Seminararbeit zum Semesterprojekt: Zählen von Kreisen

Entwicklung von Multimediasystemen

Fadi Moukayed / 538502 SS2014
Max Jürgens / 540025 Bauer

Inhaltsverzeichnis

Seminararbeit zum Semesterprojekt: Zählen von Kreisen

Entwicklung von Multimediasystemen

Fadi Moukayed / 538502 SS2014
Max Jürgens / 540025 Bauer

1. Projektbeschreibung

Ziel dieses Projektes ist es, eine Anwendung für die Erkennung von Kreisformen innerhalb von Rasterbildern zu entwickeln. Die Anwendung soll für ein Eingabe-Bild die wahrscheinliche Anzahl der Kreise anzeigen und auf dem Bild farblich hervorheben. Zusätzlich dazu soll die Anwendung dem Benutzer erlauben, die Parameter der Erkennungs- und Filteralgorithmen anzupassen und die Ergebnisse einer Erkennungsoperation zu speichern. Die Anwendung unterstützt das Importieren und Exportieren von Bildern in den gängigen Rasterbildformaten (PNG, JPEG, GIF).

Für die Realisierung der Benutzungsoberfläche wird das Qt-Framework verwendet.

Fadi Moukayed / 538502 SS2014
Max Jürgens / 540025 Bauer

2. Architektur/Modellierung des Backends

Da die einzelnen Bearbeitungsschritte sequentiell hintereinander ablaufen und der Output eines Bearbeitungsschrittes gleichzeitig auch der Input des nächsten Bearbeitungsschrittes ist, wurde der Backend-Teil der Anwendung als komponierbare Filter-Pipeline modelliert. Diese Modellierung ermöglicht eine einfache, seiteneffektfreie Repräsentation des gesamten Erkennungsablaufs als Komposition einzelner Filterstufen.

Die Filterbibliothek (`raster::*`) setzt sich zusammen aus den folgenden Komponenten:

- Die `Matrix`-Abstraktion für die Repräsentation eines Pixelrasters
- Operationen und Filterkerne auf `Matrix`-Objekte
- Verknüpfungen (`Map, Translation, Convolution`), die eine Verknüpfung einer Matrix-Operation/eines Filterkerns mit einer Bildmatrix repräsentieren

Die Backend-Frontend-Verknüpfung wird durch eine „Adaptermatrix“ realisiert, die eine `Matrix`-Schnittstelle für `QImage` implementiert und somit die Verwendung der Filterfunktionen auf `QImage`-Instanzen ermöglicht.

Um die Laufzeit der Filteralgorithmen zu optimieren, wurde hier auf die dynamische Polymorphie verzichtet. Stattdessen wurden die für die statische Polymorphie vorgesehenen C++-Sprachkonstrukte (Templates, Overloading) verwendet – die Neuerungen von C++11 machen eine solche Vorgehensweise praktikabel ohne die Nachteile, die bei früheren C++-Standards aufgetreten wären. Zurzeit wird `raster`-Teilbibliothek nur für die Kantendetektion und die damit verbundenen Filterpässe verwendet, jedoch ist eine Erweiterung und/oder eine anderweitige Nutzung im Kontext einer anderen Anwendung denkbar.

Um die Entkopplung möglichst gering zu halten, wurden die Erkennungsalgorithmen als eigenständige Module entwickelt: Das `circles::*`-Modul enthält Plotting- und Hough-Algorithmen für Kreise, und das Modul `ellipses::*` enthält Plotting-Funktionen für Ellipsen sowie eine experimentelle Implementierung eines Ellipsenerkennungsalgorithmus. Genau wie die Raster-Filterbibliothek sind diese Module Template-Basiert und mit üblichen C++-STL Abstraktionen wie Iterators interoperabel.

