

Teil V

Praktika

Praktikum 1: Lineare Filter und GUI

In diesem Praktikum sollen Sie die Anwendung linearer Filter sowie arithmetischer Operationen auf Bilder einüben. Darüber hinaus werden Sie vor allem hilfreiche Techniken bei der Erstellung grafischer Oberflächen (*graphical user interface, GUI*) kennenlernen.

14.1 Übersicht

Abbildung 14.1 zeigt exemplarisch das zu erstellende Programm. Es ist ein Fenster mit einer „geteilten“ Ansicht umzusetzen, bei der der Bildbereich links einer vertikalen Trennlinie (im Beispiel rot) mit einer Bildverarbeitungsmethode prozessiert, rechts hingegen unverändert angezeigt wird. Die Linie kann mittels der Maus verschoben werden. Hierdurch kann man einfach und schnell zwischen dem Ausgangsbild und dem verarbeiteten Bild wechseln, wodurch man einen sehr guten Eindruck des Effektes der Bildverarbeitung erhält.

Als Verfahren soll die Erhöhung der Bildschärfe durch sogenannte *unscharfe Maskierung* bzw. *Unsharp masking* umgesetzt werden. Der Grad der Erhöhung kann in der grafischen Oberfläche über einen Schieberegler variiert werden. Abbildung 14.2 zeigt jeweils ein Beispiel, in dem das Ausgangsbild nicht verändert wird (Schieberegler ganz links) und in dem eine maximale Erhöhung der Bildschärfe angewendet wird (Schieberegler ganz rechts).

14.2 Unscharfe Maskierung

Der in der unscharfen Maskierung umgesetzte Ansatz zur Erhöhung von Bildschärfe geht auf die analoge Filmfotografie zurück, findet aber auch in der digitalen Bildbearbeitung Anwendung. Das Verfahren beruht darauf, dass Strukturen wie beispielsweise Kanten einer starken lokalen Veränderung der Bildwerte entsprechen. Je stärker und steiler jedoch die lokale Änderung eines Signals ist, desto ausgeprägter sind seine hohen Frequenzanteile. Bereiche wie Flächen oder sanfte Übergänge besitzen hingegen kaum hohe Frequenzanteile.

Vor diesem Hintergrund kann man daher Bildschärfe erhöhen, indem hochfrequente Anteile verstärkt werden. Die hochfrequenten Anteile erhalten wird durch Filterung des Ausgangsbildes

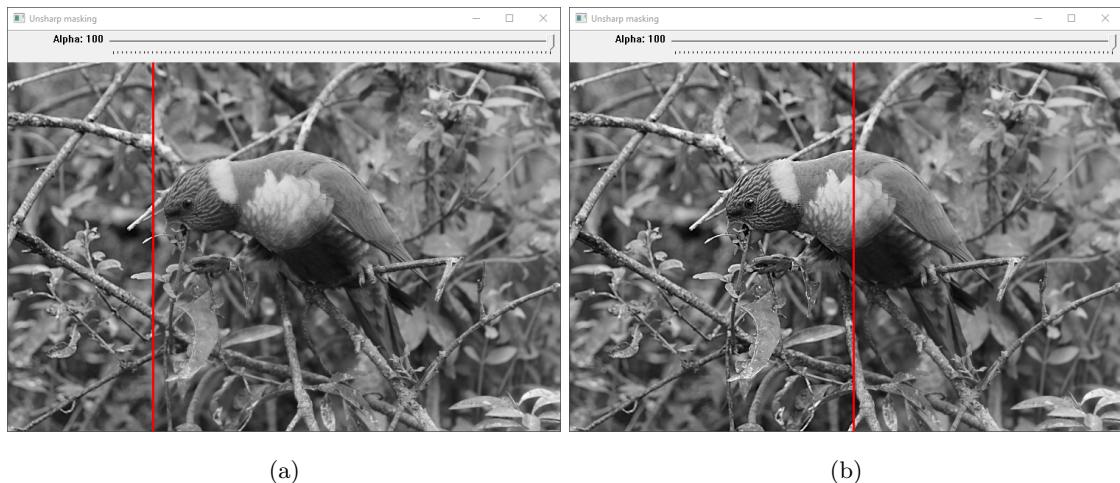


Abbildung 14.1: Darstellung. Der Bildbereich links der roten Linie ist geschärft, rechts der Linie hingegen unverändert.

