Grafik mit OpenCV

Die Open Computer Vision Library wurde als freies Open-Source-Projekt von Intel entwickelt. Sie stellt ein breites Spektrum von Bildverarbeitungsalgorithmen zur Verfügung. *von Anja Austermann*

Vom Auslesen von Videos und Ansteuern einer Webcam über DirectShow über Low-Level-Routinen, Sobel-Filter und Canny-Kantendetektor, bis hin zur Kamera-Kalibrierung sowie vollständigen Gesichtserkennungs- und Tracking-Funktionen sind Implementierungen vieler aktueller Algorithmen in der OpenCV-Bibliothek zu finden und können kostenlos in eigene Programme eingebunden werden.

Darüber hinaus lassen sich mit OpenCV mit wenigen Zeilen Programmcode einfache Benutzeroberflächen erstellen, um Videos oder Bilder anzuzeigen.

Die neueste Version der OpenCV-Bibliothek befindet sich immer unter *http://sourceforge.net/projects/opencvlibrary/*. Sie liegt sowohl in einer Windows- als auch in einer Linux-Version vor.

In diesem Beitrag geht es um die Integration der Windows-Bibliothek in Microsoft Visual C++ 2005, zusätzlich zu den Quelltexten befindet sich auf der beiliegenden DVD auch das entsprechende Visual-Studio-Projekt. Natürlich kann man OpenCV aber auch in eine beliebige Entwicklungsumgebung unter Windows oder Linux integrieren.

Nachdem die Installationsdatei aufgerufen und die OpenCV-Bibliothek installiert ist, müssen zunächst die Pfade für die Bibliothek selbst sowie für die Include- und Sourcecode-Dateien der Entwicklungsumgebung bekannt gemacht werden.

Bei Visual Studio 2005 werden die folgenden Pfade unter *Extras|Optionen|Projekte* und *Projektverzeichnisse|VC++-Verzeichnisse* eingetragen:

Unter *Include-Dateien*:

```
<Pfad zur Opencv>\cv\include
<Pfad zur Opencv>\cvaux\include
<Pfad zur Opencv>\cxcore\include
<Pfad zur Opencv>\otherlibs\highgui
<Pfad zur Opencv>\otherlibs\cvcam\include
```

Unter *Bibliotheksdateien*:

```
<Pfad zur Opencv>\lib
```

Und unter *Quelldateien* sind diese Pfadangaben zu tätigen:

```
<Pfad zur Opencv>\cv\src
<Pfad zur Opencv>\cvaux\src
<Pfad zur Opencv>\cxcore\src
<Pfad zur Opencv>\otherlibs\highgui
<Pfad zur Opencv>\otherlibs\cvcam\src\windows
```

Die Pfade müssen nur einmal angegeben werden und bleiben auch beim Wechsel des Projekts erhalten. Für jedes einzelne neue OpenCV-Programm müssen zusätzlich in den Projekt-Eigenschaften unter *Linker|zusätzliche Abhängigkeiten* die entsprechenden lib-Dateien für alle vom Entwickler verwendeten OpenCV-Includes eingebunden werden. Die vollständige Liste der lib-Dateien für die Nutzung aller OpenCV-Funktionen in einem Programm ist: *cxcore.lib*, *cv.lib*, *highgui.lib*, *cvaux.lib* und *cvcam.lib*.

Die OpenCV ist in fünf verschiedene Bereiche unterteilt, die jeweils eine Headerdatei besitzen, die mit *#include* in eigene Programme eingebunden wird.

In der *cxcore.h* sind grundlegende Datenstrukturen und Funktionen für die Arbeit mit der OpenCV deklariert, wie zum Beispiel die Datenstruktur *IplImage*, die Bilder in OpenCV darstellt, diverse mathematische Funktionen sowie Zeichenfunktionen.

Die Datei *cv.h* deklariert Funktionen und Datenstrukturen, die im Bereich Computer Vision benutzt werden, angefangen bei diversen Filterfunktionen über Mustererkennung bis hin zu vollständigen Tracking-Algorithmen.

Benutzerinterfaces werden durch Funktionen aus der Datei *highgui.h* erzeugt. Auch das Laden und Speichern von Bildern und Videos und das einfache Ansteuern von Webkameras geschieht damit.

Experimentelle Funktionalitäten, unter anderem für die Bereiche 3D-Bildverarbeitung und fortgeschrittene Objekterkennung, findet man in der Datei *cvaux.h*.

Zusätzliche Funktionen zum Ansteuern einer Webkamera und zum synchronen Ansteuern mehrerer Kameras sind in der *cvcam.h* enthalten.

Anzeigen von Bildern und Videos

Einige Grundkonzepte der Programmierung mit der OpenCV-Bibliothek lassen sich bereits an einem einfachen Beispiel zeigen. Das folgende Programm lädt ein Bild aus einer Datei und stellt es auf dem Bildschirm dar:

```
#include "stdafx.h"
#include "cxcore.h"
#include "highgui.h"

int _tmain(int argc, _TCHAR* argv[]) {
  IplImage* img = cvLoadImage("c:\\download\\Testbild.jpg");
  for (int i = 0; i < img->width; i+=5) {
    for (int j = 0; j < img->height; j+=5) {
      drawPointColor(img, i, j, 255, 255, 255);
    }
  }
  cvNamedWindow( "Testbild", 0);
  cvShowImage("Testbild", img );
  cvWaitKey(0);
```

```
  cvReleaseImage(&img);
  cvDestroyWindow("Testbild");
  return 0;
}
```

In diesem Beispiel müssen die OpenCV-Includes *cxcore.h* und *highgui.h* eingebunden sein. Eine der grundlegende Bildbearbeitungsfunktionen in *cxcore.h* sowie die hierfür notwendigen Datenstrukturen ist *IplImage*. Die in der *highgui.h* definierten Funktionen realisieren den Dateizugriff sowie die Bildschirmdarstellung.
Eine Schlüsselrolle in der Programmierung mit OpenCV hat die Struktur *IplImage* inne. Sie repräsentiert ein Pixelbild, beispielsweise ein Bild aus einer Bilddatei oder einen Frame aus einem Video. Typischerweise bekommen alle Funktionen, die auf ein *IplImage* zugreifen, dieses als Zeiger übergeben. Die wichtigsten Bestandteile einer *IplImage*-Struktur zeigt Tabelle 1.
Alle OpenCV-Funktionen beginnen mit dem Kürzel *cv*, darauf folgt der eigentliche Name der Funktion. Alle OpenCV-Datenstrukturen beginnen mit dem Kürzel *Cv*, die zugehörigen Konstruktorfunktionen wiederum mit dem Kürzel *cv*.