Seminararbeit zum Semesterprojekt: Zählen von Kreisen

Entwicklung von Multimediasystemen

Fadi Moukayed / 538502 SS2014
Max Jürgens / 540025 Bauer

2.1. Die Matrix-Abstraktion

Ein Typ ***T*** erfüllt die Voraussetzungen des `Matrix`-Konzepts, wenn es die folgenden Operationen im Overload-Set hat:

- get(***T,RowT,ColumnT***) → ***ValueT***
- set(***T,RowT,ColumnT,ValueT***) → ***ValueT***
- rows(***T***) → ***RowT***
- columns(***T***) → ***ColumnT***

Standardmäßig implementiert die `raster`-Bibliothek (In *raster/types/**) die Matrix-Operationen für Zweidimensionale-Arrays (**T[M][N]** und `std::array<std::array<T,N>,M>`), gekapselte Referenzen/Zeiger auf Matrizen (`std::unique_ptr<T>`, `std::shared_ptr<T>`, …) und für Objekte mit den entsprechenden Member-Methoden (*raster/types/object.hh*). Eine Typ kann als `Matrix` verwendet werden, wenn die oben genannten Funktionen als freie Funktionen, oder alternativ als Member-Funktionen des Typs vorhanden sind.

2.2. `Matrix`-Verknüpfungen

In der `raster`-Bibliothek werden die Matrixtransformationen als verschiedene „Verknüpfungen“ realisiert. Folgende Verknüpfungsarten können genutzt werden:

- ***map*** (*raster/map.hh*): Verknüpfung einer unären Funktion *g* mit einer Matrix. Die Ergebnismatrix ist das Ergebnis der Anwendung von *g* auf jede Zelle der Input-Matrix.
- ***conv*** (*raster/conv.hh*): Verknüpfung einer Matrix *f* mit einer Kernel-Matrix *k.* Die Ergebnismatrix ist das Ergebnis der Faltungsoperation *(f * k)*
- ***translation*** (*raster/translation.hh*): Verknüpfung einer Matrix *f* mit einer ternären Funktion *g(f,y,x)*; Ermöglicht die Manipulation der Zugriffe sowie die Rückgabewerte.

Um die Speichereffizienz zu steigern und eine Form der verzögerten Evaluation („Lazy Evaluation“) zu ermöglichen, wird die Ergebnismatrix beim Aufruf der Verknüpfungsoperation nicht direkt berechnet, sondern Zellenweise beim Aufruf der Zugriffsoperation *get*.

2.3. Filter-Pipelines

Durch Komposition der zuvor genannten Verknüpfungen ist es möglich, Rasterbildtransformationen abzubilden – so wird z.B. der in diesem Projekt verwendete Canny-Filter als Pipeline in der Form von ***hysteresis(nms(gaussian(greyscale(InputMatrix))))*** aufgebaut.

Fadi Moukayed / 538502 SS2014
Max Jürgens / 540025 Bauer

2.4. Erkennungsalgorithmen

Erkennungsmodule `circles::` ***(utils/circles.hh)*** und `ellipses::`***(utils/ellipses.hh)*** sind vollständig von anderen Teilen des Projekts entkoppelt und machen keine Annahmen über die Repräsentation der Eingabedaten – durch die Verwendung einer Push-Style API mit STL-Iteratoren und Functors kann der API-Client entscheiden, wie Eingabe- und Ausgabedaten gespeichert und repräsentiert werden.

Als Kreiserkennungsalgorithmus wurde der Hough-Transform verwendet. Für die Erkennung von Ellipses wird eine Teilimplementierung des Randomized Hough Transform [Inverso06] Verfahren verwendet.

Fadi Moukayed / 538502 SS2014
Max Jürgens / 540025 Bauer

3. Funktionalität und Entwurfsentscheidungen

3.1. Ermittlung der Konturen/Kanten und S/W-Konversion

Während der Entwicklung wurde festgestellt, dass die Verwendung eines einfachen Sobel-Filters als Vorstufe für die Feature-Erkennung nicht ausreichend ist: Trotz Thresholding des Output-Bildes war die Genauigkeit der ermittelten Konturen zu gering, und die Anzahl der erkannten Kanten zu hoch. Da der Aufwand der verwendeten Erkennungsalgorithmen linear abhängig ist von der Anzahl der Bildkanten, wurde der im Anforderungsdokument vorgeschlagene Sobel-Filter zu einem parametrierbaren Canny-Filter erweitert, sodass die Kanten eines Input-Bildes möglichst genau erkannt werden können.

