|  |
| --- |
|  |
| Barcodereader |
| Dokumentation über die Erstellung mit OpenCV für das Fach AV-Programmierung |
|  |
| **Alexander Piehl, Thomas Ehlers** |
| **11.01.2014** |

|  |
| --- |
| Dieses Dokument enthält eine kurze Umschreibung des Entwicklungsprozesses und eine Anleitung für das Programm. |

# Installationsanleitung

Das Programm wird als eine Datei mit der Endung .exe geliefert. Diese startet gleich das Programm. Eine extra Installation ist somit nicht nötig.

# Bedienungsanleitung

## C:\Users\Bla\Documents\GitHub\AVPRG-Projekt\SSfuerAnleitung.jpgAnleitung zum nutzen des Programms

Programm nach dem Start 1

Abbildung 1 Programm nach dem Start

Als erster Schritt wird die Datei „Barcodereader.exe“ mit einem Doppelklick gestartet.

Nun sollte sich das obige Fenster öffnen.

1. Der „Lade“-Knopf: Dieser Knopf erlaubt es, nach dem drücken, eine Bilddatei auszuwählen. Die üblichen Formate werden hier funktionieren. Falls eine Datei ausgewählt wird, die keine Bilddatei ist oder aber nicht gelesen werden kann, wird es eine Fehlermeldung geben.  
   Es muss dann eine andere Datei ausgewählt werden, bevor das Programm genutzt werden kann.  
   Wurde ein Bild geladen wird es in „5“ angezeigt.
2. Der „Start“-Knopf: Dieser Knopf ist dafür zuständig, einen Barcode auf dem vorher ausgewählten Bild zu erkennen und auszulesen.  
   Der Knopf ist solange ausgegraut, bis ein Bild eingelesen wurde. Wenn wieder eine falsche Datei nach dem Öffnen einer richtigen Datei eingelesen wird, wird der Knopf wieder grau.  
   Er kann somit nur benutzt werden, wenn ein korrektes Bild geladen wurde. In diesem Fall ist dann ein einfacher Mausklick nötig um das Programm zu starten.
3. Das Code Drop-Down-Menü: Hier kann zu jeder Zeit die Art des auszulesenden Barcodes ausgewählt werden. Zur Verfügung stehen hier der „UPC-Code“ und „Code39“.  
   Die Auswahl erfolgt mit einen einfach Mausklick. Nach dem Klick „öffnet“ sich das Menü und man kann mit einem weiteren Klick auf den jeweiligen Code, diesen auswählen. Falls es unsicher ist, welcher Code benötigt wird, kann man diese auch nacheinander ausführen, indem man einfach beide nacheinander startet.
4. Das „Ergebnis“: Hier wird nach drücken das „Start-Knopfes“ das Ergebnis angezeigt. Sofern ein Barcode gefunden wurde.  
   Falls kein Barcode gefunden wurde, bleibt dieses Feld leer.
5. Das „Bildfenster“: Nach Auswahl einer Bilddatei wird hier das Bild angezeigt.

Richtig ausgeführtes Programm 1

Erfolgreich ausgeführtes Programm 1

Abbildung 2 Erfolgreich ausgeführtes Programm

## Fehlermeldungen

Das Bild konnte leider nicht geladen werden. Bitte wählen Sie ein anderes Bild aus

Es kann sein, dass keine Bilddatei ausgewählt wurde oder aber die Bilddatei beschädigt ist.

### Es konnte kein Barcode ermittelt werden

Hier gibt es verschiedene mögliche Ursachen.

* Auf dem Bild befindet sich kein Barcode
* Der Barcode auf dem Bild entspricht nicht der ausgewählten Art des Barcodes. Oder ist eine Art die nicht mit diesem Programm ausgelesen werden kann.
* Der Barcode ist beschädigt und kann deswegen nicht ausgelesen werden

### Es gab einen Fehler beim internen laden der Bilddatei

Beim internen Laden des Bildes in das Programm ist ein Fehler aufgetreten. Zum Beispiel könnte die Verbindung mit dem Datenspeicher des Bildes während des Ladevorganges unterbrochen worden sein.