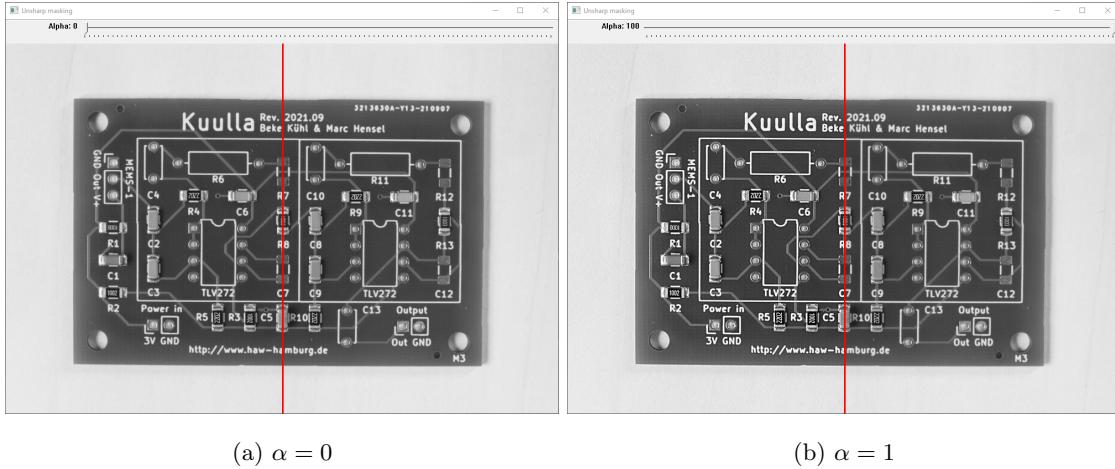


Abbildung 14.2: Schärfegrad. (a) Keine Veränderung ($\alpha = 0$): Der Bildeindruck ist links und rechts der Trennlinie identisch. (b) Maximale Erhöhung ($\alpha = 1$): Links der Trennlinie erscheint das Bild erheblich schärfer.

$g(x, y)$ mit einem Hochpass-Filter h_{HP} . Zur Verstärkung dieser Anteile addieren wir das Hochpass-gefilterte Bild $g_{HP}(x, y)$ mit einem Faktor $\alpha \in [0, 1] \subset \mathbb{R}$ zum Ausgangsbild:

$$g_{UM}(x, y) = g(x, y) + \alpha \cdot g_{HP}(x, y) \quad (14.1)$$

Für die praktische Umsetzung wird Gleichung 14.1 in der Regel etwas abgewandelt. Anstelle des Hochpass-Filters h_{HP} erzeugt man ein Tiefpass-gefiltertes Bild $g_{LP}(x, y)$. Zieht man dieses vom Ausgangsbild ab, verbleiben die hohen Frequenzen:

$$g_{HP}(x, y) = g(x, y) - \underbrace{h_{LP} * g(x, y)}_{g_{LP}(x, y)} \quad (14.2)$$

Einsetzen in Gleichung 14.1 führt schließlich zu der mathematischen Formulierung der unscharfen Maskierung, die Sie im Rahmen dieses Praktikums umsetzen sollen:

$$g_{UM}(x, y) = (1 + \alpha) \cdot g(x, y) - \alpha \cdot (h_{LP} * g(x, y)) \quad (14.3)$$

14.3 Interaktive Benutzeroberflächen

Wie eingangs erläutert, ist die Benutzeroberfläche mit einem Schieberegler sowie der Reaktion auf die Maus interaktiv zu gestalten. Daher betrachten wir zunächst, wie diese Elemente prinzipiell hinzugefügt werden.

Schieberegler Einem Fenster können beliebig Schieberegler, die in OpenCV als *Trackbar* bezeichnet werden, hinzugefügt werden. Hierzu ist eine Callback-Methode zu implementieren, die ausgeführt wird, wenn Anwender die Position des Reglers verändern. Zudem müssen die Trackbar sowie die zugehörige Methode dem Fenster bekannt gemacht werden.

