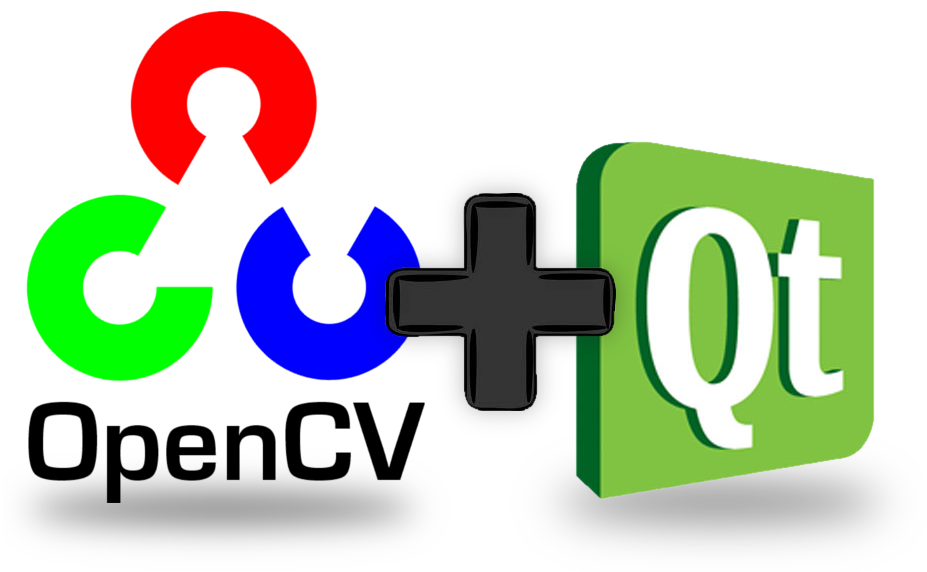
Puzzle Solver mit OpenCV und Qt

Blabla schöusslechi Titusite blabla



<inhaltsverzeichnis>

# Problemstellung

Gegeben ist ein Bild mit einem Puzzle. Nun soll ein Algorithmus geschrieben werden, der herausfindet, wie die Puzzleteile untereinander zusammenpassen. Als Bildverarbeitungsbibliothek wird die frei verfügbare und aktiv weiterentwickelte Plattform OpenCV in C++ verwendet. Als Programmierumgebung wird der Qt Creator verwendet, welcher mit der Qt GUI-Bibliothek später eventuell auch noch eine graphische Oberfläche für das Programm ermöglicht.

# Lösungsansatz

Zuerst wird das Bild binarisiert. Anschliessend werden aus dem binarisierten Bild die einzelnen Puzzleteile extrahiert. Nun wird aus der Kontur des einzelnen Puzzleteiles die vier Eckpunkte des Grundrechtecks gefunden. Anhand dieser Eckpunkte können nun die vier Seitenkonturen des Puzzleteils bestimmt werden. Diese Seitenkonturen werden nun auf verschiedene Features untersucht. Schlussendlich werden die Seitenkonturen von allen Puzzleteilen miteinander verglichen und die am besten passenden entsprechend markiert.

# Implementation

Die Implementation des Algorithmus konnte gut nach der Vorlage aus dem Lösungsansatz entwickelt werden.

## OpenCV Routinen

Die Folgenden Routinen wurden zum Dreh- und Angelpunkt unseres Projektes:

### findContours

Dieser Funktion wird ein binarisiertes Bild übergeben. Sie versucht nun nach dem Algorithmus Suzuki85, Konturen im Bild zu finden. Mit den diversen Parametern kann noch gewählt werden, ob hierarchische Konturen (also Konturen innerhalb anderer, geschlossener Konturen) ebenfalls gesucht werden und wie diese hierarchisch abgespeichert werden sollen. Ausserdem kann die Art der Konturlinie gewählt werden – also zum Beispiel 4-connected, 8-connected etc.

### approxPolyDP

Diese Funktion versucht, eine Kontur zu vereinfachen.

Hier werden nun die einzelnen Schritte genauer beschrieben:

## Binarisieren

Blablba

## Puzzleteile extrahieren

Asdf

## Kontur auf Eckpunkte analysieren

Um die Eckpunkte zu suchen, wurde die Routine approxPolyDP genutzt. Der frei einstellbare Parameter *epsilon*, der der Funktion mitgegeben wird, wurde so gewählt, dass nur gerade and den vier Eckpunkten des Grundrechteck der Winkel zwischen zwei Geraden ungefähr 90° beträgt. Überall sonst soll er kleiner sein. Hier mit den Werten epsilon = 0, 2, 10, und 64:

|  |  |
| --- | --- |
| C:\Users\Cyril\Pictures\Screenpresso\2012-12-23_15h39_06.png | C:\Users\Cyril\Pictures\Screenpresso\2012-12-23_15h40_41.png |
|  |  |
| C:\Users\Cyril\Pictures\Screenpresso\2012-12-23_15h40_47.png | C:\Users\Cyril\Pictures\Screenpresso\2012-12-23_15h40_56.png |

Die blaue Kontur wurde mit der Routine findContour gefunden, das rote Rechteck ist das minimale umfassende Rechteck und die gelbe Linie ist die approximierte Kontur. Wie man gut sehen kann, eignet sich der Algorithmus bei epsilon ungefähr gleich 10 am besten, da sich dort die Ecken klar als 90° Winkel von den anderen Winkel abheben.

## Seitenwände analysieren

Asdf

## Seitenwände vergleichen

Asdf

## Passende Teile darstellen

Asdf

# Fazit

Blabal OpenCV mächtig blabla

Blabla hat funktioniert blabla

asdf

## Mögliche Erweiterungen

### Andere Seitenwandanalysemethoden

Man könnte statt einfach die vordefinierte Funktion matchShapes eigene Features definieren:

* Umrisslinienlänge relativ zur Länge der entsprechenden Rechtecklinie darunter
* Blablaba

### Nebst Seitenwandanalse noch Farbanalyse

Man könnte nebst dem Analysieren der Form der Seitenwände auch noch die Farbe vergleichen, welche die Seitenwand gerade am äussersten Punkt hat. Diese muss dann innerhalb einer gewissen Toleranz mit der Farbe auf dem gegenüberliegenden Puzzleteil übereinstimmen.

### Puzzle automatisch zusammensetzen

Statt nur zu beschreiben, welche Seitenwand wohin gehört, könnte man die Puzzleteile auch noch ausschneiden und so zusammensetzen, dass das Gesamtbild sichtbar wird.

Syntax highlighting:

<http://markup.su/highlighter/>

* Style : entweder Blackboard, Eiffel, Mac CLassic,