultierende Bild nach Anwendung des ColorDifferenceWithBacklog-Filters auf das originale Bild.<br>Es ist gut zu erkennen, dass um das Mobiltelefon herum ein sehr schwaches Rauschen besteht. Ausserdem werden gewisse Schmutzstellen als schwarze Punkte abgebildet. Dies ist in der oberen rechten Ecke gut zu erkennen. | 3) Nun wurde mit Hilfe einer 3x3 Matrix auf den Schwarzwert geachtet. Falls dieser über dem Schwellwert liegt, werden die Pixel belassen, falls er darunter liegt, werden die Pixel mit der Farbe weiss überschrieben.<br>Es ist schnell sichtbar, dass das Rauschen entfernt wurde, jedoch einige kleinere Schmutzflecke immer noch vorhanden sind. |

| | |
|---|---|
|  |  |
| 4) Nun wurde mit Hilfe einer 5x5 Matrix auf den Schwarzwert geachtet. Falls dieser über dem Schwellwert liegt, werden die Pixel belassen, falls er darunter liegt, werden die Pixel mit der Farbe weiss überschrieben.<br>In diesem Bild sind bereits praktisch alle Flecken entfernt. Nur besonders starke Schwarzansammlungen wären noch weiter existent. | 5) Nun wird kontrolliert, ob der Bereich innerhalb einer 15x15 Matrix ohne Verbindung zu weiteren Kanten steht. Dazu wird der „Rand“ der Matrix analysiert. Falls sich viele Schwarzvalues am Rand befinden wird der Inhalt der Matrix belassen, andernfalls wird er mit weiss überschrieben.<br>Diese Methode löscht noch einige meist fehlerhafte Stellen, die mit den vorherigen Methoden noch nicht behandelt werden konnten.<br>Im Beispielbild ist eine kleine Änderung erkennbar in der Spiegelung auf dem Display des Mobiltelefons: Ein Teil einer freistehenden Kurve wurde weggelöscht, da er keine Verbindung zu anderen Kanten aufgewiesen hatte. |

Es ist vielleicht die Frage aufgetaucht, warum denn ein so hoher Aufwand betrieben wird, um die Fehler zu entfernen. Die Antwort ist ganz einfach: Je weniger Fehler im Bild vorhanden sind, desto besser wird der nun beschriebene Freistellprozess funktionieren. Für diesen Freistellprozess wird anschliessend von oben und unten die Kante bestimmt und so die Grundlage für das Freistellen des Bildes geschaffen. Dieser Prozess ist im Folgenden etwas genauer erläutert.

| | |
|---|---|
|  |  |
| 6) Dies ist die obere Kante. Sie wird ermittelt, in dem vom oberen Rand zum Zentrum gewandert wird und anschliessend beim Auftreffen einer schwarzen Markierung aus dem Kantenbild ein weisses Pixel gesetzt wird. Also enthält jede Spalte schlussendlich maximal ein weisses Pixel. | 7) Dies ist die untere Kante. Sie wird ermittelt, in dem vom unteren Rand zum Zentrum gewandert wird und anschliessend beim Auftreffen einer schwarzen Markierung aus dem Kantenbild ein weisses Pixel gesetzt wird. Also enthält jede Spalte schlussendlich maximal ein weisses Pixel. |

Diese zwei Konturen werden nun übereinander gelegt. Anschliessend wird anhand der Kontur das Bild freigestellt. Freistellen bedeutet, dass Alphakanal-Transparenzwerte in den zu entfernenden Bereichen gesetzt werden ***[15]***.

| | |
|---|---|
|  |  |
| Dies ist nun das Resultat, falls man beide Konturen übereinander legt. | Anhand der zuvor generierten zwei Konturen kann nun das Bild freigestellt werden. Wie man sieht, funktioniert dies relativ zuverlässig, solange keine U-Formen um 90° gedreht verwendet oder der Hintergrund zu stark gemustert ist. |

**Tabellen 14 - 18:**
**Erklärung der einzelnen Schritte des FreeForm-Filters**

### 3.3.5. Grayscale

Dieser Filter konvertiert das Bild in einen Graustufenfarbraum. Nebenbei: Graustufen haben die Eigenheit, dass jeweils immer die rot/grün/blau-Komponenten eines RGB-Wertes die gleiche Wertigkeit aufweisen. Also wäre beispielsweise RGB(42, 42, 42) ein Grauton, RGB(42, 45, 43) jedoch nicht. Die Implementation verwendet die folgenden built-in Java Funktionen um den Farbraum zu konvertieren:

```
// Erstellt ein ColorSpace mit einem Graustufenfarbraum.
ColorSpace cs = ColorSpace.getInstance(ColorSpace.CS_GRAY);

// Erstellt den Worker, welcher mit einem ColorSpace arbeiten kann.
 ColorConvertOp op = new ColorConvertOp(cs, null);

// Wendet den vorkonfigurierten Worker auf ein BufferedImage an.
 __imgPostFilter = op.filter(this.__imgPreFilter, null);
```

### 3.3.6. BlackAndWhiteOnly

Dieser Filter konvertiert das Bild in eine schwarz/weiss-Repräsentation. Dies wird durch den Vergleich eines Pixels mit einem zu definierenden Schwellwert erreicht. Liegt der Farbwert unter dem Schwellwert, wird das Pixel schwarz, andernfalls weiss. Der Vergleich arbeitet auch hier wieder mit dem Mittelwert über die einzelnen Komponenten eines RGB-Values.

### 3.3.7. BlackAreaRemover

Dieser Filter entfernt schwarze Flächen. Er war ursprünglich gedacht um Kanten zu detektieren, dies funktionierte jedoch nur eingeschränkt, da er aufgrund der Verwendung eines Rasters und der damit verbundenen grossen Granularität sehr ungenau war. Somit verfälschte er immer feine Strukturen in einem Bild. Durch das Raster führt er zu einer rasterigen Optik, welche an das Bild alter Computermonitore erinnert.

### 3.3.8. Dot

Dieser Filter soll auch wieder störende schwarze Punkte in einem Bild entfernen. Dies geschieht in dem er von Pixel zu Pixel wandert und die Schwarzpixel um den betreffenden Pixel herum zählt. Der Umfang dieses Zählvorgangs wird durch einen Parameter gesteuert, welcher die Matrixgrösse definiert, innerhalb der nach schwarzen Punkten gesucht wird. Beispielsweise wird bei einer `_matrixSize` von 5 eine 5x5 Matrix verwendet und mit dem zugehörigen `_blackThreshold` Parameter wird dann noch der Schwellwert definiert, ab wievielen schwarzen Pixeln der bereich „bereinigt“ werden soll.

### 3.3.9. InvertOnDemand

Dieser Filter zählt die schwarzen Punkte eines Bildes und wenn sie über einem zu definierenden Schwellwert liegen, dann werden die schwarzen Punkte weiss und umgekehrt. Der Einsatz dieses Filters ist äusserst beschränkt. Er wurde in den Anfangszeiten diese Projekts eingesetzt,

falls gewisse Ergebnisse nicht sehr aussagekräftig waren, um diese dann etwas anschaulicher und besser interpretierbar zu gestalten.

#### 3.3.10. MoveBWImage

Dieser Filter verschiebt die schwärzeste Zeile und Spalte in den Mittelpunkt. Der Grund für diesen Filter war, dass der ursprünglich entwickelte Filter SquareSplitCount (Kapitel 2.4.1) keine sinnvollen Resultate lieferte, falls sich zwei identische Objekte nicht am selben Ort befanden. Dieser Filter funktioniert nur mit praktisch identischen Bildern sinnvoll, da andernfalls bereits kleine Schwankungen in den Schwarzwerten massive Verschiebungen zur Folge haben und so das Resultat komplett zunichte machen können.

#### 3.3.11. ColorHistogramGraph

Dieser Filter fällt in die Kategorie der visualisierenden Filter. Dieser Filter zählt die Vorkommnisse der verschiedenen Farben im Bild und stellt sie in einem Diagramm grafisch dar.

#### 3.3.12. HorizontalBlackGraph

Dieser Filter fällt in die Kategorie der visualisierenden Filter. Dieser Filter zählt die schwarzen Pixel innerhalb einer Zeile im Bild und stellt sie für jede Zeile in einem Diagramm grafisch dar.

#### 3.3.13. VerticalBlackGraph

Dieser Filter fällt in die Kategorie der visualisierenden Filter. Dieser Filter zählt die schwarzen Pixel innerhalb einer Spalte im Bild und stellt sie für jede Spalte in einem Diagramm grafisch dar.