Bild anzeigen

Die Funktion *cvLoadImage* lädt ein Bild aus einer Datei in eine Datenstruktur vom Typ *IplImage*. Sie bekommt den Pfad zur Bilddatei übergeben und gibt einen Zeiger auf ein *IplImage* zurück, in dem nach erfolgreichem Laden der Datei das Bild enthalten ist. Die Datenformate JPEG, BMP, DIB, PNG, PBM, PGM, PPM, SR, RAS und TIFF können von *cvLoadImage* direkt gelesen werden.
Um ein Bild in einem Fenster anzuzeigen, wird nun mit der *highgui*-Funktion *cvNamedWindow* ein Fenster auf dem Bildschirm erzeugt. *cvNamedWindow* bekommt zwei Parameter übergeben: den Namen als Array vom Typ *char* und eine Konstante, die das Verhalten des Fensters bestimmt. Übergibt man 0, paßt sich das Fenster nicht der Bildgröße an und das Bild wird entsprechend skaliert. Die Fenstergröße kann dann vom Benutzer verändert werden. Wird 1 oder *CV_WINDOW_AUTOSIZE* übergeben, paßt sich das Fenster beim Anzeigen eines Bilds automatisch an die Größe des Bildes an. Die Fenstergröße kann vom Benutzer nachträglich nicht mehr verändert werden.
Der Name des Fensters ist insofern wichtig, weil Funktionen, die auf das Fenster zugreifen möchten, dies ausschließlich über den Namen des Fensters tun. Das bedeutet vor allem, daß jeder Fenstername eindeutig sein sollte und nicht mehrfach vergeben werden darf.

Die Funktion *cvShowImage*, die ein Bild in einem Fenster anzeigt, bekommt also als ersten Parameter den Namen des Fensters als *char** übergeben, der exakt genauso geschrieben sein muß wie beim Aufruf von *cvNamedWindow*. Der zweite Parameter für *cvShowImage* ist der Zeiger auf ein *IplImage*, das das anzuzeigende Bild enthält. Ist er nicht *NULL*, wird beim Aufruf des Programms das aus der Datei geladene Bild in einem OpenCV-Fenster angezeigt.
Wichtig ist, daß nicht mehr benötigter Speicherplatz wieder freigegeben und das Anzeigefenster sauber wieder entfernt wird. Hierfür stellt die OpenCV eigene Funktionen zur Verfügung: *cvReleaseImage* gibt den von einem *IplImage* belegten Speicherplatz wieder frei. Dafür bekommt es einen Zeiger auf den Zeiger zum *IplImage* übergeben. *cvDestroyFrame* entfernt ein Anzeigefenster, hierfür wird der Name des Fensters als Parameter übergeben.
Das Speichern eines Bilds funktioniert analog zum Laden mit der Funktion *cvSaveImage*. Sie bekommt den Dateinamen als Array vom Typ *char* übergeben, sowie einen Zeiger auf das zu speichernde Bild:

```
cvSaveImage("testbild.jpg", img);
```

Mit der OpenCV lassen sich aber nicht nur feststehende Bilder, sondern auch Videos und Kamerabilder verarbeiten.

Videos

Da die Bibliothek jeden einzelnen Videoframe als *IplImage* weiterverarbeitet, funktioniert die Arbeit mit Videos im Prinzip genauso wie die Arbeit mit Einzelbildern, sobald das Auslesen aus einer Datei oder das Capturing von einer Kamera abgeschlossen ist. Das folgende Programm lädt ein Video aus einer Datei und zeigt alle Frames des Videos nacheinander in einem Fenster an.

```
#include "stdafx.h"
#include "highgui.h"

int _tmain(int argc, _TCHAR* argv[]) {
CvCapture *avi = cvCaptureFromAVI("Testvideo.avi");
  cvNamedWindow( "Testbild", 0);
  while (true) {
    IplImage* img = cvQueryFrame(avi);
    if (!img) {
      break;
    }
```

Funktion	Wirkung
imageData	Die eigentlichen Bilddaten als Array vom Typ *char*.
width, height	Breite und Höhe des Bildes in Pixeln als *int*.
widthStep	Länge einer Bildzeile in Byte.
nChannels	Anzahl Farbkanäle. 1 für Monochrom, 3 für Farbe und 4 für Farbe mit Alphawert.
depth	Farbtiefe in Bit, definiert durch die Konstanten *IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16U, IPL_DEPTH_16S, IPL_DEPTH_32S, IPL_DEPTH_32F* und *IPL_DEPTH_64F*.

