Nummernschilddetektion

EDBV WS 2015/2016: AG_E1

Simon Maximilian Fraiss (1425602)
Patrick Hromniak (1425731)
Michael Pointner (1427791)
Simon Reisinger (1426220)
Mohammad Zandpour (1425603)

8. Januar 2016

1 Gewählte Problemstellung

1.1 Ziel

Ziel ist es, auf einem Foto von Autos die Nummerntafeln zu lokalisieren, die perspektivische Verzerrung herauszurechnen und alle so gefundenen Nummerntafeln neben dem Originalbild anzuzeigen.

1.2 Eingabe

Ein Foto in einem von MATLAB unterstützten Format, welches die unten beschriebenen Bedingungen erfüllt.

1.3 Ausgabe

Das Ergebnisbild besteht aus dem Originalbild, in welchem die Ecken der Schilder marikert wurden. Daneben befindet sich ein weißer Bereich, in welchem die entzerrten Nummerntafeln zu sehen sind, sofern welche gefunden wurden.

1.4 Voraussetzungen

- Die zu erkennenden Nummernschilder müssen vollständig sichtbar sein und dürfen nicht teilweise verdeckt sein.
- Die Nummernschilder sollten auf dem Bild klar sichtbar, erkennbar und nicht zu klein sein.

- Es werden nur Eurokennzeichen mit dem blauen EU-Bereich auf der linken Seite erkannt.
- Das Nummernschild sollte nicht stark gedreht sein oder gar auf dem Kopf stehen.
- Damit die Entzerrung funktioniert, sollte das Bild nicht zu nah an der Linse bzw. so stark verzerrt sein, dass die Ränder bereits rund erscheinen (keine Fischaugenobjektive z.B.)
- Die Schilder sollten möglichst gleichmäßig beleuchtet und bei Tageslicht aufgenommen worden sein.
- Das Eingabefoto sollte eine Auflösung von mindestens 1200 Pixel in der Breite haben.

1.5 Methodik

Unsere Methodik wurde durch [3] inspiriert. Um das Bild in einen algorithmisch interpretierbaren Zustand zu versetzen, verkleinern wir das Bild auf eine Maximalbreite von 1200 Pixel und erzeugen aus dem Graustufenbild ein Kantenbild mittels *Canny edge detection* um in weiterer Folge jede einzelne umschlossene Region des Bildes als Fläche zu markieren (Abbildung 1). Da im Kennzeichen viele Zahlen und Buchstaben vorkommen, entsteht durch das vorherige Füllen eine Fläche mit vielen Löchern (aufgrund der Beschriftung), was ebenso als Entscheidungskriterium zur Erkennung der richtigen Fläche herangezogen wird.

Abbildung 1

Bisher stehen uns eine Reihe löchriger Flächen zur Verfügung. Wir filtern nun weiter indem wir prüfen, ob die Bilder viereckig sind und speichern die Position der Ecken. An dieser Stelle werden die gefundenen viereckigen Objekte auf die ursprüngliche Größe hochskaliert. Hier sollte jetzt eine Überprüfung der Farb-/Kontrastverhältnisse gefundener Gebilde stattfinden, die Eurozeichenüberprüfung ist jedoch im Bezug zur farblichen Analyse hinreichend genug und macht vorher genanntes obsolet. Nun folgt eine geometrische Transformation um die Verzerrung zu korrigieren und die Flächen gerade zu richten. Die linke Seite des Schilds wird auf das Vorhandensein blauer Farbe geprüft. Sollte das gesuchte Blaue nicht vorhanden sein, so fehlt die Europa-Kennung und es werden etwa 8% der Breite des Bildes zur Linken hinzugefügt, sofern in den 8% auch Blaues auftritt. Falls nicht, so kommt es als Kennzeichen nicht in Frage und wird verworfen.

Zu guter Letzt werden alle verbliebenen Flächen noch auf das Vorhandensein von Zahlen und Buchstaben geprüft. Dies geschieht mittels durch eine von MATLAB bereitgestellte *Optical Character Recognition*-Programmierbibliothek. Durch Adjustierung der Parameter lässt sich das richtige Objekt ermitteln welches das Kennzeichen sein muss. Dieses wird zum Schluss im Output neben dem Originalbild angezeigt.

1.6 Evaluierungsfragen

- Sind alle gefundenen Objekte Schilder oder wurden falsche Objekte detektiert?
- Wurden alle vollständig sichtbaren Schilder detektiert?
- Woran scheitern nicht gefundene Objekte?
- Wie viele Prozent der Testbilder führten zu einem korrekten Ergebnis?
- Sind die Schilder korrekt transformiert?
- Sind die Schilder vollständig oder fehlt ein Teil?
- Wie verhält sich die Laufzeit des Algorithmus, worin liegen Probleme?

1.7 Zeitplan

1.7.1 Ursprünglicher Plan

Meilenstein	**Abgeschlossen Am**
Prototyp mit Matlab-Funktionen	30. November
Implementierung von Detektion	24. Dezember
Implementierung von Geometrischer Transformation	24. Dezember
Testphase	1. Jänner
Abschlussbericht	7. Jänner

1.7.2 Tatsächlicher Zeitplan

Die ursprünglichen Meilensteine waren schlecht gewählt, da ein ernstzunehmender Prototyp ohne selbstimplementierte Methoden gar nicht möglich war. Außerdem wurde der Aufwand der Geometrischen Transformation überschätzt.