## 3.4. Eigene Comparation Modules

### 3.4.1. SquareSplitCount

Dieses Comparation Module rastert das Bild und vergleicht die Vorkommnisse von schwarzen Pixeln innerhalb eines Teilbereichs. Je mehr die Anzahl von Pixeln mit denen in der Quelle übereinstimmt, desto grösser die Übereinstimmung. Da dieses Modul sehr empfindlich auf Veränderungen aller Art in den Bildern reagiert, wird es im späteren Verlauf nicht mehr speziell erwähnt, denn es funktioniert nur in ganz wenigen Situationen, in denen die zu vergleichenden Bilder praktisch identisch sind. Es könnte allenfalls sinnvoll bei einer Produktkontrolle verwendet werden, bei der jedes Produkt exakt einem Muster gleichen muss (z. B. Überwachung der Fertigung von Leiterplatinen oder Kunststoffteilen).

### 3.4.2. ColorHistogram

Dieses Comparation Module zählt die Vorkommnisse der einzelnen Farben im Bild und vergleicht die ermittelten Häufigkeiten mit denen des Originals. Je kleiner die Diskrepanz, desto besser der Match.

### 3.4.3. Size2D

Dieses Comparation Module ermittelt die Pixelfläche, welche das freigestellte Objekt umfasst und vergleicht anschliessend diese Fläche mit dem Original. Eine Voraussetzung für dieses Comparation Module ist die vorherige Anwendung des FreeFormFilters (Kapitel 2.3.4).

## 3.5. Konkrete Applizierung der Filter

Falls die jeweiligen Filter Konfigurationsparameter aufweisen, werden sie mit verschiedenen Konfigurationen demonstriert. Andernfalls wird nur der default-Filterprozess aufgezeigt.

### 3.5.1. Filter - Sharpen

Hinweis: Der Filter wurde acht Mal hintereinander angewandt, da das Resultat sonst in dieser kleinen Vorschau praktisch nicht sichtbar gewesen wäre.

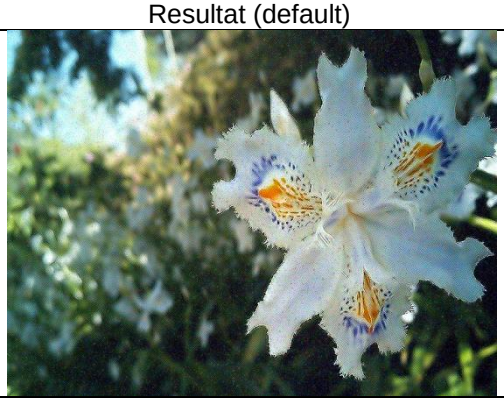
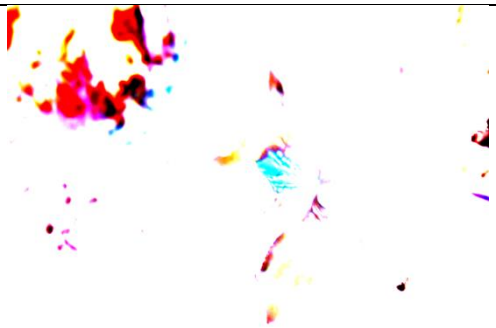
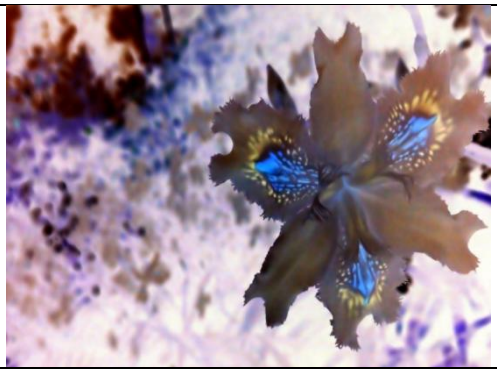
| Ausgangsbild | Resultat (default) |
|---|---|
|  |  |

Tabelle 19:
Anwendung des Sharpen-Filters auf ein Bild

### 3.5.2. Filter - Level

Hinweis: Wie in den Bildern unten zu sehen ist, würde sich dieser Filter auch als Farbinverter verwenden lassen.

| Konfiguration Resultat 1 (default) | Konfiguration Resultat 2 | Konfiguration Resultat 3 |
|---|---|---|
| `_lowLevel = 0.95f`<br>`_highLevel = 0.9f`<br>`_lowOutputLevel = 0.05f`<br>`_highOutputLevel = 0.6f` | `_lowLevel = 0.05f`<br>`_highLevel = 0.6f`<br>`_lowOutputLevel = 0.9f`<br>`_highOutputLevel = 0.95f` | `_lowLevel = 0.0f`<br>`_highLevel = 1.0f`<br>`_lowOutputLevel = 1.0f`<br>`_highOutputLevel = 0.0f` |

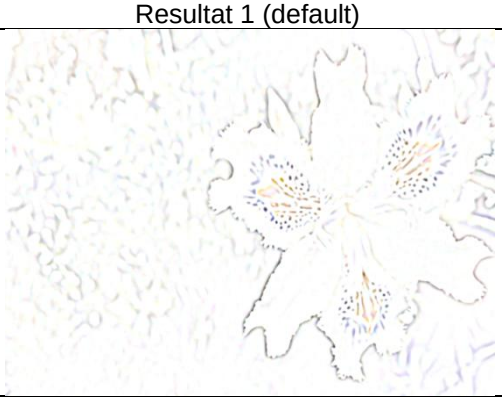
| Ausgangsbild | Resultat 1 (default) |
|---|---|
|  |  |
| Resultat 2 | Resultat 3 |
|  |  |

Tabellen 20, 21:
Anwendung des Level-Filters auf ein Bild

### 3.5.3. Filter - DifferenceOfGaussians

| Konfiguration Resultat 1 (default) | Konfiguration Resultat 2 | Konfiguration Resultat 3 |
|---|---|---|
| `_radius1 = 1.0f` | `_radius1 = 1.0f` | `_radius1 = 3.0f` |

| | | |
|---|---|---|
| `_radius2 = 2.0f`<br>`_normalize = true`<br>`_invert = true` | `_radius2 = 2.0f`<br>`_normalize = false`<br>`_invert = true` | `_radius2 = 10.0f`<br>`_normalize = true`<br>`_invert = false` |

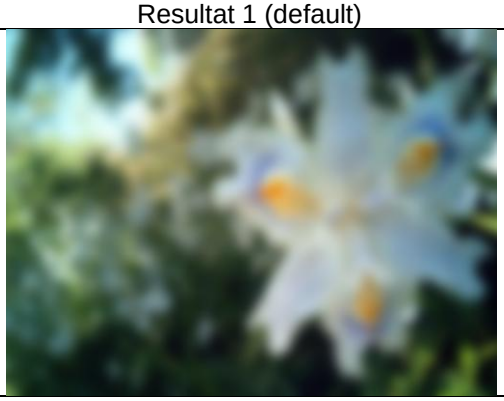
| Ausgangsbild | Resultat 1 (default) |
|---|---|
|  |  |

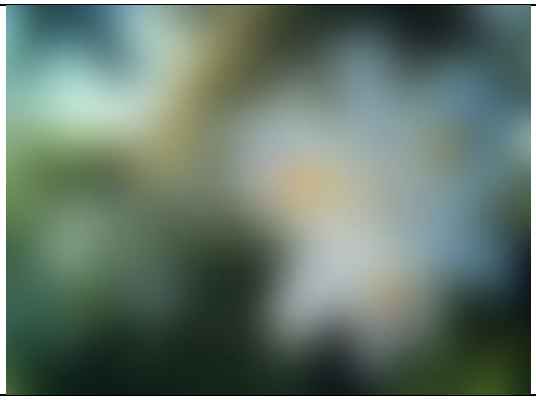
| Resultat 2 | Resultat 3 |
|---|---|
|  |  |

**Tabellen 22, 23:**
**Anwendung des DifferenceOfGaussians-Filters auf ein Bild**

### 3.5.4. Filter - GaussianBlur

| Konfiguration Resultat 1 (default) | Konfiguration Resultat 2 | Konfiguration Resultat 3 |
|---|---|---|
| `_radius = 30.0f` | `_radius = 5.0f` | `_radius = 100.0f` |

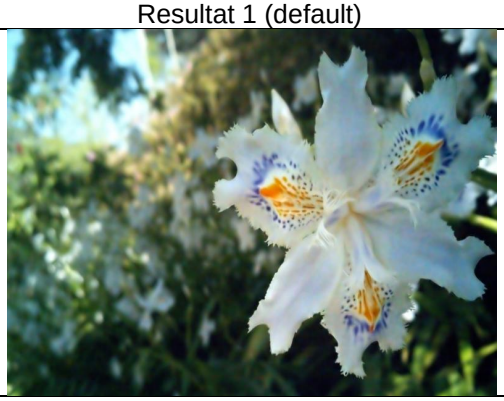
| Ausgangsbild | Resultat 1 (default) |
|---|---|
|  |  |

| Resultat 2 | Resultat 3 |
|---|---|
|  |  |

**Tabellen 24, 25:**
**Anwendung des GaussianBlur-Filters auf ein Bild**

### 3.5.5. Filter - Stretch

| Konfiguration Resultat 1 (default) | Konfiguration Resultat 2 | Konfiguration Resultat 3 |
|---|---|---|
| `_xSize = 800`<br>`_ySize = 600` | `_xSize = 320`<br>`_ySize = 240` | `_xSize = 100`<br>`_ySize = 100` |

| Ausgangsbild | Resultat 1 (default) |
|---|---|
|  |  |
| **Resultat 2** | **Resultat 3** |
|  |  |