Folgendes Beispiel enthält zunächst das Grundgerüst einer Callback-Methode, der beim Aufruf als zweites Argument ein Bild übergeben wird:

```
void onTrackbar(int trackbarPosition, void* imagePtr) {
    cv::Mat image = *(static_cast<cv::Mat*>(imagePtr));

    // Do something ...
}
```

Da der zweite Parameter vom Typ `void*` ist, wird der übergebene Zeiger zunächst auf den Datentyp `Mat*` konvertiert. Anschließend erfolgt die Dereferenzierung und Zuweisung zu einem Objekt der Klasse `Mat`.

Schauen wir uns nun eine passende Registrierung der Trackbar an:

```
cv::createTrackbar("Name", "Window name", NULL,
    maxValue, onTrackbar, reinterpret_cast<void*>(&image));
```

Die erste Zeichenkette legt den Namen der Trackbar fest. Dieser erscheint wie angegeben in der grafischen Oberfläche. Zudem lässt sich über diesen Namen mittels Methoden wie beispielsweise `getTrackbarPos()` und `setTrackbarPos()` auf die Trackbar zugreifen. Die zweite Zeichenkette entspricht dem Namen des Fensters, zu dem die Trackbar hinzugefügt werden soll.

Maus-Ereignisse Analog zu Schiebereglern ist auch bei Maus-Ereignissen eine Callback-Methode zu definieren und mit dem Fenster zu verknüpfen. Schauen wir uns zunächst das Beispiel einer Callback-Methode an:

```
void onMouse(int event, int x, int y, int flags, void* imagePtr) {
    // Do something ...
}
```

Über die Parameter `event` und `flags` kann gezielt auf bestimmte Ereignisse reagiert werden (z. B. linke Maustaste gedrückt). Die Übergabe eines `Mat`-Objektes erfolgt vollkommen analog zu Callback-Methoden für Schiebereglern.

Mit dem bislang Besprochenen sollte die passende Registrierung der Callback-Methode für ein spezielles Fenster selbsterklärend sein:

```
cv::setMouseCallback("Window name", onMouse, reinterpret_cast<void*>(&image));
```

14.4 Aufgabenstellung

Nachfolgend ist die Aufgabenstellung konkretisiert. Es bietet sich an, die Teilaufgaben in der angegebenen Reihenfolge umzusetzen. Beachten Sie zudem den Abschnitt *Tipps und Hilfen*.

Unscharfe Maskierung

1. Setzen Sie die unscharfe Maskierung nach Gleichung 14.3 um.
2. Verwenden Sie einen 9×9 -Binomialfilter h_{LP} .

Grafische Oberfläche

3. Stellen Sie zunächst nur das geschärfte Bild g_{UM} in einem Fenster dar.
4. Fügen Sie einen Schieberegler, über den der Parameter α verändert werden kann, zum Fenster hinzu. Bei einer Änderung wird das Ausgangsbild mit dem neuen Wert für α prozessiert und das Ergebnis im Fenster dargestellt.
5. Reagieren Sie in einer Callback-Methode auf Maus-Ereignisse des Fensters. Zeichnen Sie in einem ersten Schritt lediglich eine weiße vertikale Linie an der x -Position der Maus, um zu überprüfen, ob korrekt auf die Ereignisse reagiert wird.
6. Vervollständigen Sie die Callback-Methode, indem Sie ein Bild erzeugen, das links der zuletzt mit der Maus gesetzten x -Position per unscharfer Maskierung geschärft und rechts von dieser Position unverändert ist.

Folgende Aspekte sind *optional* und müssen daher nicht umgesetzt werden:

- (7.) Die Trennlinie wird nur verschoben, wenn die Maus bei gedrückter linker Taste bewegt wird.
- (8.) Stellen Sie die Trennlinie farbig dar.
- (9.) Verzichten Sie auf globale Variablen.