Tabelle 1: Bestandteile einer IplImage-Struktur

```
    cvShowImage("Testbild", img);
    cvWaitKey(1);
  }
  cvReleaseCapture(&avi);
  cvDestroyWindow("Testbild");
  return 0;
}
```

OpenCV kann alle AVI-Dateien einlesen, für die auf dem System ein passendes Codec installiert ist. Die Funktion *cvCaptureFromAvi* bereitet ein Video zum Auslesen vor, wenn ihr der Dateiname der Videodatei als Parameter übergeben wird. Die Funktion gibt einen Zeiger auf eine *cvCapture*-Struktur zurück.
Übergibt man diesen Zeiger nun an die Funktion *cvQueryFrame*, wird der aktuelle Frame des Videos als Zeiger auf ein Bild vom Typ *IplImage* zurückgegeben, das wie gewohnt weiterverarbeitet und angezeigt werden kann. Die *cvCapture*-Struktur merkt sich jeweils den zuletzt bearbeiteten Frame, so daß beim nächsten Aufruf von *cvQueryFrame* automatisch der folgende Frame zurückgegeben wird. Wird nach dem letzten Frame erneut *cvQueryFrame* aufgerufen, meldet die Funktion den Wert *NULL*.
In einer Schleife zum Anzeigen von Bildern sollte regelmäßig die Funktion *cvWaitKey* aufgerufen werden, da der OpenCV nur so genug Zeit zum Aktualisieren der Fenster bleibt. Sie erhält als Parameter eine Wartezeit in Millisekunden.
Es wird entweder gewartet, bis die angegebene Zeit abgelaufen ist oder der Benutzer eine Taste drückt. Übergibt man *cvWaitKey* den Parameter 0, wird in jedem Fall bis zu einem Tastendruck vom Benutzer gewartet.
Nach dem Anzeigen des Videos muß der von der *CvCapture*-Struktur belegte Speicher mit *cvReleaseCapture* wieder freigegeben werden. Die manuelle Freigabe der *IplImages* für die einzelnen Frames sollte hingegen unbedingt vermieden werden, da das zu einem Speicherzugriffsfehler führt.

Ansteuern einer Webcam

Damit das Ansteuern von Webcams in der OpenCV-Bibliothek unter Windows funktioniert, muß auf dem Computer zunächst das DirectShow-SDK installiert sein, das direkt von Microsoft als Bestandteil des Microsoft Platform SDK heruntergeladen werden kann. Mit dem Befehl *nmake* müssen die im Unterverzeichnis *Samples\Multimedia\DirectShow\BaseClasses* der Microsoft-Platform-SDK-Installation vorhandenen Quelltextdateien zur Bibliothek *strmbase.lib* kompiliert werden. Sie ist für die Ansteuerung von Webcams mit der OpenCV-Bibliothek erforderlich.
Beim Visual Studio 2005 ist es auf jeden Fall ratsam, die neueste Version des Microsoft Platform SDK herunterzuladen, da der Quellcode der benötigten Dateien in vorherigen Versionen zu Fehlern beim Kompilieren führen kann. Der Pfad zur Datei *strmbase.lib* muß nach ihrer Fertigstellung wie die OpenCV-Bibliotheksdateien in den Visual-Studio-Optionen als Bibliotheksdatei eingetragen werden.
Das Einbinden der Kamera in eigene Programme gestaltet sich nach einer korrekten Installation der DirectShow-Dateien genauso einfach wie das Auslesen von Daten aus einer Videodatei: statt *cvCaptureFromAVI* gilt nun an der gleichen Stelle *cvCaptureFromCAM*. Alle anderen Programmteile können unverändert bleiben:

```
CvCapture *avi = cvCaptureFromCAM(-1);
```

CvCaptureFromCAM bekommt als Parameter die Nummer der Kamera übergeben. Derzeit treten manchmal noch Probleme beim Ansteuern von mehr als einer Kamera auf, die hoffentlich in einer der zukünftigen Versionen der Bibliothek behoben sind. Wird der Wert -1 übergeben, wird, falls im System mehr als eine Kamera vorhanden ist, ein Dialogfenster zur Kameraauswahl angezeigt.

Videos speichern

Um Videos, die beispielsweise mit einer Webcam aufgenommen wurden, auf der Festplatte zu speichern beziehungsweise um beliebige Sequenzen von *IplImages* als Video auf der Festplatte abzulegen, arbeitet die OpenCV-Bibliothek mit der Datenstruktur *CvVideoWriter*. Ein *CvVideoWriter* wird mit der Funktion *cvCreateVideoWriter* gebildet.
Ihr wird zunächst der Dateiname der Ausgabedatei als *char** übergeben. Als nächster Parameter folgt das Videoformat. Verschiedene Videoformate können mit dem 4-Zeichen-Code des entsprechenden Videocodecs angegeben werden, für Motion JPEG also zum Beispiel *CV_FOURCC('M','J','P','G')*. Außerdem wird unter Windows beim Wert -1 ein Dialog angezeigt, aus dem der Benutzer das gewünschte Videoformat auswählen kann. Der nächste Parameter ist vom Typ *float* und gibt die Framerate an. Gängige Werte für das Speichern von Webcam-Videos sind 15 oder 30 Frames pro Sekunde. Stimmt die Framerate für das Speichern nicht mit der Framerate der Videoquelle überein, wird das Video später zu schnell oder zu langsam wiedergegeben. Der letzte Parameter ist die Bildgröße. Sie muß unbedingt mit der Größe der *IplImages* übereinstimmen, die später gespeichert werden sollen.
Ein *CvVideoWriter*, der ein Video in einem vom Benutzer auswählbaren Format mit 15 Bildern pro Sekunde und einer Bildgröße von 320x240 speichert, kann wie folgt erzeugt werden:

```
CvVideoWriter *writer=cvCreateVideoWriter("video.avi",
                                          -1, 15.0,
                                          cvSize(320, 240));
```

Bildgrößen werden, wie in diesem Beispiel gezeigt, in OpenCV generell durch die Datenstruktur *CvSize* repräsentiert. Die zugehörige Konstruktorfunktion *cvSize* bekommt den x- und den y-Wert übergeben und legt diese Werte in einer *CvSize*-Struktur ab. Die Frames werden nun mit der Funktion *cvWriteFrame* gespeichert. Ihr wird als erstes der Pointer auf den vorher erzeugten CvVideoWriter übergeben, und danach ein Pointer auf das als nächstes zu speichernde IplImage:

```
cvWriteFrame(writer, image);
```

Sind alle Einzelbilder eines Videos gespeichert, muß die Videodatei abgeschlossen und der *CvVideoWriter* wieder mit *cvReleaseVideoWriter* aus dem Speicher entfernt werden. Der Funktion muß die Adresse des Zeigers, der den *CvVideoWriter* referenziert, übergeben werden:

```
cvReleaseVideoWriter(&writer);
```

Es muß unbedingt *cvReleaseVideoWriter* nach dem Abschluß des Schreibens aller Frames aufgerufen werden, da ansonsten nicht nur der Speicher für den *CvVideoWriter* belegt bleibt, sondern das Video auch nicht korrekt abgeschlossen wird und später nicht angesehen werden kann.