**Tabellen 26, 27:**
**Anwendung des Stretch-Filters auf ein Bild**

### 3.5.6. Filter - Crop

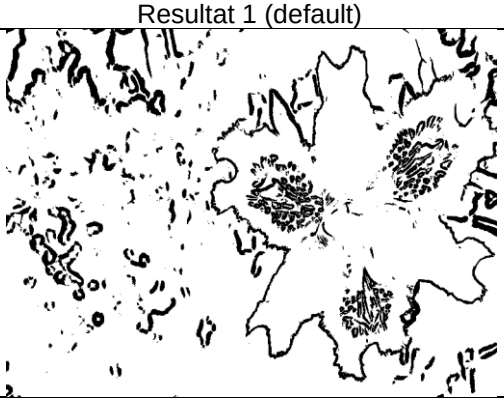
| Konfiguration Resultat 1 (default) | Konfiguration Resultat 2 | Konfiguration Resultat 3 |
|---|---|---|
| `_x1 = 0`<br>`_y1 = 0`<br>`_x2 = 200`<br>`_y2 = 200` | `_x1 = 250`<br>`_y1 = 250`<br>`_x2 = 300`<br>`_y2 = 300` | `_x1 = 100`<br>`_y1 = 100`<br>`_x2 = 400`<br>`_y2 = 150` |

| Ausgangsbild | Resultat 1 (default) |
|---|---|
|  |  |
| **Resultat 2** | **Resultat 3** |
|  |  |

**Tabellen 28, 29:**
**Anwendung des Crop-Filters auf ein Bild**

### 3.5.7. Filter - ColorDifferenceWithBacklog

| Konfiguration Resultat 1 (default) | Konfiguration Resultat 2 | Konfiguration Resultat 3 |
|---|---|---|
| `_threshold = 15` | `_threshold = 25` | `_threshold = 50` |

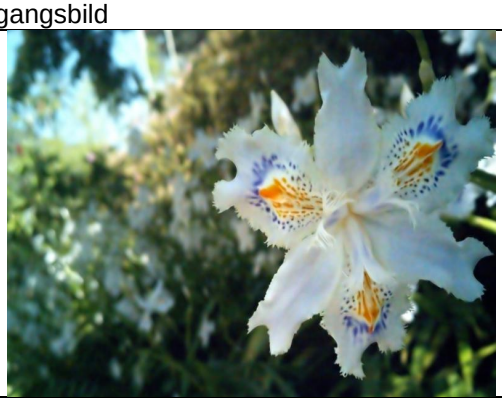
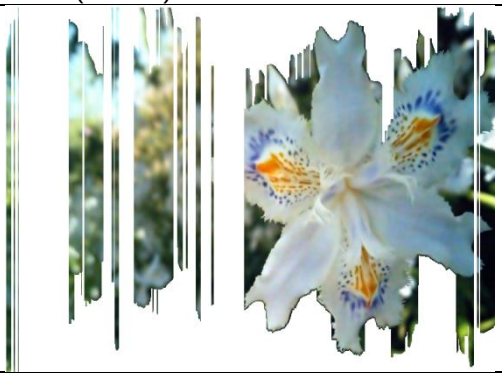
| Ausgangsbild | Resultat 1 (default) |
|---|---|
|  |  |
| **Resultat 2** | **Resultat 3** |
|  |  |

**Tabellen 30, 31:**
**Anwendung des ColorDifferenceWithBacklog-Filters auf ein Bild**

### 3.5.8. Filter - FreeFormSelect

Hinweis: Dieses Bild ist eigentlich überhaupt nicht für diesen Filter geeignet, da der Hintergrund eine wesentlich zu starke Musterung aufweist. Dennoch lassen sich anhand dieses Bildes die Fehler, bzw. die Eigenheiten gut erkennen.
Falls der Kantenschwellwert falsch eingestellt ist, erscheinen viele Hintergrundflächen, welche von starken weissen Linien durchzogen sind. Falls der Freefromschwellwert falsch eingestellt ist, wird überhaupt nichts sinnvolles mehr erkannt, da die Kanten einfach „übersprungen“ werden, oder bei einem zu tiefen Wert viel zu früh anspringen und dann praktisch das gesamte Bild unverändert retour liefern. Die „difference“ Werte beziehen sich auf den maximal möglichen Abstand, zu dem der nächste gültige Punkt liegen darf, ohne dass der aktuelle Punkt verworfen wird. Bei einem zu grossen XDifference Wert werden einzelne Spalten nicht leergeräumt, obwohl sie nicht sinnvoll auszuwerten wären. Dies offenbart im Bild gewisse Punkte die den Konturen entlang „laufen“. Bei einem zu kleinen YDifference Wert wird unter Umständen nicht das gesamte Objekt freigestellt, sondern nur ein Verschnitt davon, da die Punkte auf den Folgespalten nicht zu weit auf der Y-Achse auseinander liegen dürfen (somit kann der Filter nicht mehr wie wild umherspringen und hat gewisse Einschränkungen bezüglich der erreichbaren Höhenunterschiede).

| Konfiguration Resultat 1 (default) | Konfiguration Resultat 2 | Konfiguration Resultat 3 |
|---|---|---|
| `_thresholdEdges = 15`<br>`_thresholdFreeFrom = 2`<br>`_maxDifferenceLastPointX = 1`<br>`_maxDifferenceLastPointY = 100` | `_thresholdEdges = 25`<br>`_thresholdFreeFrom = 2`<br>`_maxDifferenceLastPointX = 1`<br>`_maxDifferenceLastPointY = 100` | `_thresholdEdges = 35`<br>`_thresholdFreeFrom = 2`<br>`_maxDifferenceLastPointX = 100`<br>`_maxDifferenceLastPointY = 20` |

| Ausgangsbild | Resultat 1 (default) |
|---|---|
|  |  |
| **Resultat 2** | **Resultat 3** |
|  |  |

**Tabellen 32, 33:**
**Anwendung des FreeFormSelect-Filters auf ein Bild**

### 3.5.9. Filter - Grayscale

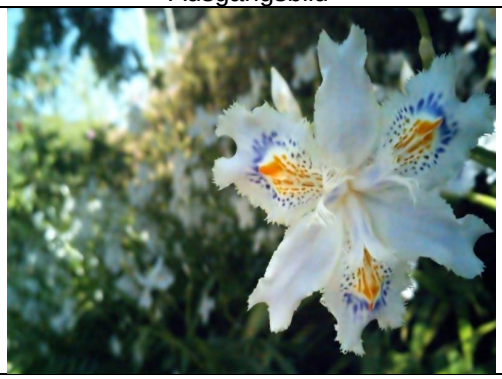
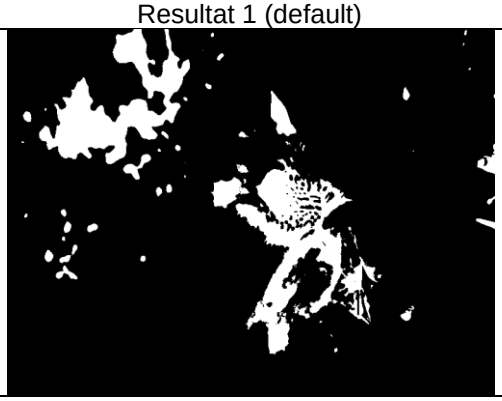
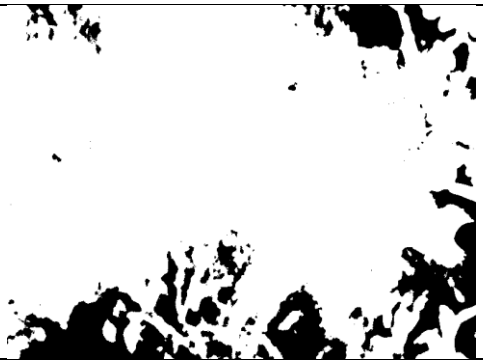
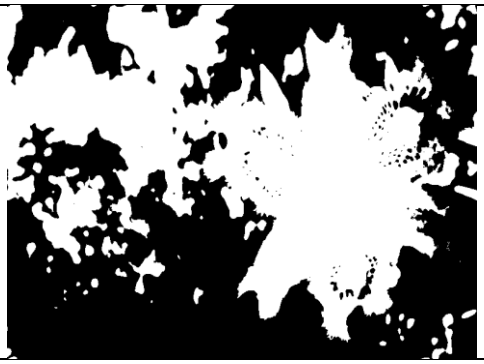
| Ausgangsbild | Resultat (default) |
|---|---|
|  |  |

