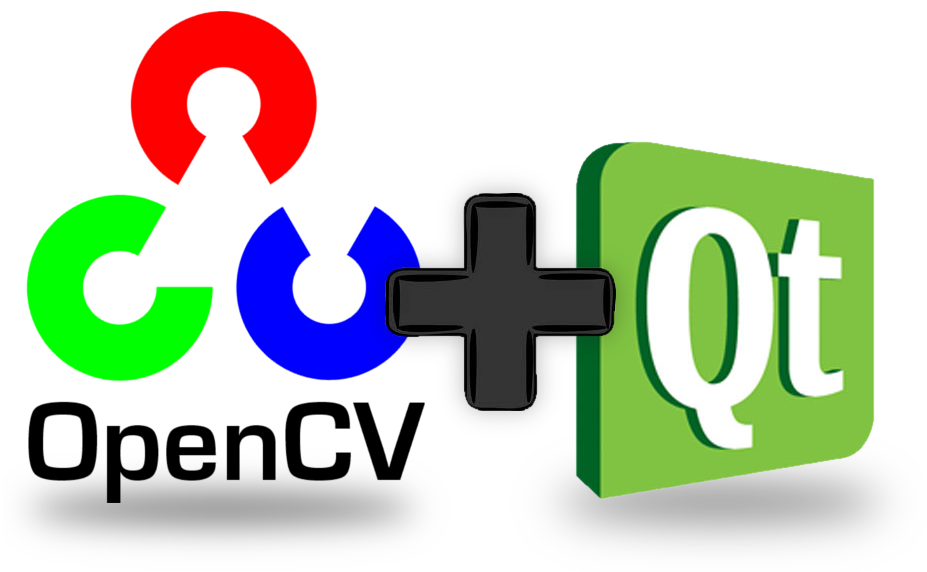
Puzzle Solver mit OpenCV und Qt





Inhalt

[Problemstellung 3](#_Toc344636272)

[Lösungsansatz 3](#_Toc344636273)

[Implementation 3](#_Toc344636274)

[OpenCV Routinen 3](#_Toc344636275)

[findContours 3](#_Toc344636276)

[approxPolyDP 3](#_Toc344636277)

[Binarisieren 3](#_Toc344636278)

[Puzzleteile extrahieren 3](#_Toc344636279)

[Kontur auf Eckpunkte analysieren 4](#_Toc344636280)

[Seitenwände analysieren 5](#_Toc344636281)

[Seitenwände vergleichen 5](#_Toc344636282)

[Passende Teile darstellen 5](#_Toc344636283)

[Fazit 5](#_Toc344636284)

[Mögliche Erweiterungen 5](#_Toc344636285)

[Andere Seitenwandanalysemethoden 5](#_Toc344636286)

[Nebst Seitenwandanalse noch Farbanalyse 5](#_Toc344636287)

[Puzzle automatisch zusammensetzen 5](#_Toc344636288)

# 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 werden 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 einem speziellen Algorithmus 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 nach dem Ramer–Douglas–Peucker ALgorithmus, eine Kontur zu vereinfachen. Es muss noch ein Parameter *epsilon* angegeben werden. Mit diesem kann man steuern, wie stark eine Kontur vereinfacht wird.

### matchShape

blablablabla

Hier werden nun die einzelnen Schritte genauer beschrieben:

## Binarisieren

Zuerst wird das Bild in ein 8-Bit Graustufenbild umgewandelt. Um grobe Verrauschungen zu entfernen, wird das Bild mit einem Blur-Algorithmus verschmiert.

Die nachfolgende Binarisierung geschieht mit der Threshold funktion, welcher als Schwelle der Mittelwert des Bildes mitgegeben wird. Um die Binarisierung flexibler zu gestalten, haben wir versucht die Funktion AdaptiveThreshold zu implementieren. Jedoch ergab diese mehr eine Kantenerkennung als eine Binarisierung. Als Verbesserung könnte hier auch das Histogramm des Bildes ausgeglichen und anschliessend eine Binarisierung vorgenommen werden.

Nun wird das Bild noch einmal entrauscht, diesmal mit morphologischen Operationen: Mit einem 3x3 Filterkern wird das Bild zwei Mal geöffnet (zwei mal erodieren und zwei mal dilatieren).

## 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 an 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

Jetzt wird die Kontur jedes Puzzleteils an den Eckpunkten geteilt und so vier Seitenwände pro Puzzleteile extrahiert.

Dannach werden die Seitenwände auf ihr Geschlecht analysiert. Herausragende Teile sind definieren wir als „positiv“, Einbuchtungen als „negativ“ und gerade Wände als „neutral“. Dies dient zur Reduktion der Anzahl Vergleiche, welche so mehr als halbiert wird.