Pixelmanipulationen

Bis jetzt wurde beschrieben, wie man Bilder und Videos aus einer Datei oder von einer Kamera ausliest, anzeigt und speichert, die Daten wurden aber nicht verändert. In der Regel möchte man aber die Bilder nicht nur anzeigen, sondern auch selbst bearbeiten.
Dafür sind Kenntnisse darüber notwendig, wie ein Bild für den Computer aussieht.
Jeder Bildpunkt wird durch einen oder mehrere Farbwerte kodiert. Bei einem 8-Bit-Graustufenbild sind das typischerweise Werte von 0 (Schwarz) bis 255 (Weiß). Pixel eines 8-Bit-Farbbildes mit drei Farbkanälen lassen sich durch ihren Rot-, Grün- und Blau-Wert beschreiben, der jeweils mit einem Wert von 0 bis 255 angegeben wird. Bei einem RGB-kodierten Bild wäre ein Pixel, der den Wert 255, 0, 0 hat, daher rot, ein Pixel mit dem Wert 0, 255, 0 wäre grün und der Bildpunkt mit dem Wert 0, 0, 0 schwarz.
Zu beachten ist, daß die Bilddaten in der Struktur *IplImage* technisch nicht in Form einer quadratischen Matrix, sondern als eindimensionales Array vorliegen. Die Zeilen des Bilds werden von oben nach unten hintereinander in das Array geschrieben. Je nach Bilddatentyp besitzt ein Bild außerdem mehrere Farbkanäle. Um auf einen Pixel zuzugreifen, ist also etwas Rechenarbeit erforderlich.
Bei einem 8-Bit-Schwarzweiß-Bild setzt man den Farbwert für einen Bildpunkt an der Stelle x, y so:

```
void drawPointSW(IplImage* image, int x, int y, int color) {
  ((uchar*)(image->imageData +
           image->widthStep * y))[x] = color;
}
```

Der Ausdruck in der Klammer dient dazu, bis zur gewünschten Zeile vorwärts zu springen. *image->imageData* liefert hierzu zunächst den Zeiger auf die eigentlichen Bilddaten. Zu ihnen wird dann per Zeigerarithmetik die Anzahl der zu überspringenden Byte addiert. Das macht die Angabe *image->widthStep*, die die Breite einer Zeile in Byte darstellt.
Multipliziert mit dem y-Wert des Bildpunkts ergibt sich die Anzahl zu überspringender Byte. Der resultierende Pointer zeigt auf den Beginn der Zeile, in der ein Pixel verändert werden soll. Auf den zu verändernden Pixel wird nun über den Arrayindex x, der die x-Koordinate des Pixels angibt, zugegriffen.

Hat man den Pixelzugriff in einem Schwarzweiß-Bild verstanden, ist die Erweiterung auf farbige Bilder nur noch eine Kleinigkeit. In einem Drei-Kanal-Farbbild liegen jeweils die Werte für die drei Kanäle, also zum Beispiel Rot, Grün und Blau, nebeneinander. Das bedeutet, daß man, um den nächsten Pixel gleicher Farbe zu erreichen, zwei Pixel überspringen muß.
Auf jeden Farbkanal wird einzeln zugegriffen. Da *widthStep* bei der Berechnung der Byte pro Zeile die drei Farbkanäle bereits korrekt einbezieht, muß lediglich der x-Wert angepaßt werden, weil nur noch jedes dritte Pixel einer Zeile zum jeweiligen Farbkanal gehört. Bei einem BGR-Bild sähe der Zugriff beispielsweise so aus:

```
void drawPointColor(IplImage* image, int x, int y,
                    int blue, int green, int red) {
  ((uchar*)(image->imageData +
           image->widthStep * y))[x * 3] = blue;
  ((uchar*)(image->imageData +
           image->widthStep * y))[x * 3 + 1] = green;
  ((uchar*)(image->imageData +
           image->widthStep * y))[x * 3 + 2] = red;
}
```

Natürlich besitzt die OpenCV-Bibliothek auch Funktionen, mit denen direkt geometrische Figuren in *IplImages* gezeichnet werden können. Einfache geometrische Figuren, für die die OpenCV-Bibliothek fertige Zeichenfunktionen zur Verfügung stellt, sind Linien, Rechtecke, Kreise und Ellipsen sowie beliebige Polygone. Außerdem kann Text in einem Bild dargestellt werden.

Zeichnen

Die Zeichenfunktionen für diese Objekte haben alle eine ähnliche Parametersignatur. Zuerst wird ein Zeiger auf das *IplImage* übergeben, in das gezeichnet werden soll. Danach folgen Parameter, die spezifisch für das zu zeichnende Objekt sind, wie zum Beispiel der Start- und Endpunkt einer Linie oder der Mittelpunkt und der Radius eines Kreises. Die darauf folgenden Parameter sind wieder für alle Zeichenfunktionen gleich. Zunächst wird die Farbe angegeben, sie ist ein Wert vom Typ *CvScalar*.
In der OpenCV ist aber das Makro *CV_RGB* definiert, das die Rot-, Grün- und Blauwerte als Integer übergeben bekommt und einen *CvScalar* zurückgibt. Anders als beim direkten Erzeugen eines CvScalars erhält man dadurch die typische RGB-Reihenfolge. Generiert man direkt einen *CvScalar* mit der Konstruktorfunktion *cvScalar*, muß man zunächst den Blau-, dann den Grün- und zum Schluß den Rotwert angeben.
Der nächste Parameter ist die Linienstärke in Pixeln als Integer, dann die Zeichenart, wobei 4 für Linien steht, die von einem Punkt aus jeweils nach unten, oben, links oder rechts weitergeführt werden können, 8 für Linien, die außerdem diagonal weitergeführt werden können, und *CV_AA* für das Zeichnen mit Anti-Aliasing-Funktionen. Der letzte Parameter muß in der Regel nicht angegeben werden, man kann ihn auf seinem Defaultwert belassen. Dieser zusätzliche Integerwert legt fest, wie genau, also mit wie vielen »Nachkomma-Bits« die Punkte einer Linie berechnet werden sollen.