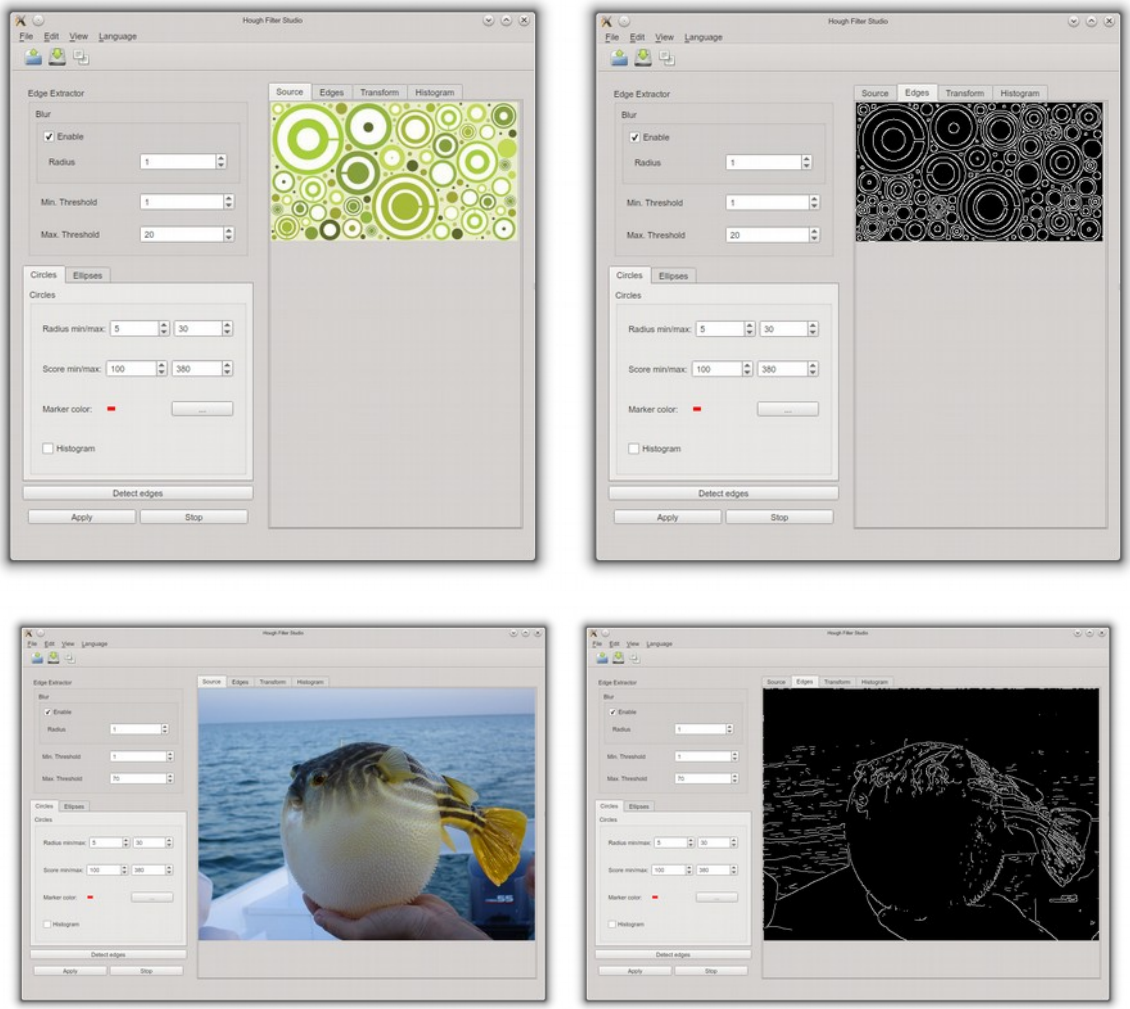

Abb. 1: Beispiele für die Kantenerkennung – Geometrische und Fotografische Bilder

Durch Anpassung der Threshold-Werte hat der Benutzer zusätzlich noch die Möglichkeit, das Ergebnis der Kantenerkennung zu verbessern – was bei fotografischen Bildern häufig erforderlich ist (im Foto des Kugelfisches war ein Threshold-Wert von 70 ausreichend, um die meisten Rauscheffekte zu eliminieren).

