Barcode-Leser

Erkennung und Auswertung von Barcodes

AVPRG WiSe 2013/14 - Prof. Dr. Plaß

Stephen Fuhrmann

Martin Trapp

Installationsanleitung

Für die Ausführung der Applikation bedarf es keiner Installation. Die .exe wird direkt durch einen Doppelklick ausgeführt und die Applikation ist Funktionsbereit.

Bedienungsanleitung

Um ein Bild zu laden drücken sie den Butten mit der Aufschrift „Load File“.
Ein Dialog öffnet sich, in welchem sie ein Bild auswählen. Es werden die Formate png, bmp, jpg, tif und gif unterstützt. Nachdem das Bild geöffnet wurde erscheint es im Applikationsfenster. Wurde ein Barcode erfolgreich erkannt, wird dessen Inhalt im Textfeld mit der Aufschrift „Decoded Text“ dargestellt. Sollte kein Barcode erkannt worden sein, erscheint die Fehlermeldung „No Barcode detected“.
Es ist möglich den Inhalt des Barcodes in die Zwischenablage zu kopieren, dazu reicht es auf den Button mit der Aufschrift „To Clipboard“ zu klicken.

Abbildung 1: Erfolgreich gelesener Barcode

Grenzen und Ausbaumöglichkeiten

Der Barcodeleser ist im Stande Barcodes im Format UPC-A zu lesen.
Es ist prinzipiell möglich mit demselben Verfahren auch andere Handelsstrichcodes wie EAN, IAN oder JAN zu lesen. Dies könnte dann mithilfe von Interfaced-Klassen realisiert werden, wobei jede Klasse die Eigenheiten des jeweiligen Systems abdeckt.
Um einen Barcode erfolgreich lesen zu können sollte dieser nicht zu stark verrauscht und gedreht sein. Die Drehung des Barcodes sollte 38° nicht überschreiten, ansonsten kann dieser nicht mehr am Stück gelesen werden. Eine Drehung um 180° wäre prinzipiell möglich, ist in dieser Version des Programms aber nicht implementiert. Obwohl eine Funktion zur Berechnung der Prüfziffer vorhanden ist, kann es bei einem zu stark verrauschten Bild vorkommen, dass ein falscher Barcode mit Korrekter Prüfziffer gelesen wird, dieser wird dann als korrekt erkannt. Das Einlesen mittels live-stream wäre ebenfalls möglich.

Programmstruktur

Für die Umsetzung des Barcode-Lesers wurde die Programmiersprache C++ mit der Microsoft-Entwicklungsumgebung Visual Studio 2010 Professional benutzt. Für das Image-Processing, den Kern-Algorithmus des Lesers, wurde die Open-Source-Library openCV eingebunden. Für die grafische Oberfläche und Bedienelemente boten sich die Microsoft Foundation Classes (MFC) an.

Aufgrund der Arbeit mit MFC war die grundlegende Programmstruktur weitestgehend vorgegeben. Die grafische Oberfläche besteht aus einem einzelnen Dialog, der über zwei Buttons, „Load File“ und „Copy to Clipboard“, einem Textfeld, in dem der dekodierte Barcode angezeigt wird und einem Bild (Picture Control).