Meilenstein	**Abgeschlossen Am**
Erster simpler Prototyp	19. Oktober
Geometrische Transformation	20. November
Flächenfindung	23. November
Viereckprüfung und Eckendetektion	24. November
Buchstabenprüfung	5. Dezember
Histogrammprüfung	15. Dezember
Eurozeichentest	17. Dezember
Canny-Filter	2. Jänner
Bericht	8. Jänner

2 Arbeitsteilung

Name	**Tätigkeiten**
Fraiss Simon Maximilian	Interpolation in Geometrischer Transformation Viereckprüfung und Eckendetektion Überarbeitung des Konzeptes Berichtabschnitte zur Viereckprüfung und Eckendetektion Berichtabschnitt zur Implementierung der Interpolation
Hromniak Patrick	Histogrammprüfung Berichtabschnitte zur Histogrammprüfung Berichtabschnitt zur Evaluation
Pointner Michael	Geometrische Transformation Eurozeichenprüfung Erstellung des Ausgabebildes Closing des Canny-Bildes Berichtabschnitte zur Geometrischen Transformation Berichtabschnitt zum Canny-Closing Datensatz und Evaluation
Reisinger Simon	Flächenfindung in Kantenbild Canny-Filter Berichtabschnitte zum Canny-Filter Berichtabschnitte zur Flächenfindung Datensatz und Evaluation
Zandpour Mohammad	Erster simpler Prototyp Buchstabenprüfung Überarbeitung des Konzeptes (Methodik) Berichtabschnitt zur Buchstabenprüfung Bericht-Schlusswort

3 Methodik

3.1 Canny Algorithmus

Der Canny Algorithmus ist in mehrere Schritte unterteilt, in denen nacheinander das Bildrauschen unterdrückt, die Kanten detektiert, auf eine Kantenstärke von 1 reduziert und schwache Kanten entfernt werden, die nicht mit starken Kanten verbunden sind. Der Algorithmus basiert auf [4], [2] und [6].

3.1.1 Rauschunterdrückung

Starke Intensitätsänderungen werden als Kanten definiert. Da nicht nur die Kanten sondern auch das Bildrauschen eine starke Unstetigkeit der Grauwertfunktion aufweist, muss Letzteres mit einem Gauß-Filter herausgefiltert werden, um die Detektion von falschen Kanten zu verringern.

3.1.2 Kantendetektion

Bei der Kantendetektion wird die Matrix des Bildes für jedes Pixel mit zwei Kantendetektionsoperatoren gefaltet. Die Ergebnisse der Faltung liefern Werte für die erste Ableitung der Intensitätsfunktion, der Steigung, in vertikaler (G_X) als auch in horizontaler (G_Y) Richtung. Daraus können zwei wichtige Größen der Kanten berechnet werden: die absolute Kantenstärke G als auch die Orientierung θ.

$$G = \sqrt{G_x^2 + G_y^2} \tag{1}$$

$$\theta = atan2(G_y, G_x) \tag{2}$$

3.1.3 Non-maximum suppression

Die nun berechneten Kanten sind zum größten Teil mehrere Pixel breit. Damit jede Kante genau definiert wird, muss die Kantenstärke auf 1 reduziert werden. Das Bild wird entlang der 4 Richtungen, an den Kanten verlaufend, gescannt. Wenn es orthogonal zum Kantenverlauf einen Nachbarpixel gibt, der einen höheren Intensitätswert aufweist, ist das aktuelle Pixel kein lokales Maximum und wird daher auf 0 gesetzt.

3.1.4 Schwellwerte mit Hysteresis

Das durch Non-maximum suppression entstehende Bild enthält nun zwar nur noch ein Pixel starke Kantenen, doch wurde bis zu diesem Zeitpunkt noch nicht die Intensität der Kante berücksichtigt. Hierfür werden zwei Schwellwerte T_1, T_2 mit T_1 ¡ T_2 eingeführt: Jedes Pixel, des einen Intensitätswert höher als T_2 hat wird als Kantenelement gewertet und jeder kleiner als T_1 wird nicht als Kantenelement gewertet. Jeder zwischen T_1 und T_2 wird als Kantenelement gewertet, wenn er in Nachbarschaft eines Kantenelements liegt.

3.2 Floodfill Algorithmus

Der Floodfill Algorithmus wird zur Füllung geschlossener Flächen verwendet. Hierbei werden Flächen zusammenhängender Pixel einer Farbe gefüllt.

3.3 Viereckprüfung und Eckendetektion

Es muss geprüft werden, ob die gefundene Form viereckig ist. Ein erster Ansatz war, einfach die corner-Funktion von Matlab zu nutzen, welche wahlweise mit dem Harris-Detektor oder der „Minimum Eigenvalue“-Methode von Shi und Tomasi arbeitet. Beide Varianten haben jeodch keine guten Ergebnisse erzielt, da oft zu wenig oder zu viele Ecken nicht gefunden wurden. Eine andere Idee war die Hough-Transformation zu nutzen, um zu prüfen, ob es vier dominante Linien im Bild gibt. Doch auch hier gab es Probleme, da die Linienlängen nicht zuverlässig ausgelesen werden konnten und so die Eckpunkte nicht detektiert werden konnten. Also haben wir eine andere Methodik implementiert, welche auf keinem uns bekannten wissenschaftlichen Ansatz beruht. Diese stellen wir hier nun grob vor (Eine detailliertere Beschreibung würde den Rahmen dieses Kapitels sprengen):

Zunächst wird die Umrandung der Form extrahiert. Die Koordinaten der Kantenpixel werden nun in Polarkoordinaten umgewandelt, dabei dient der Mittelpunkt des Objektes als kartesische 0/0-Koordinate. Wir erhalten so also eine Menge von (θ,ρ)-Tupeln.