Tipps und Hilfen

- Ich verwende in meiner Implementierung nachfolgende Methoden-Prototypen:

```
/* Prototypes */
void onTrackbarAlpha(int alphaPercent, void* imagePtr);
void onMouseSplitScreen(int event, int x, int y, int flags, void* imagePtr);
void processAndDisplay(cv::Mat image, double alphaPercent, int splitX = -1);
void unsharpMasking(const cv::Mat& source, cv::Mat& processed, double alpha);
```

- Das Binomialfilter ist separierbar, sodass lediglich nachfolgender Kern erzeugt werden braucht:

$$h_{LP}^{(1)} = \frac{1}{256} \cdot (1 \ 8 \ 28 \ 56 \ 70 \ 56 \ 28 \ 8 \ 1) \quad (14.4)$$

- Eine Überprüfung, ob die Maus bei gedrückter linker Taste bewegt wird, ist durch Verwendung von *event* und *flags* möglich.
 - Ein Teilbereich eines Bildes kann über *Regions of interest (ROI)* sowie *copyTo()* in einen gleichgroßen Bereich eines anderen Bildes kopiert werden. Zur Erzeugung einer ROI, die aus zusammenhängenden Bildspalten besteht, existiert die Methode *colRange()*:
- ```
cv::Mat roiImage = image.colRange(x0, x1);
```
- Zur Darstellung einer farbigen Trennlinie erzeugen Sie zunächst das „geteilte“ Bild als Graustufenbild und konvertieren es anschließend über *cvtColor()* in ein RGB-Farbbild.

## Kapitel 15

# Praktikum 2: Detektion von Kanten und Geraden

WESENTLICHES Lernziel dieses Praktikums ist ein fundiertes Verständnis der Detektion von Geraden mittels Hough-Transformation aus Abschnitt 5.3. Dies beinhaltet insbesondere:

- ▶ Transformation von Kantenpunkten in den diskreten Parameterraum
- ▶ Rücktransformation eines Ortes des diskreten Parameterraums in eine Gerade
- ▶ Korrekte Zuordnung zwischen Kantenzygen und Häufungspunkten im Parameterraum

## 15.1 Übersicht

Das zu erstellende Programm soll über folgende Funktionalitäten verfügen:

- ▶ Zum jeweiligen Ausgangsbild werden ein binäres Kantenbild sowie der zugehörige Parameterraum der Hough-Transformation dargestellt. Der Schwellwert  $\tau$  zur Erzeugung des Kantenbildes lässt sich über einen Schieberegler verändern (Abb. 15.1).
- ▶ Über die Maus lassen sich Punkte im Parameterraum auswählen. Die Punkte werden im Parameterraum markiert und die zugehörigen Geraden im Ausgangsbild eingezeichnet (Abb. 15.2).