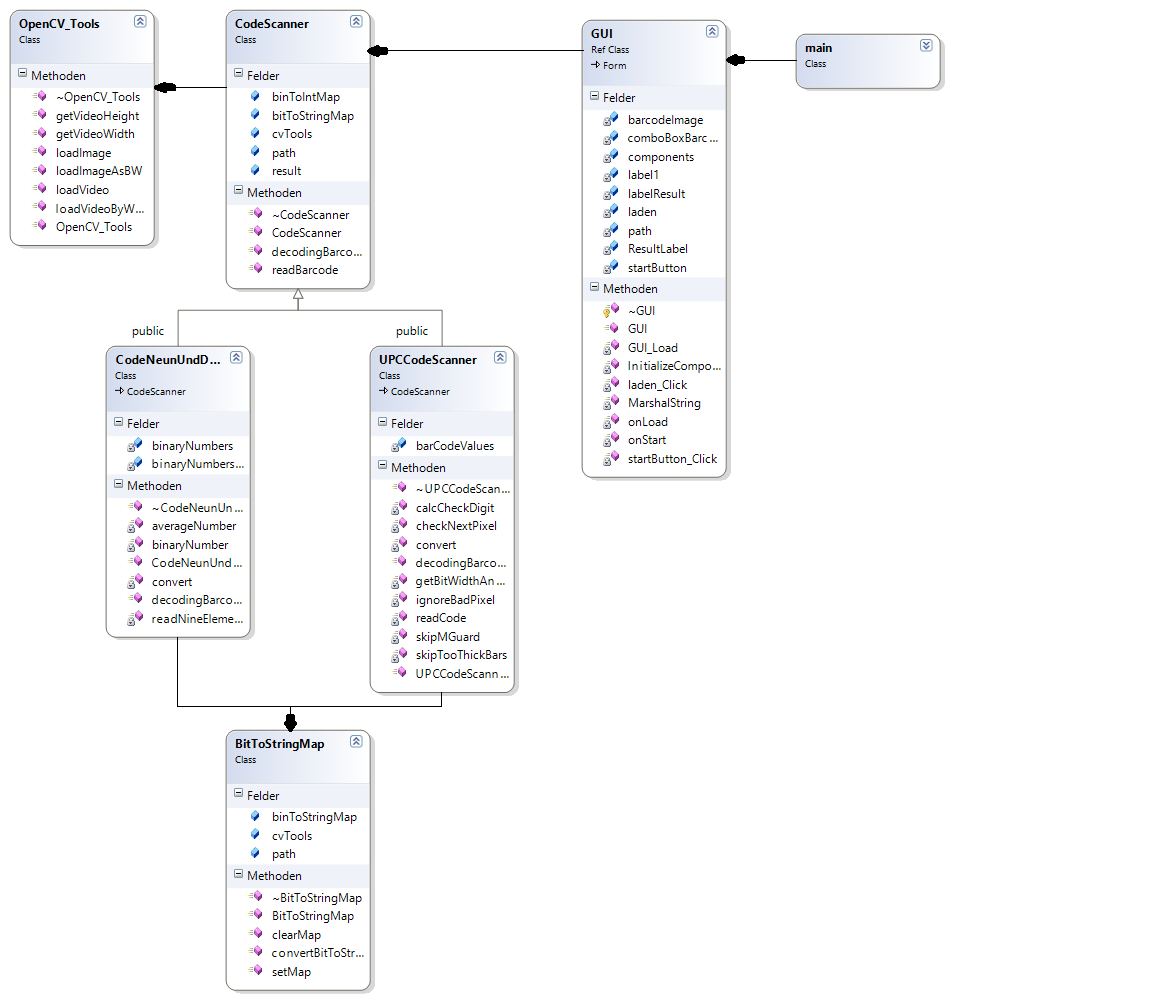
# Programmstruktur

Das Programm besteht aus einer Klasse „main“, die die GUI aufruft.

Die GUI wiederum ruft dann je nach ausgewähltem Barcode die entsprechende Klasse auf. Entweder „UPCCodeScanner“ oder „CodeNeunUndDreizig“. Diese Beiden Klassen erben von der Klasse „CodeScanner“.

Zum Auswerten der Binären Zahlen des Barcodes wird dann die Klasse „BitToStringMap“ aufgerufen, die dann Zahlen, Buchstaben und Zeichen je nach ausgewähltem Code zurückgibt.

# UML-Diagramm



# Beschreibung des UPC-Code-Scanners

## Begriffserklärung

1. Ruhezone oder Quiet-Zone
2. Linke und rechte Guard
3. Linker und rechter Datenblock
4. Mittlere Guard

Die linke und rechte „Guard“ sind identisch miteinander. Sie bestehen aus einem schwarzen, weißen, schwarzen Strich. Diese Striche legen die Standardbreite für einen einzelnen Strich im gesamten Code fest. Theoretisch sollten sie alle die gleiche Breite haben.