E wird eine Funktion aufgestellt, die uns für gegebenes θ das zugehörige ρ liefert. Abb2 zeigt eine typische Funktion für ein unregelmäßiges Viereck. Hier sieht man ein für Vierecke typisches Muster: Vier auffällige „Täler“, welche durch auffällige Ecken in der Funktion getrennt sind.

Diese “Ecken“ zeigen sich als stark auffällige Minima in der zweiten Ableitung. Bevor diese jedoch gebildet werden kann, muss die Funktion geglättet werden, da aufgrund der Rastereigenschaft von Bildern viele Fehler vorhanden sein können. Dazu wird die Funktion neu mit gleichmäßigen Abständen abgetastet (Zwischenwerte werden linear interpoliert) und die so entstehenden Werte werden mit einem Savitzky-Golay-Filter geglättet (dieser verschiebt Extremwerte nicht so stark wie ein normaler Mittelwertfilter).

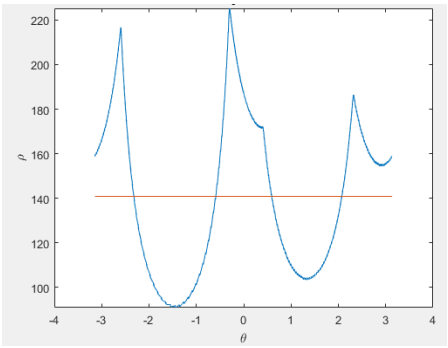


Abbildung 2

Danach wird die zweite diskrete Ableitung gebildet (welche ebenfalls geglättet werden muss). In dieser wird nach Minima gesucht, in dem ein Schwellwert gesetzt wird (gelbe Linie in Abb3). Betrachtet man nur die Teile unterhalb des Schwellwertes zerfällt die Funktion in mehrere Teilfunktionen. Von diesen Teilfunktionen wird nun jeweils das globale Minimum bestimmt. Als Schwellwert wird ein 3.2-tel des Gesamt-Minimums genommen (aber minimal -200). Dieser Wert wurde im Wesentlichen durch Ausprobieren festgelegt und für gut befunden.

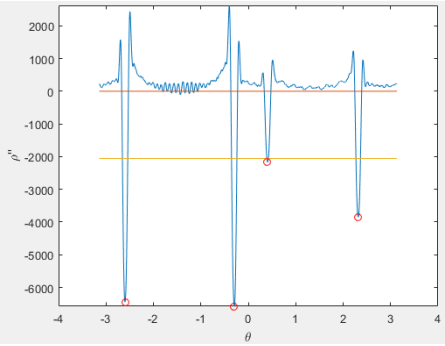


Abbildung 3

Erhalten wir durch dieses Verfahren mehr oder weniger als vier Minima handelt es sich um kein Viereck und der Algorithmus kann abgebrochen werden. Die vier θ-Werte werden dann in die Originalfunktion eingesetzt und in kartesische Koordinaten zurückumgewandelt. Um sicher zu gehen, wird zuletzt noch geprüft ob die Fläche, die die vier gefundenen Punkte einschließen, zumindest 70% der Fläche des Objektes enthält[1].

Der Algorithmus funktioniert in den meisten Fällen ausgesprochen gut. Auch unsauber ausgeschnittene Rechtecke werden bis zu einem gewissen Grad noch erkannt. Ein paar Einzelfälle, in denen falsche Objekte als Vierecke erkannt werden, gibt es zwar, aber diese werden dann meist von den anderen Prüfungen in unserem Algorithmus verworfen.

3.4 Histogrammüberprüfung

Die Histogrammüberprüfung vergleicht ein vorgegebenes Bild und seine Histogrammwerte mit vordefinierten Werten.

3.4.1 Vorüberlegungen

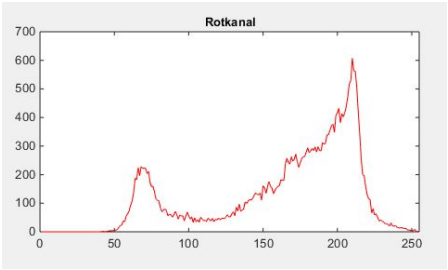


Abbildung 4

Grundgedanke war es in den Histogrammen gewisse Ähnlichkeiten zu erkennen und es besteht natürlich die Frage ob Kennzeichen in verschiedensten Lichtbedingungen und Perspektiven eindeutigen Ähnlichkeiten aufweisen. Nach Analyse von diversten Kennzeichen fielen durchaus Ähnlichkeiten auf, die wichtigste zu nennende Ähnlichkeit wären zwei Peaks die in allen Kanälen vorfinden lassen. Diese Peaks haben auch sehr oft die selbe Form und auch einen sehr ähnlichen Abstand zu einander. Im der nebenstehenden Figur findet sich ein solches Beispiel vor. Zum Vergleich werden verschiedenste vorher berechnete Histogramme aus einem Datensatz herangezogen

3.4.2 Methodiken für die Ähnlichkeit

Ein Ansatz war es das Vergleichshistogramms in der Höhe zu dehnen, wie man es zum Beispiel von Wettervorhersagen kennt. Jedoch wäre dieser Ansatz durchaus zielführend gewesen, denn dabei wird das Vergleichshistogramms in zwei neue Histogramme zerlegt.