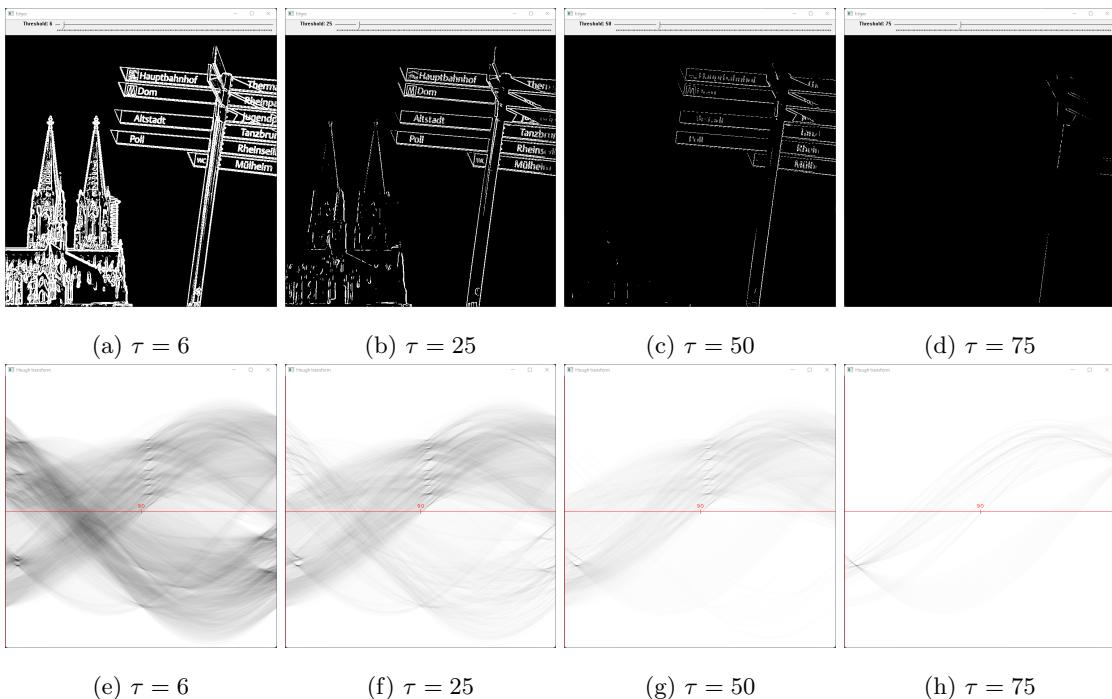


Abbildung 15.1: Mittels Sobel-Operator und Schwellwert  $\tau$  erzeugte binäre Kantenbilder sowie korrespondierende Parameterräume

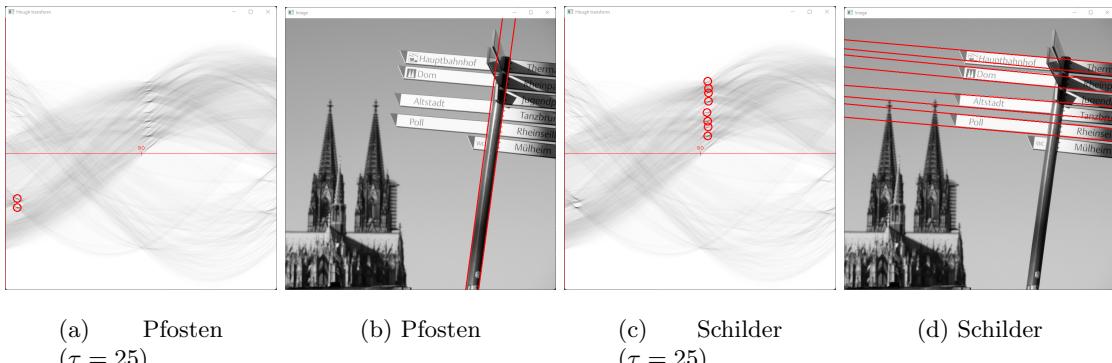


Abbildung 15.2: Im Parameterraum ausgewählte Punkte und zugehörige Geraden

## 15.2 Aufgabenstellung

Nachfolgend ist die Aufgabenstellung konkretisiert. Es bietet sich an, die Teilaufgaben in der angegebenen Reihenfolge umzusetzen. Beachten Sie zudem den Abschnitt *Tipps und Hilfen*.

### Kantenbild

1. Erzeugen Sie ein Graustufenbild der Kantenstärken unter Verwendung eines Verfahrens Ihrer Wahl.
2. Erzeugen Sie aus den Kantenstärken mittels der Methode `threshold()` ein binäres Kantenbild, in dem Kantenpixel den Wert 255 und alle übrigen Pixel den Wert 0 besitzen. Stellen Sie das Kantenbild in einem Fenster dar.

### Hough-Transformation

3. Implementieren Sie die Hough-Transformation gemäß Abschnitt 5.3 ab Seite 104 und stellen Sie den Parameterraum in einem Fenster dar.
4. *Optional:* Verwenden Sie bei der Umsetzung soweit möglich Lookup-Tabellen zur Reduktion der Laufzeit.
5. *Optional:* Zeichnen Sie die Achsen sowie die Markierung des Wertes  $\theta = \frac{\pi}{2}$  in das Bild des Parameterraumes ein.

### Interaktive Benutzeroberfläche

6. Fügen Sie einen Schieberegler hinzu, über den sich der Schwellwert  $\tau$  zum Binarisieren des Kantenbildes einstellen lässt. Alle relevanten Bilder werden entsprechend aktualisiert.
7. Bei der Auswahl eines Punktes im Parameterraum mit der Maus wird der entsprechende Ort markiert sowie die korrespondierende Gerade in das Ausgangsbild eingezeichnet.