Zum Zeichnen einer Linie werden zusätzlich zu den oben beschriebenen Parametern zwei *CvPoint*-Werte übergeben, die die Endpunkte der Linie festlegen.

Grafikelemente

Ein *CvPoint* besteht aus einem x- und einem y-Wert und beschreibt einen Bildpunkt:

```
cvLine(img, cvPoint(0, 0),
           cvPoint(img->width, img->height),
           CV_RGB(255, 0, 0), 3, CV_AA);
```

Zum Zeichnen eines Rechtecks werden der Funktion *cvRectangle* zwei CvPoint-Werte für die linke obere und die rechte untere Ecke des Bilds übergeben:

```
cvRectangle(img, cvPoint(0, 0),
                 cvPoint(img->width, img->height),
                 CV_RGB(0, 0, 255), 3, CV_AA);
```

cvCircle zeichnet einen Kreis und benötigt als Parameter den Mittelpunkt als *CvPoint* sowie den Radius als Integerwert:

```
cvCircle(img, cvPoint(50, 50), 10,
              CV_RGB(255, 255, 0), 2, CV_AA);
```

Erwartungsgemäß wird eine Ellipse mit *cvEllipse* gezeichnet. Benötigt werden der Mittelpunkt als *CvPoint* und die Breite und Höhe der Ellipse als *CvSize* jeweils parallel zur x- beziehungsweise y-Achse des Bilds. Danach kann die Ellipse um einen beliebigen Winkel um ihren Mittelpunkt gedreht werden. Dieser Winkel wird als *double*-Wert angegeben.
OpenCV kann auch unvollständige Ellipsen zeichnen. Deswegen müssen bei allen Ellipsen Start- und ein Endwinkel im Format double festgelegt werden, für eine vollständige Ellipse sind es beispielsweise die Werte 0 und 360.

```
cvEllipse(img, cvPoint(50, 50),
               cvSize(10, 20), 10.0, 0.0, 360.0,
               CV_RGB(255, 0, 255), 2, 8);
```

Text wird in OpenCV in zwei Schritten in ein Bild gezeichnet. Zunächst werden die Schrifteinstellungen definiert, dann wird der Text an die gewünschte Stelle im Bild platziert.

Text

OpenCV bietet verschiedene Schriftarten für das Zeichnen an, für jede ist eine Konstante definiert. Drei typische Schriftarten sind *CV_FONT_HERSHEY_SIMPLEX*, eine normalgroße, serifenlose Schrift, *CV_FONT _HERSHEY_COMPLEX*, eine normalgroße Schrift mit Serifen und *CV_FONT_HERSHEY_SCRIPT_SIMPLEX*, eine handschrift-ähnliche Schrift
Ein Text wird in ein *IplImage* gezeichnet, indem man zuerst eine Variable vom Typ *CvFont* anlegt. Zuständig für die Initialisierung ist die Funktion *cvInitFont*, die *CvFont* als ersten Parameter übergeben bekommt. Anschließend muß ihr die Konstante übergeben werden, die den Schrifttyp bestimmt. Es folgen zwei weitere Parameter für die Skalierung der Schrift in x- und y-Richtung. Übergibt man für beide Werte 1, wird die Schrift in ihrer Normalgröße angezeigt.
Nach der Initialisierung stellt die Funktion *cvPutText* die Schrift auf dem Bildschirm dar. Sie benötigt einen Zeiger auf ein *IplImage* und den gewünschten Text als *char**. Außerdem werden die Position der linken, unteren Ecke des Textes relativ zu den Bildkoordinaten angegeben, dann die zuvor initialisierte Schriftart und außerdem noch die Schriftfarbe:

```
IplImage* img;
CvFont font;

[...]

cvInitFont(&font, CV_FONT_HERSHEY_SIMPLEX, 1, 1);
cvPutText(img, "Ganz wichtiger Text",
               cvPoint(40, 60), &font, CV_RGB(255, 0, 0));
```

Maus-Events

Damit der Benutzer durch Mausklicks in die Bild- oder Videoverarbeitung eingreifen kann, stellt OpenCV die Funktion *cvMouseCallback* zur Verfügung. Ihr werden der Name des zu überwachenden Fensters als Text, die aufzurufende Callback-Funktion und eventuelle Parameter übergeben.
Möchte man auf einen Mausklick im Fenster mit dem Namen *Testbild* mit einem Aufruf der Funktion *on_mouse* reagieren, wird der entsprechende Callback so registriert:

```
cvSetMouseCallback("Testbild", on_mouse);
```

Die aufzurufende Callback-Funktion muß die folgende Parametersignatur besitzen:

```
meinCallback(int event,
             int x,
             int y,
             int flags,
             void* param)
```

Die folgende Callback-Funktion merkt sich Mausklicks bis zu einer Maximalzahl *CORNERMAX* und zeichnet nach dem letzten Mausklick ein Polygon aus den angeklickten Punkten in das in der Variable *img* gespeicherte *IplImage** und stellt es auf dem Bildschirm dar:

```
static void on_mouse(int event, int x, int y, int flags,
                     void* param) {
  if (event == CV_EVENT_LBUTTONDOWN) {
    polygon[cornercount++] = cvPoint(x, y);
    if (cornercount == CORNERMAX) {
      cvPolyLine(img, &polygon, &cornercount, 1,
                 1, CV_RGB(255, 0, 0));
      cvFillPoly(img, &polygon, &cornercount,
                 1, CV_RGB(255, 0, 0));
      cornercount = 0;
      cvShowImage( "Testbild", img);
    }
  }
}
```

Hier wird das Polygon mit *cvPolyLine* gezeichnet und mit *cvFillPoly* anschließend gefüllt. Die beiden bisher nicht erwähnten Funktionen werden analog zu den anderen beschriebenen Zeichenfunktionen aufgerufen.