[1] Nur 70%, da durch die Glättung die Eckpunkte um ein paar Pixel falsch liegen können, und so einige Prozent an Fläche entfallen

Ein gewisser Prozentsatz dehnt das Histogramm nach oben und nach unten. Ein Vergleichshistogramm das ähnlich ist kann dazwischen liegen.
Ein anderer Ansatz war im Bereich der Statistik vorzufinden, zum Beispiel ergab sich hierbei die Möglichkeit den Kolmogorov-Smirnov Test anzuwenden, der Auskunft darüber gibt inwiefern sich zwei Datensätze unterscheiden.
Der letzte Ansatz welcher zum Erfolg führte war die Methode pdist2 und über euklidische Distanzen.

3.4.3 Bestehende Probleme

Trotz der Vielzahl an Methodiken war es nicht einwandfrei Möglich einen exakten Test durchzuführen, zum Beispiel werden wirklich perfekte Kennzeichen nicht erkannt. Diese Kennzeichen haben ein perfektes Verhältnis von Schwarz/Weiss und ähneln von der Optik her als wären sie aus Kunststoff.

3.5 Geometrische Transformation

Die Geometische Transformation berechnet für jedes Pixel des Zielbildes, die Position dieses Pixels im Quellbild, indem das Pixel mit einer Transformationsmatrix ins Quellbild abgebildet wird, um dort den Farbwert für das Zielpixel mittels Interpolation zu berechnen. Unsere Quellen sind [5] und [1].

3.5.1 Transformationsmatrix

Um das Kennzeichen, das durch seine 4 Eckpunkte eindeutig definiert ist, gerade zu transformieren, benötigt man eine projektive Abbildung bzw. auch Vierpunkt-Abbildung genannt. Diese ist eine verallgemeinernde Form der affinen Abbildung (Dreipunkt-Abbildung), bei der zwar Linien und Kurven n-ter Ordnung erhalten bleiben, jedoch bleiben im Gegensatz zur affinen Transformation die Geraden nicht parallel und Abstandsverhältnisse ändern sich auch in der Regel.

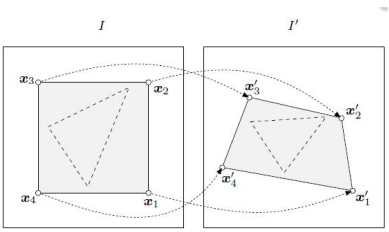


Abbildung 5: Quelle: [1]

$$\begin{pmatrix} \widehat{x}' \\ \widehat{y}' \\ h' \end{pmatrix} = \begin{pmatrix} h'x' \\ h'y' \\ h' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3}$$

Die projektive Abbildung wird definiert, indem bei der Transformationsmatrix der affinen Abbildung 2 Parameter a31 und a32 hinzugefügt werden. Da sich die Gleichung, mit der man die Parameter a11 bis a32 errechnet, als sehr komplex herausstellt, sieht man sich am besten zuerst einen verallgemeinerten Fall an, der der Kennzeichentransformation genügt, da das Zielbild ein Rechteck sein soll.

Man schlägt für jedes Pixel des Zielbildes den Farbwert im Quellbild nach bzw. interpoliert zwischen verschieden Pixels, falls ein Punkt zwischen Pixeln getroffen wird. Wenn man die Position jedes Pixels im Zielbild durch die Höhe/Breite des Zielbildes dividiert, kann man von einem Quadrat als Zielbild ausgehen (S1 = Zielbild, Q = Quellbild). Siehe Abb6

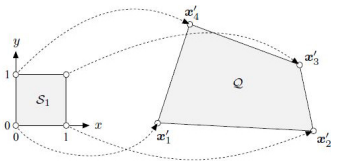


Abbildung 6: Quelle: [1]

Nach diversen Umformungen erhält man damit folgende Formeln für a11 bis a32:

$$
\begin{aligned}
a_{31} &= \frac{(x'_1 - x'_2 + x'_3 - x'_4)\cdot(y'_4 - y'_3) - (y'_1 - y'_2 + y'_3 - y'_4)\cdot(x'_4 - x'_3)}{(x'_2 - x'_3)\cdot(y'_4 - y'_3) - (x'_4 - x'_3)\cdot(y'_2 - y'_3)}) \\
a_{32} &= \frac{(y'_1 - y'_2 + y'_3 - y'_4)\cdot(x'_2 - x'_3) - (x'_1 - x'_2 + x'_3 - x'_4)\cdot(y'_2 - y'_3)}{(x'_2 - x'_3)\cdot(y'_4 - y'_3) - (x'_4 - x'_3)\cdot(y'_2 - y'_3)}) \\
a_{11} &= x'_2 - x'_1 + a_{31}\cdot x'_2 \qquad a_{12} = x'_4 - x'_1 + a_{32}\cdot x'_4 \qquad a_{13} = x'_1 \\
a_{21} &= y'_2 - y'_1 + a_{31}\cdot y'_2 \qquad a_{22} = y'_4 - y'_1 + a_{32}\cdot y'_4 \qquad a_{23} = y'_1
\end{aligned}
\tag{4}
$$

4 Implementierung

4.1 Vorverarbeitung

- plateDetection.m

Bevor mit der eigentlichen Verarbeitung des Bildes begonnen werden kann, muss das Bild zuerst einer Vorverarbeitung unterzogen werden, in der das Bild auf eine passende Größe von maximal 1200 Pixel in der Breite reduziert und in ein Grauwertbild überführt wird. Der Wertebereich wird auf double- Werte im Intervall [0,1] verschoben.