Die mittlere „Guard“ besteht aus einem weißen, schwarzen, weißen, schwarzen, weißen Strich, die auch alle der Standardbreite entsprechen.

Jeder Datenblock entspricht 6 Zahlen, die jeweils in 7Bits codiert sind. Eine bestimmte Bitreihenfolge steht dann für einen bestimmten Wert im Dezimalsystem.

## Beschreibung

UPCCodeScanner.cpp erbt von der Klasse CodeScanner.cpp und ist dafür zuständig aus eingelesenen Bildern UPC-Codes zu erkennen und auszulesen.

Zu diesem Zweck geht zuerst eine Methode, die von CodeScanner.cpp geerbt wird durch das Bild und sucht nach einer Ruhezone im Bild. Eine solche Ruhezone besteht aus einer bestimmten Anzahl weißer Pixel. Diese errechnet sich aber aus Werten die durch den Barcode ermittelt werden und stehen somit am Anfang des Programms nicht zur Verfügung. Von daher wird dieser wert auf zehn weiße Pixel festgelegt, die mindestens gefunden werden müssen.

Nach diesen mindestens zehn weißen Pixeln würde nun jedes gefundene schwarze Pixel zur Ausführung der Methode führen, die nach UPC-Code (oder auch bei jeweiliger Auswahl nach Code39) sucht und ihn entschlüsselt.

Zu diesem Zweck wird zuerst die „Guard“ gesucht. Diese folgt dem einfachen Muster, einen Schwarzen, einen Weißen und einen Schwarzen Strich zu finden. Deren breite wird dabei gespeichert.

Wenn zwei der drei Striche die gleiche Breite haben, wird davon ausgegangen die „Guard“ gefunden zu haben. Es wird nicht geprüft ob alle drei gleich groß sind, da dadurch kleine Fehler im Bild das Auslesen unmöglich machen würden. Falls so eine falsche „Guard“ gefunden wird, wird es spätestens durch die Prüfziffer erkannt.

Wird keine „Guard“ erkannt, geht es zurück zum Suchen nach einem Anfang und einer Guard.

Wenn eine „Guard“ gefunden wurde, läuft das Programm allerdings weiter und liest je nach Farbe und Breite der nächsten sieben Striche den Binärwert dieser aus.

Beim Lesen dieser Striche wird noch überprüft, ob ein Strich mal zu kurz oder aber zu lang ist und gleicht dies dann dementsprechend aus. So kann der Code durch kleine Fehler im Bild oder Fehler bei der Erstellung trotzdem lesbar bleiben. Mehr als ein Pixel wird hier aber nicht toleriert, da größere Diskrepanzen bei der „Guard“ zu einer sehr starken Ungenauigkeit führen könnten.

Dies macht das Programm nun sechs Mal hintereinander, um die sechs Werte zwischen der ersten rechten „Guard“ und der mittleren „Guard“ zu ermitteln. Gleichzeitig wandelt sie die Binärzahlen durch eine Tabelle in Form einer „Map“ in echte Zahlen um und speichert sie.

Nachdem dies passiert ist, wird die mittlere „Guard“ übersprungen, um die sechs rechts stehenden Zeichen auf die gleiche Weise wie auf der linken Seite auszuwerten.

Als letztes wird nun noch mit Hilfe der letzten Zahl eine Prüfsumme errechnet. Ist diese richtig hat man den Barcode gefunden und ausgelesen.

Falls irgendwo im Laufe dieser Prozedur ein Fehler auftritt, springt das Programm immer zurück an den Anfang. Dort versucht es dann mit Hilfe der letzten x- Position weiter nach dem Barcode zu suchen.

# Externer Source Code

Die Methode „MarshalString ( String ^ s, std::string& os )“ in der Klasse GUI ist externer Code. Sie stammt vom Microsoft Developer Network(MSDN), <http://msdn.microsoft.com/de-de/library/1b4az623.aspx> .

# Quellen

<http://de.wikipedia.org/wiki/Barcode>

<http://www.strichcode.co.at/code39.htm> (Binär-Tabelle)

<http://de.wikipedia.org/wiki/Code39>

<http://felix.abecassis.me/2011/10/opencv-barcode-reader-part-1/(Binär-Tabelle)>

<http://msdn.microsoft.com/en-us/library/3bstk3k5.aspx>

<http://www.cplusplus.com/reference/>

<http://msdn.microsoft.com/de-de/library/1b4az623.aspx>