3.2. Erkennung von Kreisen

Nach der Kantenerkennung kann die Kreiserkennung ausgeführt werden. Der Suchraum kann durch die Eingabefelder „Radius min/max." beschränkt werden; Die „Score"-Felder erlauben die Festlegung der Erkennungsgenauigkeit. Die Wahl der Hervorhebungsfarbe durch den Benutzer wird auch unterstützt.

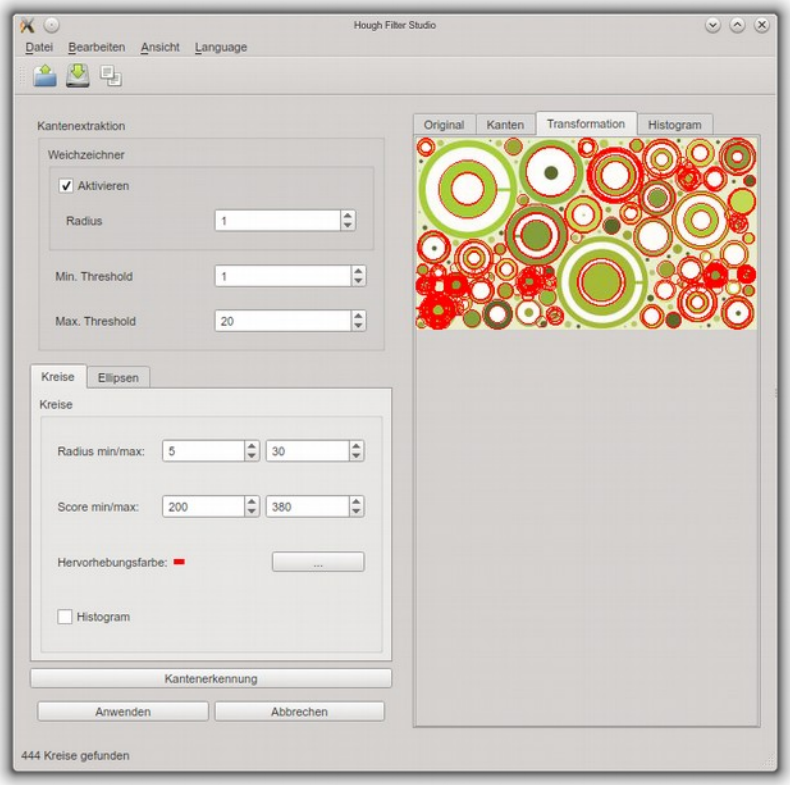

Abb. 2: Kreiserkennung

In der Statusleiste wird die zu vermutende Anzahl der gefundenen Kreise angezeigt. Es ist zu beachten, dass dieser Wert lediglich die Anzahl der Kreise darstellt, die die eingestellten Bedingungen erfüllen – so wird z.B. ein Kreis mit einer Umfangsdichte von 4px als 4 Kreise erkannt (jeweils mit einer Radiusdifferenz von +1px) und entsprechend hervorgehoben.

Die Inhalte der erkannten Kreise, hier als „Kreismaske" bezeichnet, können in die Zwischenablage kopiert werden oder als Bilder gespeichert werden. Bei der Wahl eines Export-Formats mit Alpha-Kanal-Unterstützung sind alle Pixel außerhalb der erkannten Kreise transparent.