4.2 Canny Algorithmus

- cannyalgorithmn.m
- dilate.m
- imagefilter.m
- gaussian.m
- nonMaximumSuppression.m

In dem nun erhaltenen Grauwertbild sind Kanten markante Stellen, die durch große Veränderung der Intensität zwischen zwei benachbarten Pixel, gekennzeichnet sind. Dies kann als Unstetigkeit der Grauwertefunktion des Ausgangsbildes aufgefasst werden. Um nun Kanten zu finden, wird der Canny Algorithmus verwendet. Dieser ist in mehrere Schritte unterteilt:

Abbildung 7

Abbildung 8

4.2.1 Rauschunterdrückung

Für die Rauschunterdrückung wird der Gauß-Filter verwendet. Bei diesem linearen Filter entsprechen die einzelnen Werte des Filters der Dichtefunktion der zweidimensionalen Gaußschen Glockenkurve (5) mit dem Ursprung der Glockenkurve im Mittelpunkt der Matrix.

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{s\sigma^2}} \tag{5}$$

Als Varianz σ wird in der aktuellen Implementierung 0.5 gewählt. Der Filter ist eine 3x3 große Matrix. Daher entsprechen die Werte für x und y den Werten -1,0,1. Der Kernel des Gauß-Filters wird in der Gleichung (6) gezeigt.

$$B = \begin{bmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & 0.6193 & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{bmatrix} \tag{6}$$

Bei der Programmierung wurden mehrere verschiedene Filtergrößen und Varianzen getestet. Zu große Filter führten meist zwar zu weniger Rauschen, allerdings wurde dadurch auch die Anzahl der gefundenen Kennzeichen vermindert. Schlussendlich entschieden wir uns für einen kleinen Filter. Dadurch entstand unter anderem zwar eine hohe Laufzeit des Algorithmus, jedoch auch eine große Trefferquote bei der Erkennung der Kennzeichen.

Filter Damit die Anzahl der Durchläufe einer Schleife bei der Faltung des Bildes mit dem Filter möglichst gering gehalten werden, wird die Faltung nicht für jedes Pixel

einzeln berechnet, sondern es wird die gesamte Matrix des Bildes mit jedem Wert des Filters einzeln multipliziert. Anschließend werden die daraus gewonnenen Matrizen, je nach Position des Filterwertes verschoben. Als Randbehandlung werden die noch nicht definierten Pixel des Bildes mit Nullen aufgefüllt.

4.2.2 Kantendetektion

Zur eigentlichen Detektion der Kanten wird in dieser Implementierung ein 3x5 Sobel-Kernel verwendet. Dieser berechnet die erste Ableitung der Intensitätsfunktion des Bildes. Außerdem wird durch die Ungleichgewichtung der Kanten, anders als beim Prewitt Operator, zusätzlich noch orthogonal zur Ableitungsrichtung geglättet. Da der Sobel Filter nicht richtungsunabhängig ist, wird hier nacheinander in x – Richtung und in y –Richtung gefiltert. Dabei wird die Ableitung nach x G_x und die Ableitung nach y G_y des Grauwertbildes A wie folgt berechnet.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -4 & 0 & 4 \\ -6 & 0 & 6 \\ -4 & 0 & 4 \\ -1 & 0 & 1 \end{bmatrix} * A \tag{7}$$

$$G_y = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & -4 & -6 & -4 & -1 \end{bmatrix} * A \tag{8}$$

Die Faltung der Filter mit der Bildmatrix wird im Unterpunkt „Filter“ ausführlich erklärt.

Kantenrichtung Um die Richtung θ der Kanten zu berechnen wird die Matlab Funktion atan verwendet. Diese entspricht der Inversen Tangensfunktion. Diese Gleichung entspricht Gleichung (1) . Da jedes Pixel genau 8 Nachbarn hat, müssen die Ergebnisse der inversen Tangensfunktion auf die Winkel 0 rad, $frac14$ * pi rad, $frac12$ * pi rad und $frac34$ * pi rad gerundet werden.

Absolute Kantenstärke Man berechnet ein in beide Richtungen gefiltertes Bild mit absoluter Kantenstärke G wie in Gleichung (2) beschrieben.

4.2.3 Non-maximum suppression

- nonMaximumSuppression.m

Der Methode nonMaximumSuppression wird das Kantenbild G und die Matrix THETA, die die Orientierung jedes Pixels enthält, übergeben. Wie auch bei der Filterung wird nicht jedes Pixel des Bildes einzeln überprüft, ob es ein Maximum repräsentiert oder nicht, sondern es wird die ganze Matrix 8 mal, für jeden Nachbarn eines Pixels (also

nach rechts oben, oben, links oben, rechts, links, rechts unten, unten, links unten), verschoben. Jede dieser Matrizen wird von G subtrahiert. Aus den daraus resultierenden Matrizen wird jeweils ein Binärbild erstellt, wobei positive Werte mit 1(true) und negative Werte mit 0(false) bewertet werden. Nun werden immer zwei invers verschobene Matrizen logisch UND verbunden. Dadurch erhält man 4 logische Matrizen, die jeweils überprüfen ob ein Pixel, im Vergleich zu zwei benachbarten Pixeln, die jeweils auf einer Geraden durch den aktuellen Pixel liegen, einem lokalen Maximum entspricht. Nun wird eine neue Matrix erzeugt die speichert, welche Pixel orthogonal zu ihrer Kantenrichtung ein lokales Maximum bilden. Das erhaltene Binärbild wird dann als Maske über das ursprüngliche Kantenbild gelegt. So bleiben die Intensitätswerte erhalten.