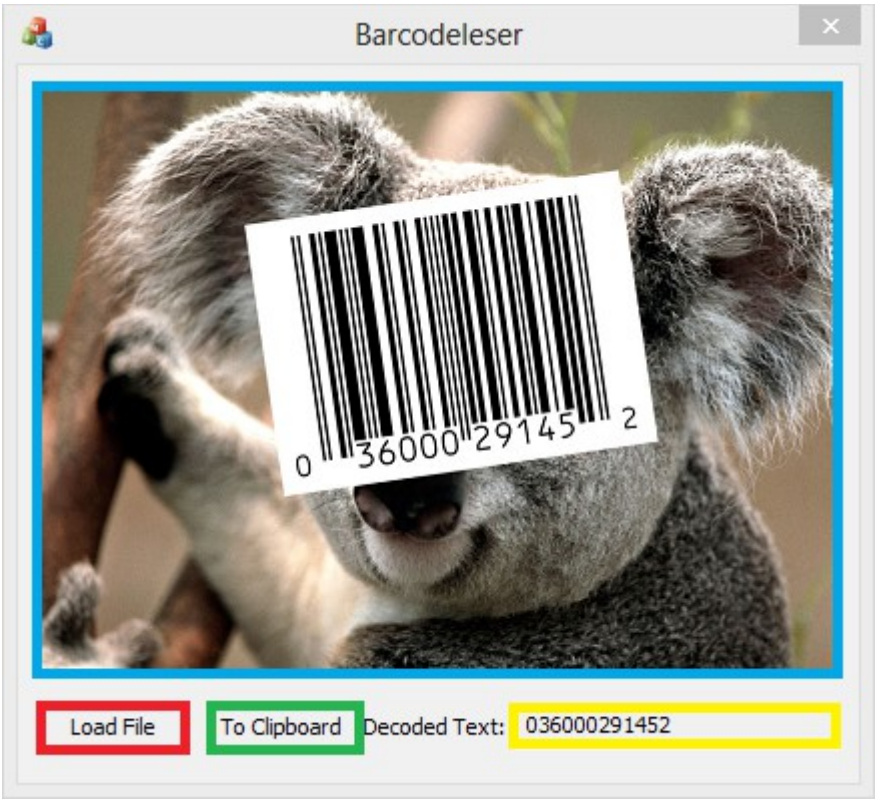

Abbildung 2: Elemente des Barcode-Leser-Dialogs

Struktur des Dialogs

Daraus ergibt sich bereits eine Struktur für die Programmierung: Eine Klasse für die gesamte Applikation (Barcode_Leser_Trapp_Fuhrmann) erzeugt den Dialog mit seinen Elementen (Barcode_Leser_Trapp_FuhrmannDlg). Dieser verfügt über folgende Handler für seine grafischen Elemente (siehe Abbildung :

- **CStatic Handler für die Picture Control (m_picture, ID_PICCTRL)**
- **CEdit Handler für das Textfeld (m_result, ID_RESULT)**
- **Event für das Klicken des Buttons „Load File“ (OnBnClickedLoadFile, ID_LOAD_FILE)**
- **Event für das Klicken des Buttons „To Clipboard“ (OnBnClickedCopy, ID_COPY)**

Neben diesen Membern besitzt der Dialog noch die Klasse, die das Image Processing durchführt (Barcode_Decoder).

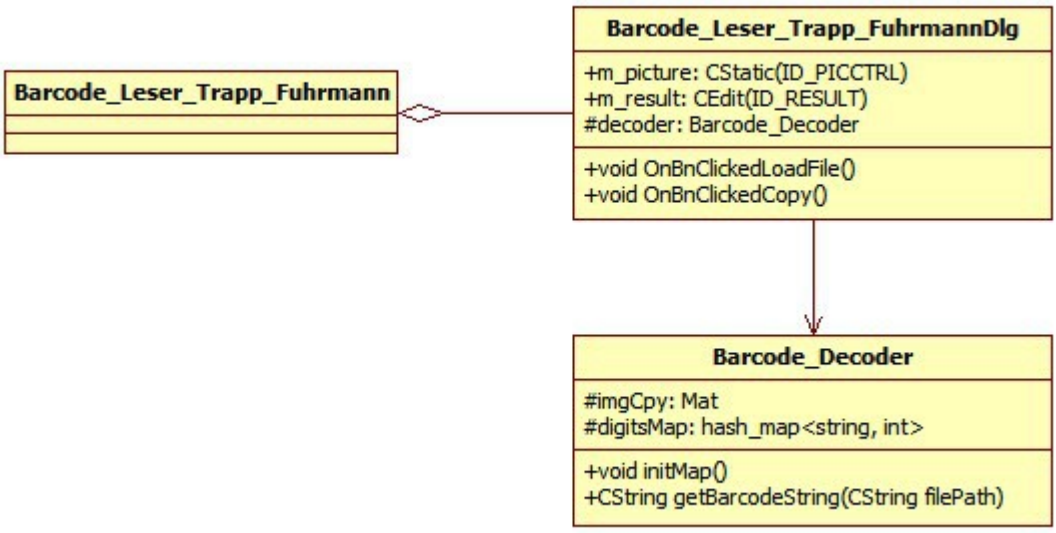

Diagramm 1: Klassendiagramm des Barcode-Leser-Codes – für die Struktur irrelevante Methoden und Attribute wurden nicht berücksichtigt

Funktionen der Handler innerhalb der Programm-Logik

Klickt der Benutzer auf den „Load File“-Button, so öffnet sich ein OpenFileDialog. Das geöffnete Bild wird mithilfe der Barcode_Decoder Klasse auf einen Barcode untersucht. Sie wirft dann die Zahlenfolge des Barcodes aus. Dieser wird mittels des CEdit m_result im Textfeld angezeigt. Außerdem wird das Bild mithilfe des CStatic m_picture auch im Dialog in der Mitte angezeigt. Danach kann der Nutzer auf „To Clipboard“ klicken, um den Text aus m_result in die Windows Zwischenablage zu kopieren und zu nutzen.
Eine weitere Verwendung (z.B. mittels einer Waren-Datenbank) ist nicht vorgesehen gewesen.