Fadi Moukayed / 538502 SS2014
Max Jürgens / 540025 Bauer

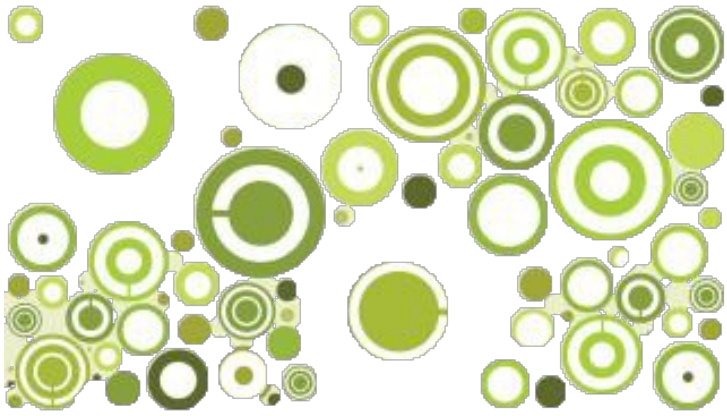

Abb. 3: „Kreismaske" für den Erkennungsergebnis von Abb. 2

3.3. Histogramm

Wahlweise kann auch ein Histogramm als Teil eines Erkennungsvorgangs generiert werden. Beim Histogramm handelt es sich um eine grafische Repräsentation des Hough-Accumulator-Arrays. Jede Zelle der Accumulator-Matrix wird hier als Pixel dargestellt: Zellen mit hohem Score (Trefferwert) werden als helle Pixel dargestellt; Zellen mit niedrigem Score werden als dunkele Pixel dargestellt. In der Statusleiste wird der kumulierte Wert der (als Pixel dargestellten) Accumulator-Zelle unterhalb des Mauszeigers angezeigt.

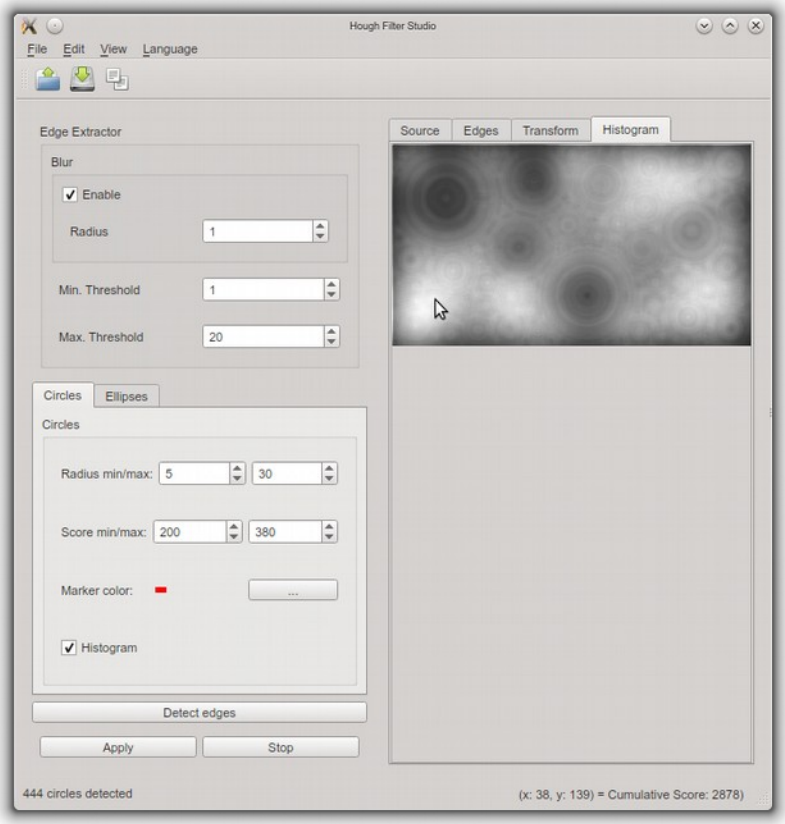

Abb. 4: Histogramm

3.4. Kamera-Aufnahmen

Als zusätzliche Bildquelle können auch Camera- oder Webcam-Schnapschüsse verwendet werden. Das Aufnahmedialog kann über das Menü *Datei→ Webcam-Schnapschuss* aufgerufen werden.