4.2.4 Schwellwert mit Hysteresis

- cannyalgorithmn.m

Nun werden nur noch zwei Schwellwerte verwendet um Kanten mit zu geringen Intensitätswerten herauszufiltern. Dafür werden zwei Werte vom Typ double angenommen. Diese Werte wurden mittels empirischer Testung angenommen um möglichst gute Ergebnisse zu erzielen. Zuerst wird eine Maske erstellt, die alle Pixel sucht, die Intensitätswerte über dem höheren Schwellwert haben. Danach wird mittels Dilation diese Maske erweitert. Diese neue Maske wird über das Bild mit den Intensitätswerten gelegt. Aus dem Ergebnisbild werden abschließend nur noch die Werte auf Null gesetzt, die unter dem niedrigeren Schwellwert liegen

4.2.5 Löcher schließen mit Hilfe der Dilation

- closeOnePixelHoles.m

Da beim Canny oftmals die Kanten des Kennzeichens wegen einem einzigen Pixel nicht geschlossen waren, war es notwendig eine Spezielle Art von Dilation einzubauen, die diese Löcher schließt. Die Häufigste Art von Löcher war, dass sich links und rechts von Loch eine Kante befand. Dazu invertiert man das Bild für den mittleren Pixel, verschiebt das Bild nach rechts um den linken Pixel zu erhalten bzw. nach links für rechten Pixel und verknüpft die 3 Bilder mit logisch UND. Das Ergebnis verknüpft man dann noch mit logisch ODER um die neuen Punkte einzufügen. Selbiges wiederholt man für links unten - rechts, links - rechts unten und oben - unten, jeweils vom Ursprungsbild des Cannys ausgehend, um die weiteren Arten von Löchern zu schließen. Bei manchen Kennzeichen mit geringer Auflösung hatte dies zur Folge, dass die Buchstaben sich mit dem Kennzeichenrand verbunden, aber im Allgemeinen verbesserte dies die Erkennung der Kennzeichen.

4.3 Flächen finden

- findareas.m

- findnextBlackPixel.m

In dem erzeugten Kantenbild werden nun mit Hilfe der Funktion findareas die geschlossenen Flächen gesucht. Hierbei wird zuerst die Funktion findnextBlackPixel aufgerufen. Diese gibt die Position des Seedpoints zurück. Der Seedpoint ist der Ausgangspunkt des Floodfill Algorithmus. In der aktuellen Implementierung wird die, von Matlab zur Verfügung gestellte Funktion imfill verwendet um die neuen Flächen zu berechnen und zu färben. Alle auf diese weise gefunden Flächen werden in einer Matrix abgespiechert. Zusätzlich wird immer eine Version der gleichen Matrix gespeichert, in der die letzte Füllung einer Teilfläche noch nicht statt gefunden hat. Diese beiden Matrizen werden dann subtrahiert. So erhält man nur die, durch den letzten Vorgang gefärbten Pixel. Wenn die neue Fläche mindestens 0.5 Prozent und maximal 50 Prozent der Gesamtfläche des Bildes bedeckt, wird die Matrix in einer neuen Ebene eines Arrays gespeichert. Dieser Vorgang wird so lange wiederholt, bis kein neuer Seedpoint mehr gefunden werden kann und alle Flächen gefüllt wurden. Das Array mit den ausgewählten Flächen wird zurückgegeben.

4.4 Auf Löcher in der Fläche prüfen

Da die Buchstaben in den Kennzeichen auch geschlossene Flächen sein sollten, wird im ersten Schritt überprüft, ob die gefundenen Flächen auch Löcher enthalten. Dazu werden alle Löcher der gefundenen Matrizen gefüllt. Hat sich durch diese Operation kein Pixel verändert, wird diese Fläche nicht mehr als Kennzeichen in Betracht gezogen.

4.5 Viereckprüfung und Eckendetektion

4.5.1 DiscreteZyclicFunction.m

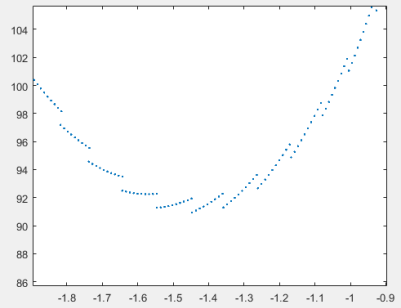
Da dieser Teil des Algorithmus viel mit Funktionen arbeitet, wurde eine eigene Klasse zum Darstellen einer diskreten, zyklischen Funktion erstellt. Die Klasse enthält zwei Attribute: X und Y. Beides sind Arrays der gleichen Länge. X ist dabei aufsteigend sortiert. Y(i) enthält den zugehörigen Y-Wert zu X(i).

Konstruktor Der Konstruktor der Klasse erhält ein X- und ein Y-Array. X muss bei der Übergabe noch nicht sortiert sein. Wichtig ist jedoch, dass die beiden Arrays gleich lang sind, sonst wird ein Fehler geworfen. Außerdem darf ein X-Wert nur dann doppelt vorkommen, wenn der zugehörige Y-Wert in beiden Fällen gleich ist, ansonsten wird ebenfalls ein Fehler geworfen. Doppelte X-Y-Paare werden im Konstruktor eliminiert.[2]

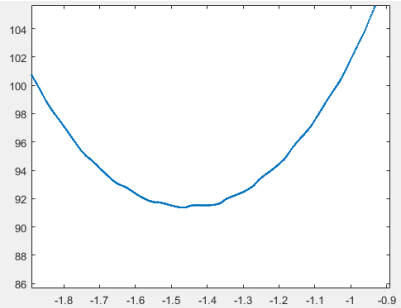
Funktion equidistant() Diese Funktion tastet die linear interpolierte Funktion in gleichmäßigen Schritten ab und gibt das Ergebnis als neue Funktion zurück. Die ursprüngliche Funktion bleibt dabei unverändert. Bezeichnet n die Anzahl der Elemente in der Originalfunktion, besitzt die neue Funktion $2n - 1$ Abtastpunkte.