**Tipps und Hilfen** Zur Erhöhung der Übersichtlichkeit, habe ich in meiner Implementierung Methoden, die im Zusammenhang mit der Hough-Transformation stehen, in eine Datei `HoughLine.cpp` ausgelagert. Listing 15.1 zeigt den zugehörigen Header mit den deklarierten Prototypen.

```
#ifndef IP_HOUGH_LINE_H
#define IP_HOUGH_LINE_H

/* Include files */
#include <opencv2/core/core.hpp>

namespace ip
{
 /* Prototypes */
 void houghTransform(const cv::Mat& edgeImage, cv::Mat& houghSpace, int height =
 721, int width = 720);
 void houghSpaceToLine(cv::Size imgSize, cv::Size houghSize, int x, int y, double&
 r, double& theta);
 void drawLine(cv::Mat& image, double r, double theta);
 void drawHoughLineLabels(cv::Mat& houghSpace);
}

#endif /* IP_HOUGH_LINE_H */
```

Listing 15.1: Beispielheader *HoughLine.h*



## Kapitel 16

# Praktikum 3: Bildanalyse

ANDERS als bei den ersten beiden Praktika ist der Lösungsansatz dieses Laborversuches nicht vorgegeben. Ein wesentliches Lernziel besteht darin, eigene Ideen zur Lösung der nachfolgenden Aufgabenstellung zu entwickeln, umzusetzen, aufgrund der beobachteten Ergebnisse zu modifizieren – und gegebenenfalls auch wieder zu verwerfen, falls sich der eingeschlagene Weg als nicht zielführend erweist.

### 16.1 Aufgabenstellung

Nachfolgend ist die Aufgabenstellung konkretisiert:

1. Entwickeln Sie ein Programm, das die Anzahl der Augen eines in einem Foto abgebildeten Würfels bestimmt und ausgibt.
2. Überprüfen Sie Ihr Programm für den Fall, dass mehrere Würfel abgebildet sind, und passen Sie es gegebenenfalls an.
3. Erstellen Sie ein zweites Programm, das die von Ihnen entwickelte Methode auf die Live-Bilder einer Kamera (z. B. Laptop-Kamera) anwendet.

### 16.2 Beispieldausgabe

Abbildung 16.1 enthält eine Auswahl der zur Verfügung gestellten Bilder inklusive der in einer Beispillösung erkannten Augenzahlen.

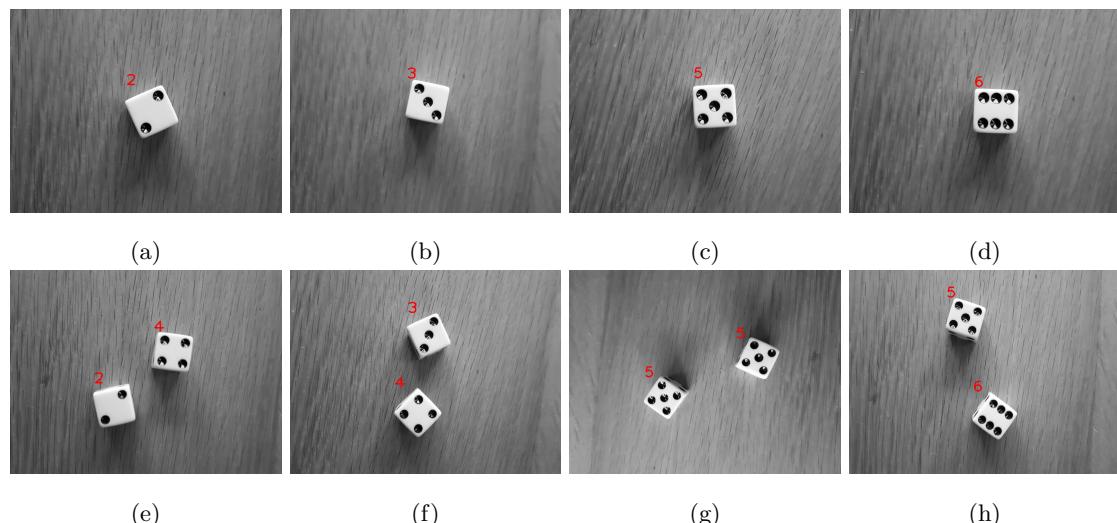


Abbildung 16.1: Würfel mit erkannten Augenzahlen