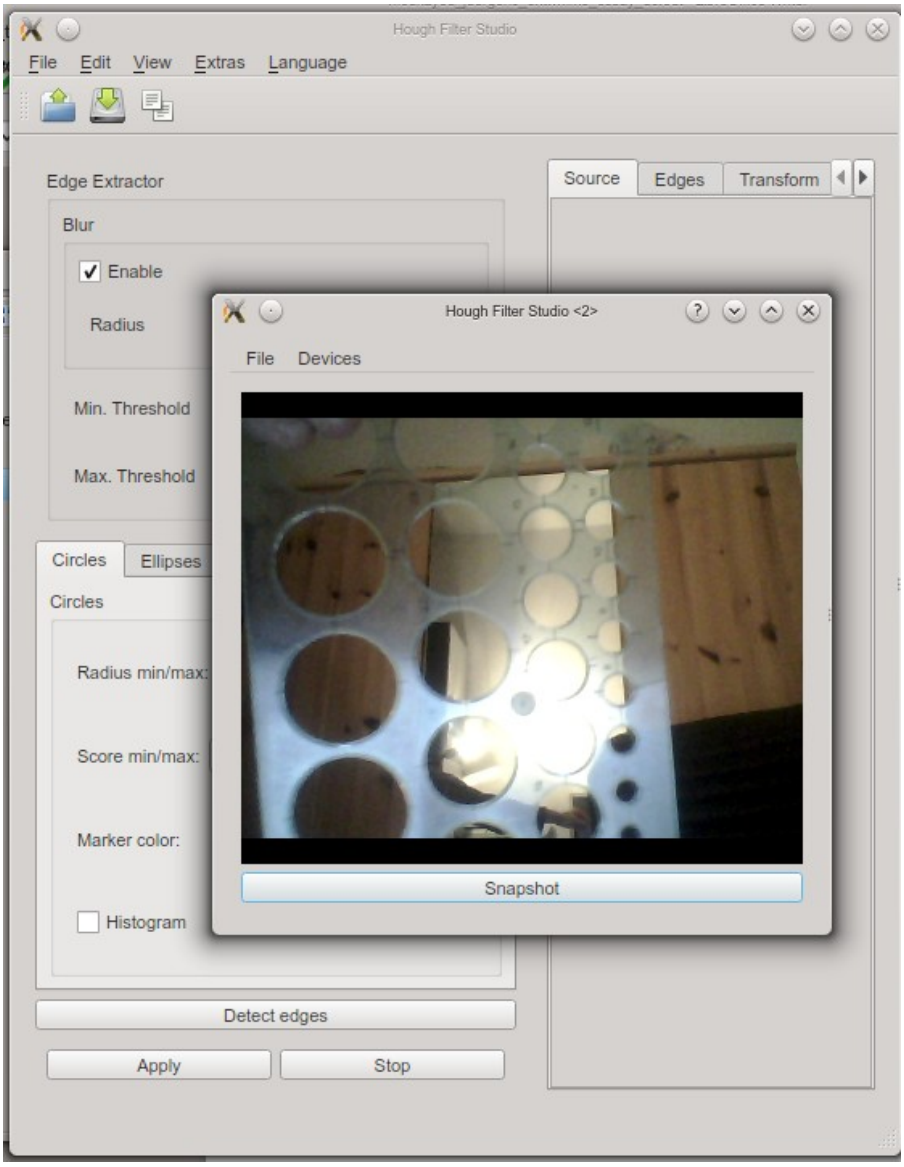

Abb. 5: Webcam-Capture

3.5. TTS (Text-To-Speech)

Über Extras→ Sprachausgabe kann die Sprachausgabe aktiviert werden. Wenn der Sprach-Synthesizer *festival*[1] installiert ist, wird die Anzahl der erkannten Kreise gesprochen ausgegeben werden.