[2] In der Praxis kam es eigentlich nie vor, dass ein θ-Wert mehrmals vorkam

Funktion sgSmooth(span) Diese Funktion glättet die Funktion mithilfe des Savitzky-Golay-Filters. Die geglättete Funktion wird zurückgegeben, das ursprüngliche Objekt bleibt unverändert. Voraussetzung hierfür ist, dass die Funktion äquidistant abgetastet ist, ansonsten kann sie zu falschen Resultaten führen. Die tatsächliche Filtergröße berechnet sich durch $span/d$, wobei d der Abstand zwischen zwei Abtastpunkten auf der X-Achse ist. Vor dem Anwenden der sgolayfilt-Funktion wird das Y-Array links und rechts um einige Elemente von der anderen Seite aufgestockt, um eine korrekte Randbehandlung zu gewährleisten (Die Funktion ist zyklisch). Diese Elemente werden nach dem Glätten wieder entfernt (Gleiches Prinzip wie es von Bildfiltern bekannt ist).



(a) Originalfunktion



(b) Nach equidistant() und smooth(0.2)

Funktion getXNeighbour(x) Diese Funktion erhält eine X-Koordinate. Existiert für genau diese X-Koordinate ein Abtastpunkt, wird die Koordinate selbst wieder zurückgegeben. Ist die X-Koordinate kleiner als alle gespeicherten X-Koordinaten, wird die kleinste X-Koordinate zurückgegeben. Ist sie größer, wird die größte X-Koordinate zurückgegeben. In allen anderen Fällen liegt x zwischen zwei X-Abtastpunkten. Es wird dann diejenige der beiden X-Koordinaten zurückgegeben, welche den größeren zugehörigen Y-Wert besitzt. Dabei verwendet diese Methode binäre Suche, um die X-Werte, die in Frage kommen, zu finden.

Es ist auch möglich, ein Array von x-Werten zu übergeben. Dann wird die Methode rekursiv für jeden Wert aufgerufen und liefert am Ende ein Array aus Rückgabewerten für jedes übergebene x.

Diese Funktion wird im Algorithmus genutzt, um von einem θ-Wert aus der geglätteten Ableitung auf einen Wert aus der Originalfunktion zu schließen. Aufgrund der äquidistanten Abtastung muss der Wert nämlich nicht existieren. Die Wahl, den größeren Wert zu nehmen, hängt damit zusammen, dass bei Funktionsabschnitten wie dem in Abb10 die Punkte beim Glätten in Richtung des starken Falls verschoben werden.

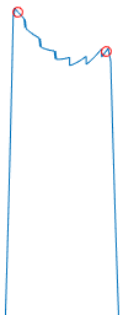


Abbildung 10

getYLinearInterpolated(x) Diese Funktion liefert den zu x zugehörigen Y-Wert zurück, falls x im X-Array enthalten ist. Ist x größer oder kleiner aller gespeicherten X-Werte wird 0

zurückgegeben. Ansonsten liegt x zwischen zwei X-Werten. Diese werden mittels binärer Suche gefunden. Der Ergebniswert der Funktion wird dann zwischen den beiden Y-Werten linear interpoliert.

Auch in dieser Funktion ist es möglich, ein Array von x-Werten zu übergeben, um ein Array von zugehörigen Y-Werten zu erhalten.

getDerivative(nr) Liefert die nr-te Ableitung der Funktion als neues Funktionsobjekt. Voraussetzung ist, dass die Funktion äquidistant ist. Die Formel für die diskrete Ableitung lautet:

$$Y'(i) = \frac{Y(i+1) - Y(i-1)}{2d} \tag{9}$$

d bezeichnet hierbei die Distanz zwischen zwei Punkten auf der X-Achse. Dieser Wert ist für alle Werte definiert, da die Funktion zyklisch ist ($\forall z \in \mathbb{Z} : Y(i+zn) = Y(i)$)[3]. Um dies effizient zu berechnen, werden zwei Vektoren erstellt, die die Indexe von 1 bis n (n = Anzahl der abgetasteten Werte) enthalten. Ersterer ist (zyklisch) nach links geshiftet, zweiterer nach rechts. Durch Übergabe dieser Indexmengen an das Array können die Werte schnell subtrahiert werden.

getLowMinima(treshold) Diese Funktion sucht auf die in der Methodik beschriebene Art und Weise nach Minima. Es werden werden alle Y-Werte herausgefiltert, welche unter der Schwellwert-Linie liegen. Jeder Funktionsabschnitt wird nach einem Minimum durchsucht und dieses zum Ergebnis hinzugefügt. Das Ergebnis enthält X- und Y-Werte der Minima.

Der tatsächlich verwendete Schwellwert berechnet sich hierbei durch $\min(Y) * treshold$. Er muss jedoch mindestens -200 sein. Nimmt man zum Beispiel das Bild eines Kreises, so ist die zweite Ableitung eigentlich nur eine gerade Linie. Da aber auch diese nicht exakt ist sondern leichte Schwankungen hat, gäbe es ohne diese Einschränkung ebenfalls Minima, welche detektiert werden würden.

Eine einfache Suche nach lokalen Minima wäre nicht zielführend, da die Funktion trotz Glättung oft noch leichte Zacken enthalten kann.