## Zu vergleichende Seitenwände auswählen

Wir haben uns aus folgenden Gründen entschieden, nicht das ganze Puzzle zu analysieren:

* Es ist kompliziert, die verschiedenen Vergleichskombinationen zwischen den Seitenwänden aller Puzzleteile zu finden.
* Es ist zeitaufwändig.
* Es kann ohnehin nicht übersichtlich dargestellt werden, vorausgesetzt wir fügen das Puzzle nicht schon ganz zusammen. Aber das ist hier nicht die Aufgabenstellung.

Wir vergleichen nur jeweils eine Seitenwand mit allen anderen im Bild. Mit der Maus kann eine Seite angeklickt werden. Die Übereinstimmungen mit den anderen passenden (passend heisst engegengesetztes Geschlecht) Seitenwänden wird dann mit stärker oder schwächer gefärbten Konturlinien angezeit und die beste Übereinstimmung mit einer Bezierkurve verbunden. Um eine Seite anzuwählen, kann auf dem Originalbild mit der Maus auf eine Seite gedrückt werden.

Um einen Mausklick zu registrieren muss der MouseCallback auf das entsprechende Fenster gesetzt werden. Danach kann in der Funktion onMouse die gewünschte Funktion eingefügt werden. In unserem Fall wird dort nur die Position des Klicks in einer globalen Variable gespeichert. Die eigentliche Auswertung geschieht danach in der Hauptroutine. Dort wird die Distanz der Mausposition mit den Puzzleteilen mit der Funktion pointPolygonTest gemessen, und das Puzzleteil mit dem kleinsten Abstand zur Mausposition ausgewählt.

Um noch die Seitenwand des entsprechenden Puzzleteils zu bestimmen, wird die Position noch mit den Seitenwandschwerpunkten verglichen und auch wieder die Seite mit dem kleinsten Abstand gewählt.

## Seitenwände vergleichen

Um die Seitenwände zu vergleichen können verschiedene Features erstellt werden. Wir verwenden nur die Funktion matchShapes um zwei Seitenwände miteinander zu vergleichen. Da die Funktion für gute Übereistimmung einen tiefen Wert zurückgibt, wird davon der Kehrwert genommen und abgespeichert.

## Passende Teile darstellen

Lorem Ipsum sit hat dolor

# Fazit

Schon von Beginn weg waren wir der Überzeugung, dass das Projekt in OpenCV realisierbar ist. Jedoch haben wir zuerst versucht, die OpenCV-Bibliothek selbst zu kompilieren und Microsoft VisualStudio als Entwicklungsumgebung zu verwenden. Dieses Vorhaben scheiterte und wir entschieden uns, den bereits vorhandenen QtCreator zu verwenden. Die Software MATLAB kam für uns nicht in Frage, da wir nur die Studentenversion besitzen, bei welcher die Bildverarbeitungs-Toolbox nicht dazugehört. Zudem gingen wir davon aus, dass OpenCV besser dokumentiert ist als die MATLAB Bildverarbeitungsbibliothek (Was wir aber nicht beurteilen können, da wir nicht mit MATLAB gearbeitet haben).

Die OpenCV Bibliothek ist sehr mächtig und umfassend. Beinahe für jedes Problem gibt es eine Funktion, die sehr intuitiv benutzbar ist. Auch die Dokumentation ist sehr umfangreich und es gibt eine Menge Beispielprojekte. Das grösste Problem liegt darin, die richtige Funktion zu finden und sie richtig einzusetzen.

Auch der QtCreator ist praktisch und mit dem im Kurs behandelten Tutorial, fällt der Einstieg leicht.

Unser grösstes Manko lag in der Programmiersprache C++. Es war unser erstes Projekt in dieser Sprache, was die Handhabung der Vektoren und anderen Datentypen erschwert hat. Zudem haben wir wahrscheinlich vom Aufbau und Ablauf des Programmes eher einen C-Code in C++ geschrieben. Der objektorientierte Ansatz kommt wenig bis gar nicht zum Vorschein.

Schlussendlich sind wir mit dem Ergebnis zufrieden. Für unser vorgegebenes Puzzleteilchen funktioniert der Algorithmus recht gut. Jedoch könnte er noch mit weiteren Features verbessert werden. Auch der Binarisierungs-Algorithmus funktioniert noch nicht für alle Bilder. Zudem wird das Programm nicht beendet, wenn das Bild nicht gelesen werden konnte.

Abschliessend können wir sagen, dass wir einen guten Einstieg in die Bildbearbeitung gehabt haben und für weitere Projekte gerüstet sind und auf etwas zurückgreifen können.

## Mögliche Erweiterungen

### Andere Seitenwandanalysemethoden

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

* Umrisslinienlänge relativ zur Länge der entsprechenden Rechtecklinie darunter

### 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,