Tabelle 34:
Anwendung des Grayscale-Filters auf ein Bild

### 3.5.10. Filter - BlackAndWhiteOnly

| Konfiguration Resultat 1 (default) | Konfiguration Resultat 2 | Konfiguration Resultat 3 |
|---|---|---|
| `_red = 189`<br>`_green = 189`<br>`_blue = 189` | `_red = 20`<br>`_green = 20`<br>`_blue = 20` | `_red = 100`<br>`_green = 100`<br>`_blue = 100` |

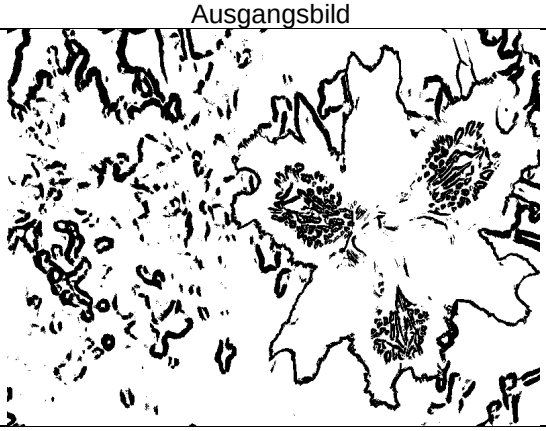
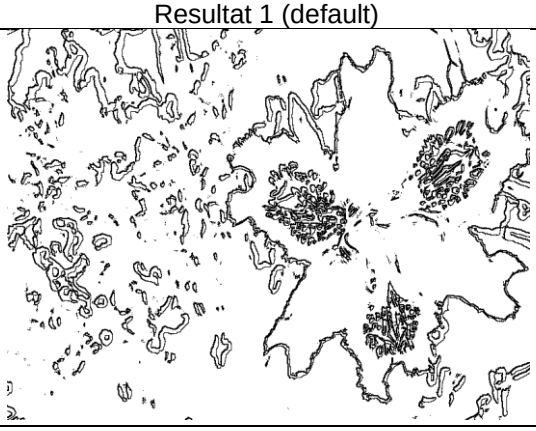
| Ausgangsbild | Resultat 1 (default) |
|---|---|
|  |  |
| **Resultat 2** | **Resultat 3** |
|  |  |

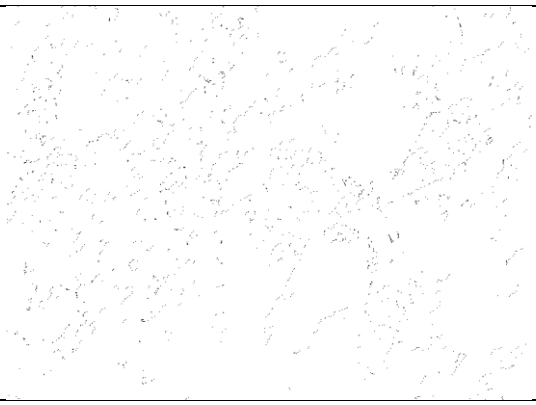
Tabellen 35, 36:
Anwendung des BlackAndWhiteOnly-Filters auf ein Bild

### 3.5.11. Filter - BlackAreaRemover

Hinweis: Da dieser Filter einen schwarz/weiss Input benötigt, wurde ihm ein ColorDifferenceWithBacklog mit einem Threshold von 12 vorgeschaltet.
Falls der „BlackAroundPositionThreshold“ zu hoch gesetzt wird, erkennt dieser Filter keine schwarzen Flächen mehr und gibt als Resultat das unveränderte Original zurück.

| Konfiguration Resultat 1 (default) | Konfiguration Resultat 2 | Konfiguration Resultat 3 |
|---|---|---|
| `_blackAroundPositionThreshold = 15` | `_blackAroundPositionThreshold = 10` | `_blackAroundPositionThreshold = 15` |

| Ausgangsbild | Resultat 1 (default) |
|---|---|
|  |  |

| Resultat 2 | Resultat 3 |
|---|---|
|  |  |

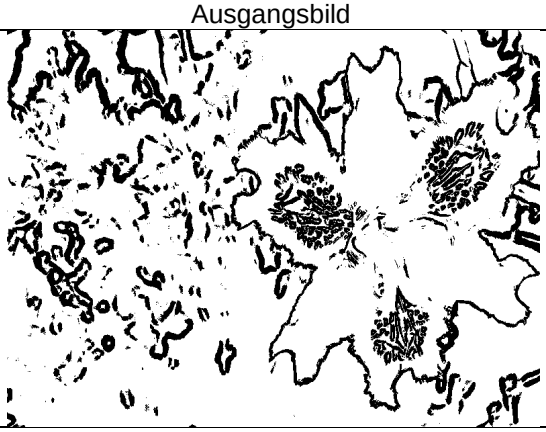
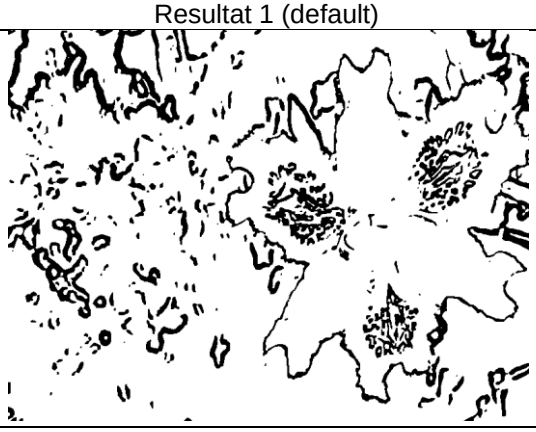
**Tabellen 37, 38:**
**Anwendung des BlackAreaRemover-Filters auf ein Bild**

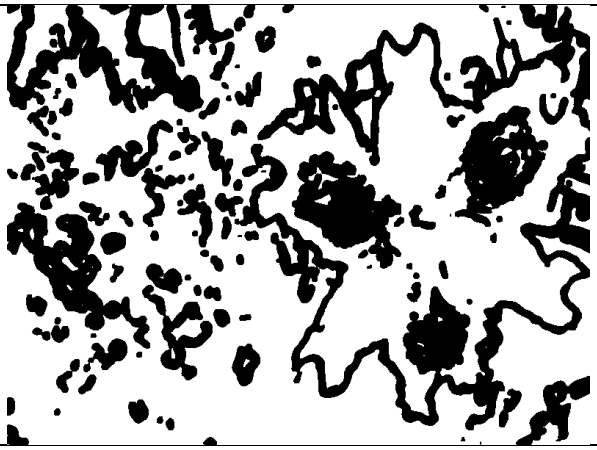
### 3.5.12. Filter - Dot

Hinweis: Da dieser Filter einen schwarz/weiss Input benötigt, wurde ihm ein ColorDifferenceWithBacklog mit einem Threshold von 12 vorgeschaltet.
Dieser Filter reagiert sehr stark auf die Matrixgrösse und weniger auf den Threshold. Wenn die Matrix vergrössert wird, erhöht sich in quadratischer Relation (Flächenzuwachs ist quadratisch) auch die Laufzeit (entspricht ungefähr $T(n) = O(n^2)$); d. h. wenn die Matrixgrösse von 2 auf 4 geändert wird, dauert der Vorgang nun 4x so lange wie zuvor. Daher ist dieser Filter nicht sinnvoll einzusetzen mit grossen Matrixsizes, obwohl dann teilweise eine comichafte Darstellung erfolgt.
In der folgenden Tabelle sind einige Zeitmessungen zum Dot-Filter mit einem 800x600px Bild und einer 3.2 GHz Intel Core i7 CPU gelistet. Es wurde jeweils der Mittelwert dreier Messungen verwendet.

| | |
|---|---|
| benötigte Zeit ohne Filter | ca. 75 ms |
| benötigte Zeit mit Dot (default, matrixSize = 2) | ca. 740 ms |
| benötigte Zeit mit Dot (default, matrixSize = 4) | ca. 2‘050 ms |
| benötigte Zeit mit Dot (default, matrixSize = 8) | ca. 8‘400 ms |

| Konfiguration Resultat 1 (default) | Konfiguration Resultat 2 | Konfiguration Resultat 3 |
|---|---|---|
| `_blackThreshold = 15`<br>`_matrixSize = 2` | `_blackThreshold = 15`<br>`_matrixSize = 4` | `_blackThreshold = 3`<br>`_matrixSize = 2` |

| Ausgangsbild | Resultat 1 (default) |
|---|---|
|  |  |

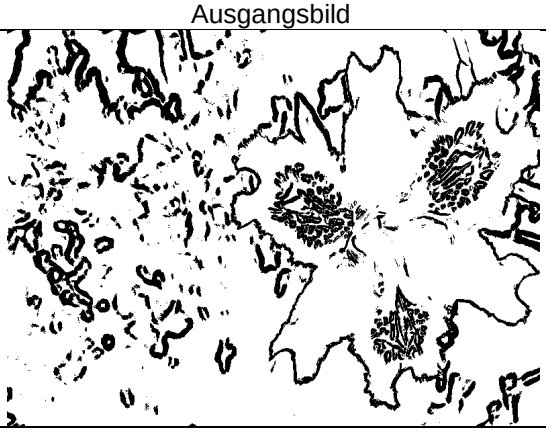
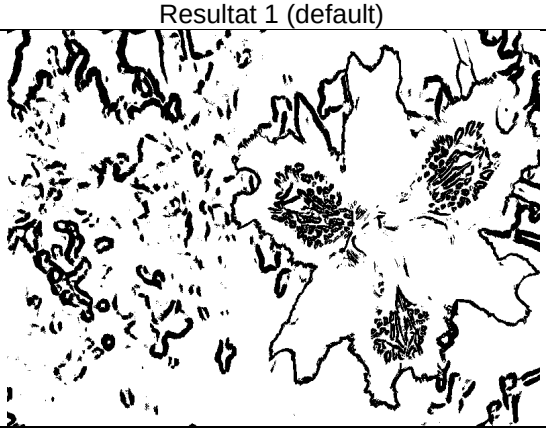
| Resultat 2 | Resultat 3 |
|---|---|
|  |  |