Aufbau UPC-A Barcode

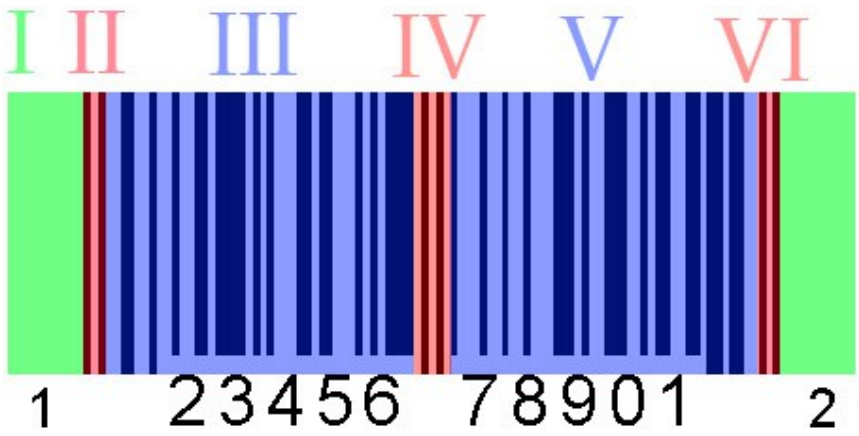

Abbildung 3: Aufbau UPC-A Barcode

1. Ruhezone
2. Linkes Randzeichen
3. Linker Ziffernblock
4. Trennzeichen
5. Rechter Ziffernblock
6. Rechtes Randzeichen

Auslesen eines Barcodes *(Barcode_Decoder.cpp)*

Sobald sich der User für ein Bild entschieden hat, wird dieses einem thresholding unterzogen, da der Barcode Monochrom ist und der Rest des Bildinhaltes für das Auslesen des Barcodes nicht interessant ist. Zudem erhöht es die Geschwindigkeit mit der das Programm ausgeführt wird.

Das Auffinden eines Barcodes im Bild geschieht durch Zeilen und Spaltenweise Analyse der Farbinformationen eines Pixels mittels einer einfach verschachtelteten For-Schleife. Wurden mehr als Zehn aufeinanderfolgende weiße Pixel in X-Richtung ausgemacht und stößt man nun auf ein Schwarzes Pixel, ist dies ein Indiz dafür die Ruhezone (1.) des Barcodes durchwandert zu haben. Die Ruhezone beinhaltet keinerlei verwertbare Informationen, sollte aber mindestens zehn mal die Breite eines Balken haben.
Als Ausgangsbasis dient hierbei ein Minimum von 10 Pixeln.
Ist dieses Kriterium erfüllt, ermittelt die Funktion *getUnitWidth()* die Breite eines einzelnen Balkens. Die Balken des Randzeichens (2.) folgen immer dem Muster Balken – Leerraum – Balken. Sowohl die Breite der Balken als auch die des Leerraums zwischen den Balken werden getrennt in ein Array geschrieben und miteinander ins Verhältnis gesetzt. Folgende Situationen können eintreffen.

- Die Breite des Bildes wurde beim Lesen erreicht, der Lesevorgang wird abgebrochen.
- Ist die Abweichung von Minimum zu Maximum Größer als 25% handelt es sich vermutlich nicht um das Randmuster des Barcodes sondern um ein beliebiges Muster aus dem Bild, oder das Randmuster ist zu zerstört um erfolgreich gelesen zu werden.
- Das Verhältnis vom schmalsten zum breitesten Balken ist innerhalb der Toleranz. Das Mittel der drei Breiten wird in die Variable *unit_width* geschrieben.

Nun Folgt der erste Datenblock bestehend aus Sechs Ziffern (3.). Für jede der Ziffern wird die Funktion *read_digit()* aufgerufen. Jede Ziffer besteht aus einer Kombination von vier Zeichen, welche Separat in einem Array gespeichert werden. Im Linken Block ist das erste Zeichen immer ein Leerzeichen. Da jede Ziffer die Kombination zweier Balken und Leerzeichen ist, ergibt sich die Arraybelegung *Leer – Balken – Leer – Balken*.
In der While-Schleife wird festgestellt wie breit in Pixeln die einzelnen Komponenten sind. Um ein interpretierbares Resultat zu bekommen werden die Ergebnisse noch durch die *unit_width* geteilt. Um Ein Ergebnis in der Range 0 bis 9 zu erhalten wird das Ergebnis mit einer Hashmap verglichen und zurückgegeben.

Nach dem ersten Ziffernblock folgt das Trennzeichen (4.), welches aus dem Muster *Frei – Balken – Frei – Balken – Frei* besteht. Dieses wird übersprungen, da es keine Interpretierbaren Daten enthält.
Beim zweiten Ziffernblock (5.) wird die Prozedur wiederholt, nur das alle Ziffern invertiert werden. Aus 001011 wird somit 110100.
Sind alle 12 Zeichen des Barcodes eingelesen, erfolgt eine Prüfung mittels Prüfziffer.
Die Prüfziffer ist die letzte Ziffer des Barcodes und kann mithilfe der restlichen Ziffern ausgerechnet werden. *[*1]*

1. Addiere alle ungeraden Stellen und multipliziere das Ergebnis mit 3
2. Addiere zur Summe alle geraden Stellen außer der Letzten (Prüfstelle)
3. Die positive Differenz zu einer durch Zehn Teilbaren Zahl, ist die Prüfziffer

→Bsp für Barcode *123456789012*

1. (1+3+5+7+9+1) * 3 = *78*
2. *78* + 2+4+6+8+0 = *98*
3. 10 – *98*mod10 = **2**

Vergleicht man nun das Ergebnis mit der letzten Ziffer des Barcodes und sind beide Identisch, handelt es sich um einen gültigen UPC-A Barcode.

Quellen

*[*1]* https://en.wikipedia.org/wiki/Check_digit#UPC