Filter

Videobilder und Fotos lassen sich auf mannigfaltige Weise im Computer weiterverarbeiten. Die Möglichkeit des Zeichnens in *IplImage*-Bilder wurde hier schon beschrieben. Interessanter ist die Anwendung mathematischer Filteroperationen auf die einzelnen Bildpunkte eines Bilds. Exemplarisch sollten deshalb einige solcher Filter und ihr Einsatz vorgestellt werden.
Ein Filter läßt sich in Form einer Matrix veranschaulichen, die zeilenweise über ein Bild geschoben wird, wobei jeweils für den Bildpunkt, der sich unterhalb des Mittelpunkts der Matrix befindet, ein neuer Wert berechnet wird, indem die Werte der Matrix jeweils mit den darunterliegenden Bildpunkten multipliziert und dann zusammenaddiert werden. Mit Filtermatrizen lassen sich Bilder weichzeichnen oder Kanten in einem Bild finden.

Weichzeichnen

Ein Filter zum Weichzeichnen ist Bestandteil jedes Bildverarbeitungsprogramms und auch in der OpenCV ist einer implementiert: *cvSmooth*.

Ein Bild wirkt weichgezeichnet, wenn Übergänge zwischen Farben fließend und nicht scharf und abrupt sind. Dieser Effekt entsteht, indem man den Farbwert eines Pixels neu berechnet und neben seinem alten Wert auch die Farben der umgebenden Pixel mit einbezieht. Das bedeutet zum Beispiel, daß sich der Farbwert eines blauen Pixels neben mehreren roten Pixeln ins Violette verschieben wird und die Farbwerte auf einer Kante zwischen einer schwarzen und weißen Fläche zu einem dunkel- bis hellgrauen Farbübergang werden.
Natürlich kann man die Pixel aus der Umgebung unterschiedlich gewichten. Eine Möglichkeit wäre, die Farbwerte aller Pixel in einer Umgebung zusammenzurechnen und den Mittelwert als neuen Farbwert zu verwenden. Häufig ist eine Gewichtung in Form einer Gaußfunktion, was dazu führt, daß Umgebungspixel, die sich in der Nähe des neu zu berechnenden Punktes befinden, stärker berücksichtigt werden als weiter entfernte Pixel. cvSmooth bekommt im einfachsten Fall ein *IplImage** als Eingabeparameter und ein zweites *IplImage** als Ausgabeparameter übergeben. Weitere mögliche Parameter sind der Weichzeichnertyp, mit dem das Bild geglättet werden soll, und bis zu drei Integerwerte. Ein typischer Filter ist zum Beispiel *CV_GAUSSIAN* für einen Gaußfilter oder *CV_BLUR*, um alle Umgebungspixel gleichmäßig zu berücksichtigen. In beiden Fällen sind zwei Integerwerte für die Breite und Höhe der Region, die vom Filter berücksichtigt werden soll, um den neuen Farbwert für einen Punkt zu berechnen. Für das Weichzeichnen mit einer 5x5-Gauß-Matrix sieht der Aufruf von *cvSmooth* beispielsweise so aus:

```
cvSmooth(smoothinput, smoothoutput, CV_GAUSSIAN, 5, 5);
```

Bild 1 zeigt die Wirkung von *cvSmooth*.
Da viele Filteroperationen in der OpenCV nur auf bestimmten Typen von Eingabedaten funktionieren, also beispielsweise nur auf 8-Bild-Graustufenbildern, wie der weiter unten beschriebene Sobelfilter, ist es oft empfehlenswert, eine Filterfunktion in mehreren Schritten zu implementieren. Hierbei wird zunächst das Ausgabebild erzeugt, in das die Filterfunktion ihre Ausgaben ablegen soll und dann das Eingabebild in das gewünschte Format konvertiert. Falls man weiß, welche Form von Eingabedaten man

Bild 1: Der Weichzeichner cvSmooth

bekommt, und wenn diese mit den benötigten Eingabedaten des Filters übereinstimmen, kann man diesen Overhead natürlich vermeiden und auf das Erzeugen eines temporären Eingabebilds und die Konvertierung des Eingabebildes verzichten.

Ein Bild wird in zwei Schritten in ein anderes Format konvertiert: Zunächst erzeugt man ein Bild im Zielformat. Im folgenden Beispiel entsteht mit *cvCreateImage* ein Bild mit 8-Bit-Farbtiefe und drei Farbkanälen. Dann wird die Funktion *cvConvertImage* aufgerufen. Ihr wird zunächst das Ein- und dann das Ausgabebild übergeben. *cvConvertImage* erkennt die Typen beider Bilder und konvertiert das Format entsprechend. Nach dem Anwenden des Filters wird der von dem temporären Eingabebild belegte Speicher wieder freigegeben und das Ausgabebild zurückgeliefert:

```
IplImage* filterSmooth(IplImage* inputImg) {
  IplImage* smoothoutput=cvCreateImage(cvGetSize(inputImg),
                                       8, 3);
  IplImage* smoothinput = cvCreateImage(cvGetSize(inputImg),
                                        8, 3);
  cvConvertImage(inputImg, smoothinput);
  cvSmooth(smoothinput, smoothoutput, CV_GAUSSIAN, 5, 5);
  cvReleaseImage(&smoothinput);
  return smoothoutput;
}
```

Zwei häufig benötigte Filterfunktionen, wenn es darum geht, Bilder für eine Computer-Vision-Anwendung vorzubereiten, sind Erode und Dilate. Mit Erode werden dunkle Bereiche des Bilds ausgedehnt, indem jeder Bildpunkt durch den dunkelsten Punkt in seiner unmittelbaren Umgebung ersetzt wird, Dilate ist das Gegenstück. Hier werden helle Bereiche des Bilds ausgedehnt, indem jedes Pixel durch den hellsten Bildpunkt seiner unmittelbaren Umgebung ersetzt wird.