1 http://www.cstr.ed.ac.uk/projects/festival/

Fadi Moukayed / 538502 SS2014
Max Jürgens / 540025 Bauer

4. Testergebnisse (Speicherverwaltung)

Die Backend-Funktionalität wurde mithilfe von *AddressSanitizer* auf Allokations- und Speicherverwaltungsfehler geprüft. Alle Tests im */tests*-Verzeichnis werden standardmäßig mit *AddressSanitizer*-Unterstützung kompiliert – somit schließt die Ausführung der Test-Cases auch die Überprüfung durch AddressSanitizer ein.

Die letzten Testergebnisse sind in der Datei *test/testlog.txt* vorhanden.

5. Checkliste: Features/Anforderungen

5.1.1. Allgemeine Anforderungen

Anforderung	Kommentar
Qt-Widget-Anwendung mit einem Hauptfenster	-
Dynamische Fenstergröße	-
Vollbild-Modus	Menü Ansicht → Vollbild-Modus
Lokalisierung	Menü Language/Sprache → Deutsch / English
Interaktivität	Anwendung verwendung QThreads

5.1.2. Anforderungen an den Entwicklungsprozess

Anforderung	Kommentar
Versionsverwaltungssystem	GIT (Host: Github, URL: github.com/kochab/qtfeaturerecog)
Dokumentation	Doxygen (Mit `qmake && make doc`)
Entkopplung / Wiederverwendbarkeit	Verwendung von Templates, Iterators, …
Unit-Tests	Verzeichnis tests/; Benötigt das Google Unit Testing Framework
Memory/Allocation-Check	Als Teil der Unit-Tests vorhanden (AddressSanitizer)

5.1.3. Projektspezifische Anforderungen

Anforderung	Kommentar
S/W-Bild als Input per Dateiauswahl	Keine spezielle Behandlung für Monochrome-Bilder nötig
S/W-Bild als Input per Drag-And-Drop	Keine spezielle Behandlung
Hervorhebung von Formen	Farbe frei wählbar
Export der Kreisinhalte	Datei → Kreisinhalte speichern

Anforderung	Kommentar
Histogram	Histogram-Tab (Wenn aktiviert)
Histogram-Explorer	Statusleiste (Zeigt Position und Accumulator-Wert)
Unterstützung für Farbbilder	RGB-Rasterbilder, Kantenextraktion (Canny) vom Benutzer gesteuert
Bildformatunterstützung	Gängige Rasterbildformate (PNG,JPG,GIF); Verwendet QImage/QPixmap
Kamera-Aufnahme	Datei → Kamera-Aufnahme
Text-To-Speech	Extras → Sprachausgabe (*festival* muss installiert sein)

5.1.4 Eigene Anforderungen

Anforderung	Kommentar
Benutzergesteuerte Kantenextraktion	Anpassbarer Canny-Filter (Threshold, Weichzeichner, …)
Kantenansicht	Als Bild abspeicherbar
Histogram als Bild speichern	-

6. Errata

Die Anwendung enthält eine Teilimplementierung der „Randomized Hough Transform“-Algorithmus [Inverso06] – da diese Teilimplementierung nur die ersten beiden Schritte (Parametrisierung und Scoring) umfasst, ist sie leider für komplexe Bilder mit einer hohen Kantenanzahl nicht brauchbar – deshalb wurde diese Funktion als „experimentell“ markiert und aus der Anforderungsliste nachträglich entfernt.

Seminararbeit zum Semesterprojekt: Zählen von Kreisen

Entwicklung von Multimediasystemen

Fadi Moukayed / 538502
Max Jürgens / 540025

SS2014
Bauer

I. Quellenverzeichnis

Ellipse Detection Using Randomized Hough Transform, Samuel S. Inverso, 2006, URL: http://www.saminverso.com/res/vision/EllipseDetection.pdf

The Festival Speech Synthesis System, University of Edinburgh, URL: http://www.cstr.ed.ac.uk/projects/festival/ (12.07.2014)