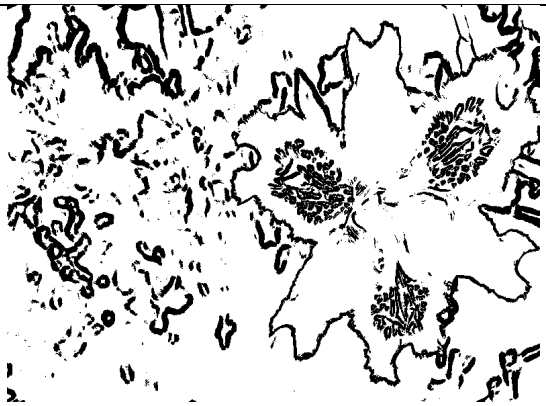
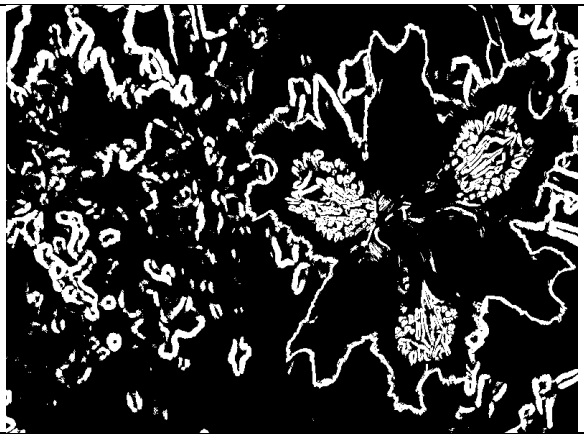
**Tabellen 39, 40:**
**Anwendung des Dot-Filters auf ein Bild**

### 3.5.13. Filter - InvertOnDemand

Hinweis: Da dieser Filter vornehmlich einen schwarz/weiss Input benötigt, wurde ihm ein ColorDifferenceWithBacklog mit einem Threshold von 12 vorgeschaltet.
Falls der Schwellwert zu hoch gesetzt wird, gibt dieser Filter einfach wieder den Input retour.

| Konfiguration Resultat 1 (default) | Konfiguration Resultat 2 | Konfiguration Resultat 3 |
|---|---|---|
| `_thresholdBlackPercentage = = 0.75f` | `_thresholdBlackPercentage = 0.25f` | `_thresholdBlackPercentage = 0.1f` |

| Ausgangsbild | Resultat 1 (default) |
|---|---|
|  |  |

| Resultat 2 | Resultat 3 |
| --- | --- |
|  |  |

Tabellen 41, 42:
Anwendung des InvertOnDemand-Filters auf ein Bild

### 3.5.14. Filter - MoveBWImage

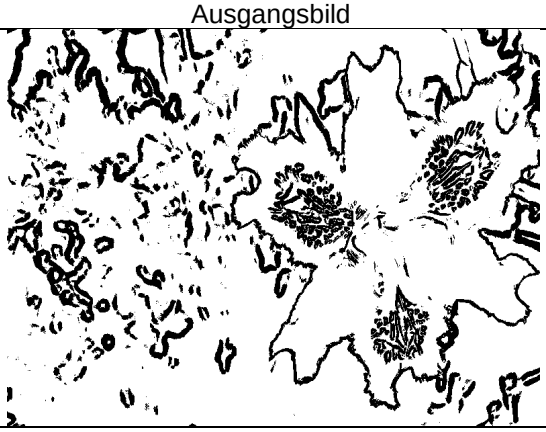
| Ausgangsbild | Resultat |
| --- | --- |
|  |  |

Tabelle 43:
Anwendung des MoveBWImage-Filters auf ein Bild

### 3.5.15. Filter - ColorHistogramGraph

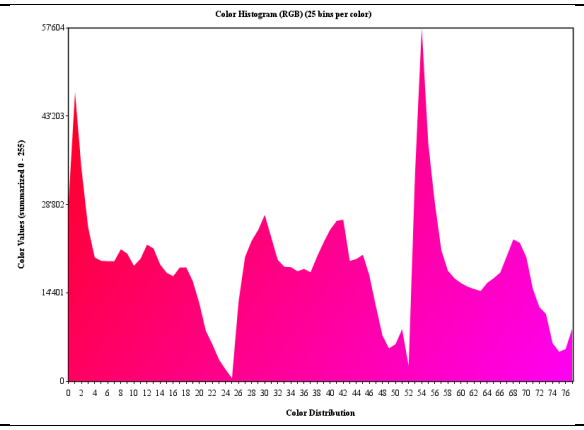
| Ausgangsbild | Resultat |
| --- | --- |
|  |   |

Tabelle 44:
Anwendung des ColorHistogramGraph-Filters auf ein Bild

### 3.5.16. Filter - HorizontalBlackGraph

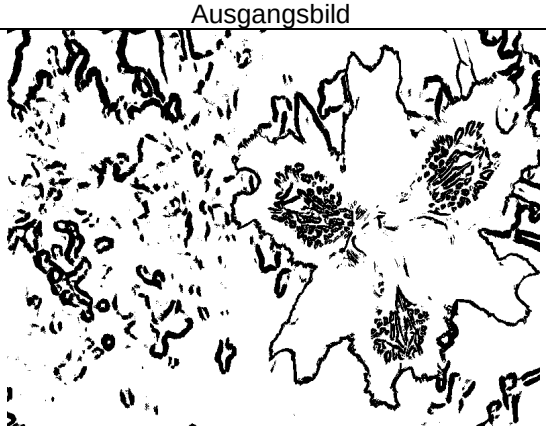
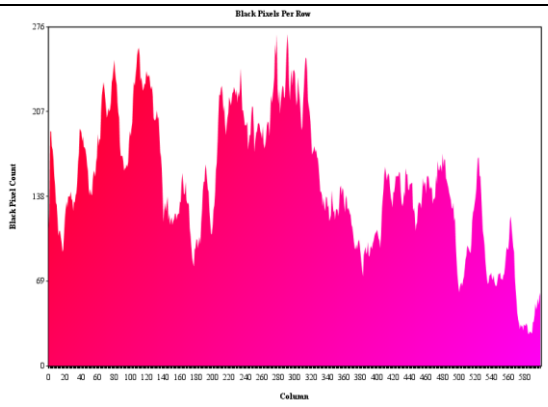
| Ausgangsbild | Resultat |
|---|---|
|  |   |

Tabelle 45:
Anwendung des HorizontalBlackGraph-Filters auf ein Bild

### 3.5.17. Filter - VerticalBlackGraph

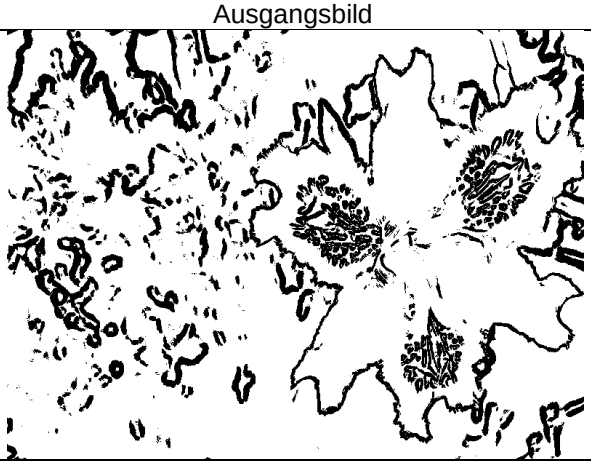
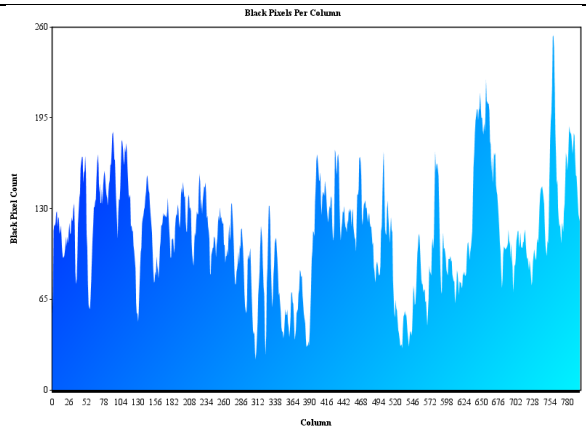
| Ausgangsbild | Resultat |
|---|---|
|  |   |