Erode und Dilate

Ein Anwendungsbereich für dieses Filter-Duo ist die Entfernung von »Schnee« in Bildern, also isolierten dunklen oder hellen Punkten auf ansonsten gleichfarbigen Flächen. Es wird zunächst Erode und dann Dilate angewandt, um dunkle Störpixel zu entfernen. Mit Dilate werden helle Bereiche ausgedehnt und einzelne, dunkle Pixel verschwinden. Um das Ausdehnen der hellen Flächen wieder rückgängig zu machen, ruft man anschließend Erode auf. Große, helle Flächen schrumpfen wieder auf ihr Normalmaß zurück, die einzelnen von Dilate gelöschten, dunklen Störpixel bleiben unsichtbar. Es bleibt allerdings bei Farb- und Graustufenbildern in der Regel eine etwas unnatürliche Veränderung des Bildes an Kanten und Farbübergängen zurück. Analog ruft man zunächst Erode und danach Dilate auf, um störende, helle Pixel zu entfernen.

Der Erode-Filter wird in der OpenCV durch die Funktion *cvErode* realisiert. Sie bekommt je ein *IplImage** für die Ein- und Ausgabe übergeben:

```
cvErode(erodeinput, erodeoutput)
```

Als optionaler Parameter kann ein *IplConvKernel** für die Form der Region, über die der Erode-Filter angewendet werden soll, angegeben werden. Der Defaultwert – wurde nichts oder *NULL* angegeben – ist 3x3 Pixel. Das heißt, daß ein Pixel durch die Farbe des dunkelsten direkt angrenzenden Pixels ersetzt wird. Außerdem kann die Anzahl der Durchläufe des Erode-Filters als Integer übergeben werden. Der Dilate-Filter ist in der Funktion *cvDilate* realisiert. Die Parameter sind identisch:

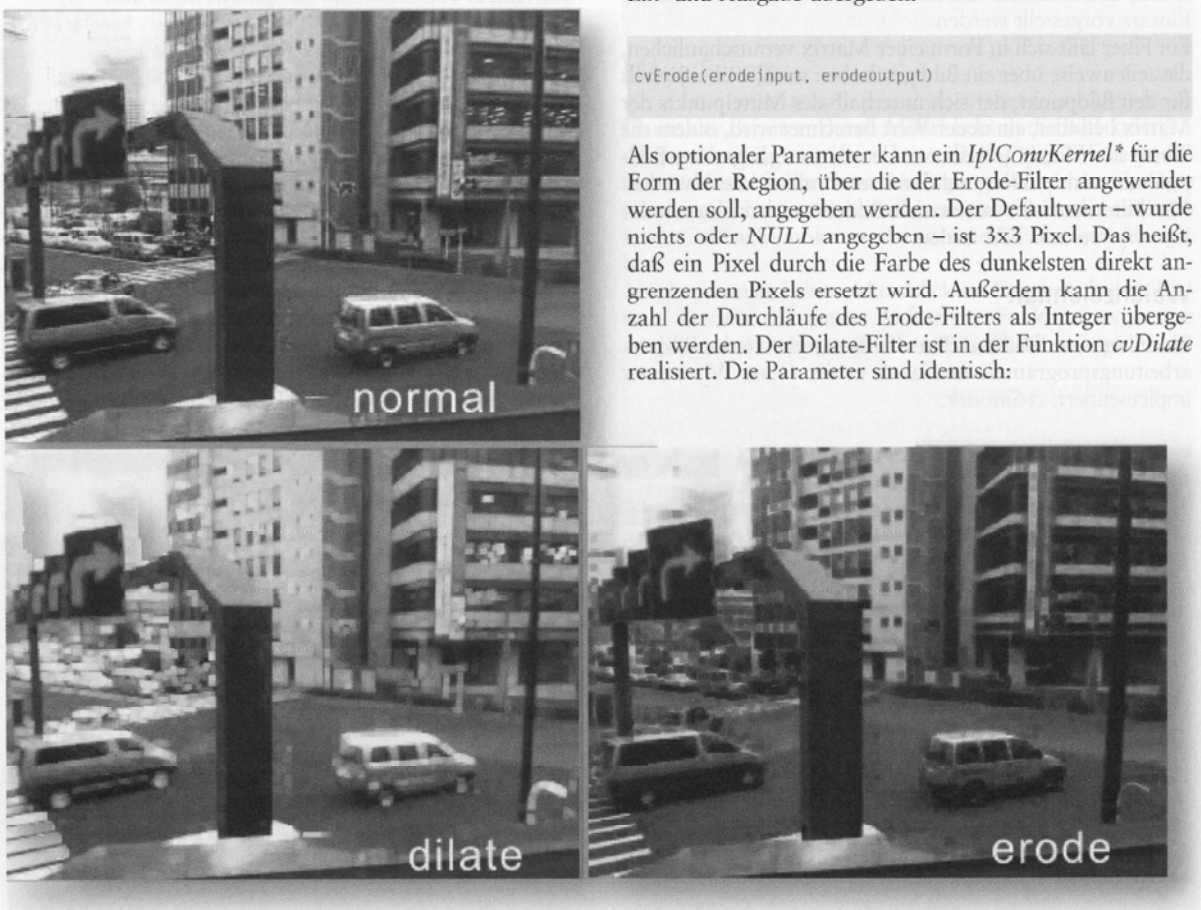

Bild 2: Die Funktionen cvDilate und cvErode

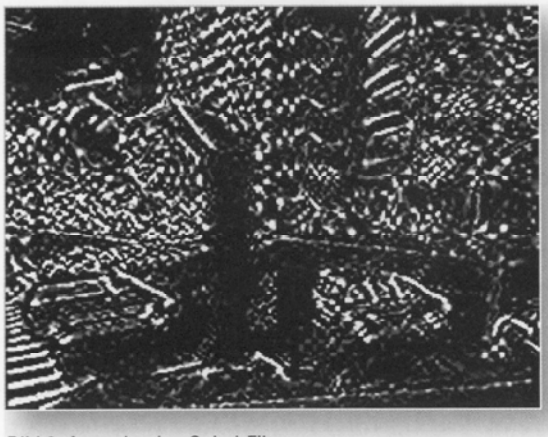

Bild 3: Ausgabe des Sobel-Filters

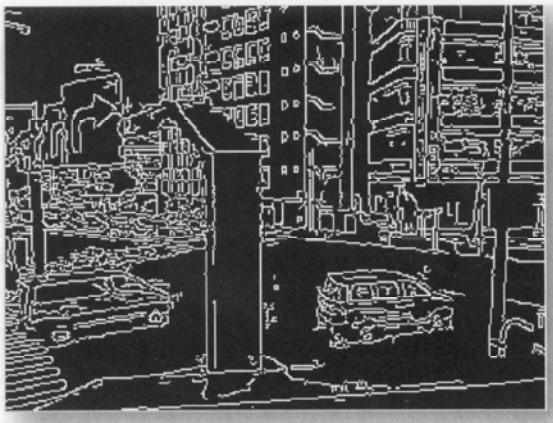

Bild 4: Das Ergebnis des Canny Edge Detectors

```
cvDilate(dilateinput, dilateoutput);
```

Bild 2 verdeutlicht die Wirkung von *cvDilate* und *cvErode*. Vor allem beim Vergleichen der weißen Zebrastreifen und der Pfeile der Straßenschilder sowie an den Fenstern des Hauses im Hintergrund erkennt man die Auswirkung der beiden Filterfunktionen.

Da die OpenCV-Bibliothek in erster Linie auf Anwendungen im Bereich Computer Vision ausgelegt ist, besitzt sie eine Vielzahl von Funktionen, mit denen ein Bild für den Computer leichter verständlich gemacht werden soll. Eine davon ist die Kantenextraktion, die häufig als Vorverarbeitungsschritt für die Erkennung von Objekten in Bildern oder Videos dient, um die Komplexität eines Bilds zu reduzieren.

In OpenCV sind dafür ein Sobel-Filter sowie der darauf aufbauende Canny-Edge-Detektor implementiert.

Die Grundidee des Sobel-Filters ist einfach: Eine Kante ist ein mehr oder weniger schneller Übergang von Hell nach Dunkel oder umgekehrt. Außerdem läßt sich die Richtung einer Kante durch einen horizontalen und einen vertikalen Anteil darstellen. Ein Sobel-Filter besteht daher aus zwei Teilen: je einem vertikalen und einem horizontalen Kantendetektor. Mathematisch wird hierfür jeweils die diskrete Ableitung der Bildfunktion in x- und y-Richtung benutzt. Je abrupter der Farbübergang in Arbeitsrichtung des Kantendetektors ist, desto größer der Rückgabewert. Addiert man die absoluten Werte für beide Richtungen, erhält man große Werte dort, wo Kanten, also abrupte Farbübergänge vorliegen, und kleine Werte an Stellen mit homogener Farbgebung. Dessen Darstellung als Graustufenbild ergibt einen dunklen Hintergrund mit hellen Kanten, wie in Bild 3 zu sehen ist.

Der Sobel-Filter ist in der Funktion *cvSobel* realisiert. Als Parameter bekommt er jeweils einen Zeiger auf ein *IplImage* für die Ein- und Ausgabe übergeben. Danach folgen zwei Parameter für die Ordnung der Ableitung in x- und y-Richtung. Dieser Wert kann häufig bei 1 belassen werden. Danach folgt die Breite beziehungsweise Höhe der quadratischen Sobel-Filtermatrix. Der Defaultwert ist 3.

```
cvSobel( sobelinput, sobeloutput, 1, 1, 3);
```

Außerdem sind hier die Werte 1, 5 und 7 möglich. Aufbauend auf dem Sobel-Filter kann der Canny-Edge-Detektor zusammenhängende Kanten finden und ein binäres Kantenbild erzeugen. Seine Ausgabe ist ein binäres Bild mit schwarzem Hintergrund, auf dem die Kanten des Originalbilds in Weiß eingezeichnet sind.

Die Grundidee hinter dem Canny-Edge-Detektor ist ein Ausnutzen der vom Sobel-Filter berechneten Intensitäten in horizontaler und vertikaler Richtung, um die Richtung einer Kante zu bestimmen und ihren Verlauf weiterzuverfolgen. Hierzu wird zunächst ein Weichzeichner benutzt, um Rauschen aus dem Bild zu entfernen. Danach kommt der beschriebene Sobelfilter zum Einsatz. Aus dem Quotienten der Rückgabewerte des horizontalen und des vertikalen Kantendetektors läßt sich die Richtung der Kante bestimmen.

In dieser Richtung wird nun nach dem nächsten Pixel der Linie gesucht. Nachdem alle Kanten gefunden sind, versucht der Algorithmus, Linien, die nicht zusammengehören, aufzubrechen und Lücken in zusammengehörenden Linien zu schließen, um das endgültige Ausgabebild zu berechnen.

In der OpenCV heißt der Canny-Edge-Detektor *cvCanny*. Nach Übergabe des Ein- und Ausgabebilds bekommt die Funktion zwei Schwellwerte übergeben. Mit dem kleineren der beiden werden Lücken geschlossen und Kanten verbunden. Der größere dient dem Auffinden von Start- und Endpunkten von Kanten. Als letzten Parameter kann man hier die Größe der intern verwendeten Sobel-Filtermatrix übergeben:

```
cvCanny(cannyinput, cannyoutput, 50, 100, 3);
```

Dieser Beitrag zeigte die Grundfunktionen der OpenCV-Bibliothek, die sehr interessante Funktionen für das Anzeigen und Manipulieren von Bildern und Videos bietet. Neben diesen Grundfunktionen stellt die OpenCV aber auch eine fortgeschrittene Funktionalität für typische Anwendungen im Bereich Computer Vision zur Verfügung, wie das Verfolgen von Punkten und Objekten, das Erkennen von Objekten in einem Bild oder Video, das Ausblenden eines störenden, unbewegten Hintergrundes und viele weitere Algorithmen. ■