Tabelle 46:
Anwendung des VerticalBlackGraph-Filters auf ein Bild

## 3.6. Testreihen mit den vordefinierten Filterprofilen

### 3.6.1. Profil „Color Comparison“

Dieses Profil ist in der Datei „Color.ser“ innerhalb des Projekts gespeichert. Es umfasst einen „Stretch“-Filter, sowie das „ColorHistogram“ ComparationModule.
Dieses Profil vergleicht Farben und ist daher für Vergleiche von Landschaften, oder Räumlichkeiten gedacht. Es kann auch für Objekte verwendet werden, dann muss allerdings sichergestellt sein, dass der Untergrund immer die gleiche Farbe aufweist. Ausserdem spielt die Grösse des Objekts im Bild eine wesentliche Rolle und kann nur bei sehr kleinen Datenbanken vernachlässigt werden, da dann andere Faktoren (Primärfarbbereich des Objekts) eine grössere Rolle spielen.

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 92.81868% |

***Kommentar:*** Dieser Vergleich funktioniert gut, da es auch praktisch das gleiche Bild ist. Die knapp 8% Differenz lassen sich durch die leichte Änderung der Belichtung, sowie den schwarzen Bereich rechts oben im zweiten Bild erklären.

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 86.48785% |

***Kommentar:*** Hier lässt sich der Nachteil dieser Methode langsam erkennen, denn beide Bilder enthalten ähnlich viel Anteil der gleichen Farben und daher fällt der Match entsprechend hoch aus, obwohl das Objekt sehr unterschiedlich ist.

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |   | 17.428543% |

***Kommentar:*** Hier ist auch wieder ein typisch schlechtes Resultat, welches entsteht, wenn erstens das Objekt nicht identisch ist und zweitens der Hintergrund verfärbt ist.

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 31.499931% |

***Kommentar:*** Es sind zwar nicht exakt die gleichen Stifte, doch würde dies nicht die 31% rechtfertigen. Dies ist passiert, weil der Hintergrund in diesem Beispiel die tragende Rolle übernommen hat. Man könnte hier sagen, dass der Hintergrund und nicht das Objekt verglichen wurde, daher der schlechte Match.

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 52.550488% |
| ***Kommentar:*** Dieser Vergleich ist etwas konstruiert, aber zeigt immer noch die Fallstricke dieser Vergleichsmethode auf. Beide Bilder enthalten relativ dunkle Töne und daher ist auch ein Match von 52% erreicht worden, obwohl die beiden Bilder unterschiedlicher kaum sein könnten. | | |

**Tabellen 47 - 51:**
**Einige exemplarische Testreihen mit dem „Color Comparison“-Profil**

### 3.6.2. Profil „FreeForm With Color Comparison“

Dieses Profil ist in der Datei „FreeForm_Color.ser“ innerhalb des Projekts gespeichert. Es umfasst einen „Stretch“-Filter, den Filter „FreeForm“, sowie das „ColorHistogram“ ComparationModule.
Dieser Filter eignet sich vor allem für Objekte, welche frei auf einem Tisch oder einer sonstigen Unterlage stehen. Die Unterlage sollte möglichst keine Struktur aufweisen, damit der Freistellvorgang problemlos ablaufen kann. Dier Filter ist nicht für Landschaftsfotos und ähnliches geeignet, da in diesen Situationen kein sinnvolles Objekt mehr freigestellt werden kann und daher die Vergleiche einem Vabanque-Spiel gleichen. Auch können beispielsweise mehrere Farbstifte eines bestimmten Typs, jedoch mit unterschiedlichen Farben, nicht miteinander verglichen werden. Ganz wichtig zu beachten ist, dass die zu verwendenden Objekte nicht über die Kanten des Bildes hinausgehen, da sonst die Freistellung nicht korrekt funktioniert. Damit alle Vergleiche durchgeführt werden konnten, wurde das BadInputImageFlag nicht ausgewertet.

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 95.02625% |
| ***Kommentar:*** Dieser Vergleich funktioniert einwandfrei und liefert sogar noch bessere Resultate, als der reine Farbvergleich, da bei beiden Bildern der Tisch im Hintergrund entfernt wird und somit Farbverfälschungen durch die unterschiedliche Beleuchtung entfernt werden. | | |

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 94.26347% |
| ***Kommentar:*** Hier auch wieder ein typisches Beispiel, welches eine noch höhere Matchpercentage liefert, als beim reinen Farbvergleich. Der bessere Match entsteht durch die entfernte Verfälschung im linken Bild, welche nun nicht mehr die Farbwerte durcheinander bringt. Solche Bilder sind ungeeignet für | | |

| |
|---|
| die Erkennung per Farbe. Wobei hier vielleicht noch angemerkt werden muss, dass in der Regel (wenn die Bilder sauber fotografiert werden) kleine Unterschiede bei der Matchpercentage dennoch für eine eindeutige Identifikation ausreichen können. |

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |   | 82.10264% |
| ***Kommentar:*** Das Stop-Schild wurde nicht korrekt freigestellt, da es über den Rand des Bildes hinaus geht. Dennoch ist die Matchpercentage immer noch um Welten besser, als beim reinen Farbvergleich. Dies ist auch wieder durch den fast schon blauen Hintergrund zu erklären, welcher nun entfernt wurde. | | |

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 89.127914% |
| ***Kommentar:*** Auch hier wieder eine wesentlich bessere Erkennung, jedoch führte die Maserung zusammen mit der Bildunschärfe zu Problemen beim Freistellprozess. Da er jedoch bei beiden etwa gleich schlecht verlief, kommt eine erstaunlich gute Übereinstimmung zu Stande. | | |
| Bild A | Bild B | Matchpercentage |
|  |  | 62.141045% |
| ***Kommentar:*** Die Bilder sind so unterschiedlich wie sie kaum anders sein könnten. Da beim Freistellprozess der Hintergrund durch die Farbe weiss ersetzt wird und dieser nun einen Grossteil des Bildes ausmacht, kommen immerhin noch mehr als 60% zusammen. | | |

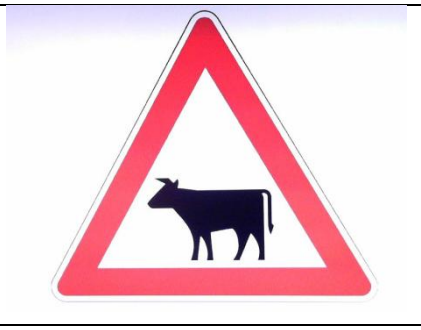
**Tabellen 52 - 56:**
**Einige exemplarische Testreihen mit dem „FreeForm With Color Comparison“-Profil**

### 3.6.3. Profil „FreeForm With Size Comparison“

Dieses Profil ist in der Datei „FreeForm_Size.ser“ innerhalb des Projekts gespeichert. Es umfasst einen „Stretch“-Filter, den Filter „FreeForm“, sowie das „Size2D“ ComparationModule. Dieser Filter eignet sich vor allem für Objekte, welche frei auf einem Tisch oder einer sonstigen Unterlage stehen. Die Unterlage sollte möglichst keine Struktur aufweisen, damit der Freistellvorgang problemlos ablaufen kann. Dier Filter ist nicht für Landschaftsfotos und ähnliches geeignet, da in diesen Situationen kein sinnvolles Objekt mehr freigestellt werden kann und daher die Vergleiche einem Vabanque-Spiel gleichen. Mit diesem Filter können beispielsweise mehrere Farbstifte eines bestimmten Typs, jedoch mit unterschiedlichen Farben, miteinander verglichen werden. Ganz wichtig zu beachten ist, dass die zu verwendenden Objekte nicht über die Kanten des Bildes hinausgehen, da sonst die

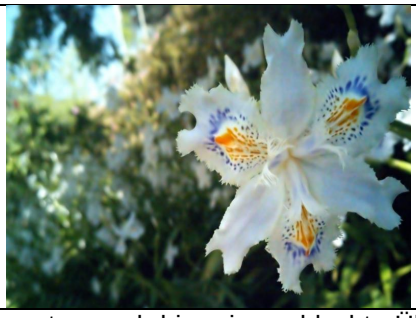
Freistellung nicht korrekt funktioniert. Damit alle Vergleiche durchgeführt werden konnten, wurde das BadInputImageFlag nicht ausgewertet.

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 95.541664% |
| ***Kommentar:*** Dieser Vergleich funktioniert wirklich gut. Der Hintergrund kann gleichmässig entfernt werden, die Farben spielen eine untergeordnete Rolle. Die Fläche der beiden Mäuse ist praktisch gleich, daher auch die hohe Erkennungsrate. | | |

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 97.163956% |
| ***Kommentar:*** Dieser Match trügt, da beim zweiten Bild die Freistellung nicht sauber funktionierte. Dies tritt auf, weil auch bei diesem Bild das Objekt über den Rand des Bildes hinaus geht. Ansonsten wäre der Unterschied wesentlich grösser gewesen und nicht per Zufall beinahe 100%. | | |

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 77.50271% |
| ***Kommentar:*** Auch dieser Vergleich hinkt etwas, da beim Stopp-Schild die Farben über die Kanten hinaus gehen und daher die Freistellung wie zuvor nicht perfekt funktioniert. Dennoch erreicht dieser Size2D Vergleich einen Match von knapp 80%. | | |

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 88.562294% |
| ***Kommentar:*** Beide Bilder werden gleich schlecht freigestellt, daher ist der Match sehr ansprechend. Er bewegt sich in etwa auf dem Niveau vom Profil „FreeForm_Color.ser“. | | |

| Bild A | Bild B | Matchpercentage |
|---|---|---|
|  |  | 62.357502% |
| ***Kommentar:*** Wie nicht anders zu erwarten auch hier eine schlechte Übereinstimmung, doch hier ergibt sie auch einen Sinn. Erstaunlich ist, dass auch hier wieder die Matchpercentage sehr nahe am Match des Profils „FreeForm_Color.ser“ liegt. | | |

**Tabellen 57 - 61:**
**Einige exemplarische Testreihen mit dem „FreeForm With Size Comparison“-Profil**

# 4. Abschliessende Erkenntnisse

## 4.1. Rückblick & Erfahrungen

Dieses Projekt hat mir viele Technologien und Methoden näher gebracht. Ich habe so zum ersten Mal die Funktionsweise von Unschärfe- bzw. Schärfefiltern erarbeiten können und auch Überlegungen zum Vergleich von Bildern konkret ausprogrammiert. Ausserdem habe ich viele Erkenntnisse zum Thema Reflection gesammelt, sowie zum ersten Mal eine eigene Annotationklasse geschrieben.
Es hat mir sehr stark geholfen, dass ich die detaillierten Klassenstrukturen erst relativ spät aufgezeichnet habe. Somit wurde ich von ständigen Überarbeitungen verschont, wenn der Code in der frühen Phase wieder mal komplett umgekrempelt wurde.
Natürlich war immer ein gewisses Risiko vorhanden, dass die geplanten Funktionen nicht implementiert werden hätten können. Ich versuchte dieses Risiko möglichst klein zu halten, indem ich oft im Voraus einen Prototyp für jedes komplexere Element erstellte. Somit konnte ich unabhängig von der Serverumgebung am Prototyp eine Lösung zu einem bestimmten Problem suchen und anschliessend ins Hauptprojekt transferieren.
Ausserdem ist es oft so, dass man gewisse Aufgaben unter bzw. überschätzt. Ich war davon ausgegangen, dass die Filterhandhabung im GUI ungefähr in der dritten oder vierten Woche abgeschlossen sein würde. Dies war grösstenteils auch so, doch flossen realistisch gesehen über den gesamten Projektverlauf immer wieder Verbesserungen in diese Komponente ein. Dagegen gestaltete sich der Code für das Speichern & Laden von Profilen einfacher als gedacht, nachdem ich endlich auch wirklich wusste, wie ich es programmieren wollte.
Dies zeigt beispielhaft auf, dass man beim Programmieren überhaupt nichts erreicht, solange man nicht exakt ein Ziel vor Augen hält. Andernfalls eiert man stundenweise umher und ärgert sich über Belanglosigkeiten.
Dieses Projekt hat mir ausserdem die Augen geöffnet, wie dilettantisch gewisse Softwarefirmen agieren. Adobe Photoshop in der aktuellsten Version CS5.5 hat zum Beispiel einen Kantenhervorhebungsfilter, der nicht mal annähernd zu brauchen ist. Er kreiert störende Viereck-Muster und auch sonst ist er unterdurchschnittlich. Gimp macht dies schon besser und hat erst noch eine grössere Auswahl. Dass nun aber mein eigener Filter mit ca. 200 Codezeilen, welche erst noch nicht stark optimiert sind, ein vergleichbares, teilweise sogar besseres Resultat liefert, als der Filter von Gimp (und Photoshop sowieso), erfüllt mich schon etwas mit Schadenfreude. Ich habe ja schliesslich erst vor ca. drei Jahren programmieren gelernt und nicht schon Jahrzehnte in der Softwarebranche gearbeitet, wie aller Voraussicht nach die Entwickler von Adobe Photoshop.
Zum Schluss sei noch erwähnt, dass eine Arbeit alleine zu schreiben nicht nur Nachteile mit sich bringt. So konnte ich zu jeder Zeit Entscheide fällen und an den Orten arbeiten, an denen gerade das Feuer brannte. Ich musste also nie zeitintensive Rücksprachen halten, nur um einige Aspekte zu klären. In gewissen Situationen wäre es allerdings sehr praktisch gewesen, wenn man einen Mitstreiter gehabt hätte, denn manchmal hat man sprichwörtlich ein Brett vor dem Kopf und steckt in einer Sackgasse. Ich erhielt zum Glück immer wieder Feedback von Mitstudenten und weiteren Personen, welches mir meist wieder zu einer etwas klareren Sicht verhalf.

## 4.2. Fazit

Es ist schön ein derart funktionierendes und flexibles Konzept erschaffen zu haben. Dass ich es auch noch mit Netzwerkanbindung und einem Client für den PC erschaffen konnte, ist für mich eine Bestätigung gegen all jene Kommilitonen, die zu Beginn sagten, dass ich dies nicht hinkriegen würde. Ich hatte eine interessante Zeit in der ich viel gelernt hatte, sie war jedoch auch fordernd und führte zu nicht gerade wenigen schlaflosen Nächten. Daher bin ich mittlerweile froh wenn sich die Arbeit langsam dem Ende zuneigt.

## 4.3. Dankesworte

Dr. Josef M. Joller, Betreuer,
für die Möglichkeit dieses Projekt aus eigenem Interesse durchziehen zu dürfen

Richard Schulthess, Mitstudent,
für die vielen konstruktiven Inputs (Vorschau der Filterkonfiguration, Codequalität)

Ralph Rudolf Baumann, Mitstudent,
für die vielen konstruktiven Inputs (Vergleich aufgrund der 2D-Grösse)

Christian Geissler, ehem. Mitarbeiter,
für die vielen konstruktiven Inputs (Handhabung von schlechten Fotos bei FreeFormSelect)

# 5. Anhang

## 5.1. Proof-of-Concepts

| Name | Aufgabe |
|---|---|
| **comparation** | Erlaubt einen Vergleich von zwei Bildern direkt via Server API durchzuführen. Es finden keine Zugriffe auf Datenbanken resp. Netzwerk statt. Jedes erzeugte Zwischenbild wird auf dem Bildschirm dargestellt. Dieses PoC [4]ist primär für die Analyse des Filterverhaltens gedacht. |
| **comparation_camera** | Ermöglicht einen Vergleich direkt via Kamera auszuführen. Stand-alone ohne Server / Netzwerk / Datenbank Ansteuerung. Es diente ursprünglich dazu, den Zugriff auf externe Hardware und Videos zu testen mit dem VLC Plug-in. |
| **comparation_db** | Erlaubt Vergleich von einem Bild mit den Bildern in der Datenbank. Es ist nur sehr eingeschränkt nutzbar, da es genau das Filterprofil verwenden muss, welches auch während der Datenbankerstellung verwendet wurde. Ursprünglich gedacht, um den Zugriff und Vergleich von Bildern via Datenbank zu testen und analysieren. |
| **dbaccess** | Testet den Zugriff auf die Postgres SQL Datenbank. Erstellt eine Verbindung über den JDBC-Treiber und kontrolliert, ob ein Zugriff auf eine Tabelle möglich ist. |
| **dbaccess_derby** | Testet den Zugriff auf die Apache Derby SQL Datenbank. Erstellt eine Verbindung über den JDBC-Treiber und kontrolliert, ob ein Zugriff auf eine Tabelle möglich ist. |
| **- ab hier deprecated -** | |
| **comparation_network** | Erlaubt einen Vergleich von einem Bild mit denen in einer Datenbank beim Server über Netzwerk. Diente als Vorläufer für den „Client PC“, in dem einige Technologien umgesetzt wurden ***[16]***.<br>Deprecated, weil:<br>Das Protokoll des Servers hat unterdessen geändert und diese Implementation wurde nicht mehr angepasst. |
| **dbreader** | Liest lowlevel von der Datenbank. Dies war ein PoC welches zeigen sollte, ob der Zugriff auf BLOBs praktikabel ist.<br>Deprecated, weil:<br>Die Struktur der Datenbank hat unterdessen geändert und daher funktionieren die Aufrufe nicht mehr korrekt. |
| **dbviewer** | Ruft alle Bilder aus der Datenbank ab. Ursprünglich war dieses PoC für die Kontrolle der Datenbank gedacht; als der Code jedoch langsam zuverlässig zu laufen begann, wurde dieses Tool überflüssig. |

[4] Proof-of-Concept

| | |
|---|---|
| | Deprecated, weil:<br>Die Datenbankstruktur hat sich geändert (es können seit Einführung der Löschfunktion einzelne Zeilen entfernt werden in der Datenbank und diese entfernten Zeilen führen zu Sprüngen im Index), daher kann dieses Programm die Datenbank nicht mehr korrekt ansteuern. |
| **dbwriter** | Schreibt lowlevel in die Datenbank. Dies war ein PoC welches zeigen sollte, ob die Speicherung von BLOBs praktikabel ist.<br>Deprecated, weil:<br>Die Struktur der Datenbank hat unterdessen geändert und daher funktionieren die Aufrufe nicht mehr korrekt. |
| **fileaccess** | Testet den Zugriff auf mehrere Dateien bzw. Quellen, die intern und extern liegen. Ist als Kontrolle gedacht, ob die Pfade auch noch nach Auslagerung in ein RunnableJAR korrekt funktionieren.<br>Deprecated, weil:<br>Einige Pfade funktionieren nicht entsprechend, wenn dieses PoC als RunnableJAR ausgeführt wird. Diese müssten angepasst werden. Da aber die betreffenden Pfade schlussendlich doch nicht im Programm verwendet werden, wurde das PoC fallen gelassen.<br>Ursprünglich war geplant, Profile dynamisch im JAR hinzuzufügen, doch dies ist nur sehr kompliziert umzusetzen (wenn überhaupt), da das JAR ja von der JavaVM zur Laufzeit geladen wird und daher nicht bearbeitet werden kann. Daher wurde entsprechend herum experimentiert, doch als die Erkenntnis endlich kam, dass man nur mit riesigem Aufwand dies implementieren kann und die Alternative (jetzige Implementation) keine gravierenden Nachteile aufwies, wurde dieses PoC schnellstmöglich über Bord geworfen. |
| **filterchoosergui** | Ein erster Entwurf des kompletten Filterdialogs. Die Filterauswahl-Grundfunktionen sind praktisch voll funktionsfähig.<br>Deprecated, weil:<br>Diese Implementation wurde anschliessend in das Server GUI eingebettet und dann laufend erweitert. Mittlerweile ist das ChooserGUI so stark überarbeitet, dass es nicht mehr viel mit dieser Variante zu tun hat. Da der Aufwand zwei solche Implementationen parallel zu warten zu gross ist und dieses PoC auch seinen „Zweck“ erfüllt hat, wurde es nicht mehr weiter aktualisiert. |
| **servergui** | Ein erster Entwurf des kompletten ServerGUIs. Die Grundkomponenten sind praktisch vollständig implementiert.<br>Deprecated, weil:<br>Diese Implementation wurde anschliessend in das Server GUI eingebettet und dann laufend erweitert. Mittlerweile ist das ServerGUI so stark überarbeitet, dass es nicht mehr viel mit dieser Variante zu tun hat. Da der Aufwand zwei solche Implementationen parallel zu warten zu gross ist und dieses PoC auch seinen „Zweck“ erfüllt hat, wurde es nicht mehr weiter aktualisiert. |

**Tabelle 62:**
**Kurzbeschrieb aller im Projektverlauf erzeugten Proof-of-Concept Demos**

## 5.2. Verwendete Plug-ins

### 5.2.1. Video- & Kameraansteuerung

| Name | VLCj (Open-Source) |
|---|---|
| **Quelle** | http://code.google.com/p/vlcj/ |
| **Vorteile** | - ist mit allem kompatibel (spielt alles ab was VLC auch kann)<br>- einfach anzusteuern, wenn‘s denn mal läuft<br>- schnell (kann 720p h264-Videos ohne Probleme im Java GUI abspielen)<br>- verwendet äusserst bewährte Basis (VLC media player) |
| **Nachteile** | - Mühsame Basteleien mit den Pfaden, bis es läuft (benötigt VLC Pfad)<br>- x86 und x64 Versionen benötigen unterschiedliche JavaVMs / VLCs<br>- Entwicklung schwankt sehr stark, viele Bugs im Plugin vorhanden<br>- sogar die grundlegendsten Methodenaufrufe ändern teilweise von Version zu Version |
| **Wie wurde es eingesetzt ?** | Es ermöglicht im PC Client Screenshots von Kameras oder Videos direkt an den eyedentificator Server zu senden. |
| **Hinweise** | Es müssen zwingend die JavaVM x86, sowie VLC x86 verwendet werden. Andernfalls erscheinen komische Fehlermeldungen, welche in die Irre führen. In früheren Versionen konnte man einfach den Pfad anpassen, dann lief es auch unter x64 mit VLCx64 aber es war äusserst buggy und crashfreudig. Daher wurde wahrscheinlich der Support von x64 temporär gestrichen bzw. etwas verborgen. |

Tabelle 63:
Detailinformationen über das VLCj Plug-in

### 5.2.2. Datenbank 1

| Name | Postgres |
|---|---|
| **Quelle** | http://jdbc.postgresql.org/download.html |
| **Vorteile** | - ist Open-Source<br>- einfach anzusteuern<br>- bewährte Technologie |
| **Nachteile** | - die GUI-basierte Management-Anwendung ist langsam und teilweise unausgereift<br>- Datenbank muss separat aufgesetzt und konfiguriert werden |
| **Wie wurde es eingesetzt ?** | Die ganze Datenbankansteuerung läuft über dieses Plugin (optional), welches die Verbindung zum PostgreSQL Server herstellt. |
| **Hinweise** | Es läuft einfach, keine Basteleien notwendig.<br>Wenn ResultSet Objekte angefordert werden, zuerst ein .Next() aufrufen, damit man überhaupt auf eine gültige Position zeigt. |

Tabelle 64:
Detailinformationen über das Postgres Plug-in

### 5.2.3. Datenbank 2

| Name | Apache Derby |
|---|---|
| **Quelle** | http://code.google.com/p/vlcj/ |
| **Vorteile** | - ist Open-Source<br>- einfach anzusteuern<br>- bewährte Technologie<br>- Datenbank direkt als Datei speicherbar, daher keine externe Konfiguration notwendig<br>- Portabilität der Daten innerhalb der Datenbank |
| **Nachteile** | - etwas komplexer in der Verwendung als PostgreSQL<br>- beherrscht viel weniger unterschiedliche Datentypen als PostgreSQL<br>- verlangt eine striktere SQL-Syntax als PostgreSQL |
| **Wie wurde es eingesetzt ?** | Die ganze Datenbankansteuerung läuft über dieses Plugin, welches die Verbindungs- und Datenbanklogik des Apache Derby SQL Servers enthält. |
| **Hinweise** | Es muss ein sinnvoller Pfad angegeben werden für den Speicherort der Datenbank. |

Tabelle 65:
Detailinformationen über das Apache Derby Plug-in

### 5.2.4. Diagramme

| Name | jCharts |
|---|---|
| **Quelle** | http://sourceforge.net/projects/jcharts/files/jCharts/ |
| **Vorteile** | - ist Open-Source<br>- ermöglicht ansprechende Diagramme<br>- Diagramme sind stark individualisierbar |
| **Nachteile** | - enthält noch so einige Fehler (kreiert dann einfach weisses Diagramm)<br>- Verwendung oft durch Trial & Error, dank schlechter & inkonsistenter Dokumentation<br>- StackTraces werden geliefert, falls eine Aufrufkonvention nicht eingehalten wurde |
| **Wie wurde es eingesetzt ?** | Diagramme, welche in den Visualization Filtern eingesetzt werden. |
| **Hinweise** | Immer von einem Beispiel ausgehend starten und Zeile für Zeile anpassen. Nach JEDER Änderung die Lauffähigkeit überprüfen. Einzelne Methodenaufrufe erwarten die korrekte Arraygrösse, ausgehend von einer Parameterliste , welche an einem komplett anderen Ort verwendet wird. |

Tabelle 66:
Detailinformationen über das jChart Plug-in

### 5.2.5. Pfadmanagement für VLC

| Name | Java JNA [5]/ JNA Plattform |
|---|---|
| **Quelle** | https://github.com/twall/jna |
| **Vorteile** | - ist Open-Source<br>- ermöglicht Klassenpfade zur Runtime zu laden |
| **Nachteile** | - keine, welche die Verwendung in diesem Projekt betreffen |
| **Wie wurde es eingesetzt ?** | Es wird von VLCj vorausgesetzt, damit er zur Laufzeit seine VLC Pfade managen kann. |
| **Hinweise** | - |

Tabelle 67:
Detailinformationen über das JNA Plug-in

## 5.3. JavaDoc

*siehe separate Dokumente in HTML-Form*

[5] JNA, Java Native Access