4.5.2 checkQuad.m

Diese Datei enthält die Funktion checkQuad, welche mit Hilfe der DiscreteZyclicFunction-Klasse den eigentlichen Algorithmus implementiert. Übergeben wird ein Binärbild, welches genau eine gefüllte Fläche enthält. Handelt es sich bei der Fläche um ein Viereck, werden die vier Punkte zurückgegeben, ansonsten wird false zurückgegeben.

Abbildung 11

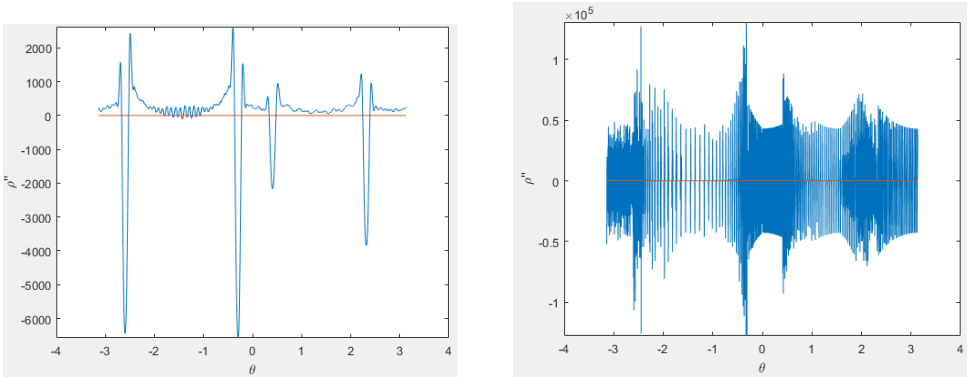
[3]Diese Definition der Zyklizität ist eigentlich etwas versimpelt. Eigentlich gilt $\rho(\theta) = \rho(\theta \pm 2\pi)$, da unsere Funktion immer von $-\pi$ bis $+\pi$ geht. Es kann also am Rand zu leichten Fehlern kommen, die sich praktisch aber nicht auswirken.

Zunächst wird das Bild vorverarbeitet. Es wird ein Closing durchgeführt, um dünne Löcher wie in Abb11 zu schließen. Als Strukturelement wird einfach ein 3*3-Viereck genommen. Danach werden alle Löcher mittels der Matlab-Funktion imfill gefüllt. Die Kantendetektion wird mit dem Canny-Detektor von Matlab durchgeführt.

Als nächstes wird der Mittelwert aller Umrisspixel berechnet. Liegt dieser auf einem Pixel, der im gefüllten Binärbild schwarz ist, wird abgebrochen, da der Mittelpunkt in der Form liegen muss.

Danach wird von allen Punkten der Mittelwert abgezogen. Die neuen Koordinaten werden dann mittels Matlab-Funktion cart2pol in Polarkoordinaten umgewandelt.

Als nächstes wird für diese Werte ein DiscreteZyclicFunction-Objekt erzeugt. Dieses wird dann äquidistant abgetastet und geglättet (Filtergröße 0.2). Dann wird die 1. Ableitung gebildet und diese wieder geglättet (Filtergröße 0.1). Diese wird wieder abgeleitet. Diese 2. Ableitung wird dann geglättet, wobei die Filtergröße von der Anzahl der Pixel der Umrandung abhängig ist (Bei kleinen Objekten muss stärker geglättet werden als bei großen).



(a) Geglättete 2. Ableitung

(b) 2. Ableitung ohne Glättungen

Abbildung 12

Als nächstes wird getLowMinima mit einem fixen Treshold von 1/3.2 aufgerufen. Ist der Betrag des Schwellwert zu hoch, kann es sein, dass Eckpunkte nicht detektiert werden. Ist er dagegen zu niedrig, kann es passieren, dass andere Punkte für Eckpunkte gehalten werden. 1/3.2 hat sich im Testen als guter Wert herausgestellt.

Liefert die Funktion nicht genau vier Punkte zurück, wird abgebrochen. Ansonsten werden die Punkte in die Originalfunktion eingesetzt. Hierbei muss zunächst getXNeighbour aufgerufen werden um θ-Koordinaten zu erhalten, die auch in der Funktion vorkommen. Die Koordinaten werden dann in kartesische Koordinaten zurück umgewandelt und der Mittelwert wird wieder dazuaddiert, um die tatsächlichen Koordinaten zu erhalten.

Als letztes wird mittels der Matlab-Funktion inpolygon geprüft, wieviel % des Objektes die gefundenen Ecken einschließen. Sind es weniger als 70% wird abgebrochen. Ansonsten werden die gefundenen Koordinaten zurückgegeben.

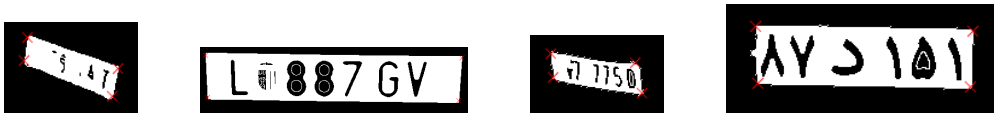


Abbildung 13: Verschiedene Nummernschilder mit den detektierten Ecken

4.6 Histogrammüberprüfung

4.6.1 histogrammer.m

Gegeben ist ein Farbbild welches eingelesen wird. Dieses Farbbild wird in seine Bestandteile zerlegt um genügend Informationen aus dem Bild zu erhalten für die entsprechende Weiterverarbeitung. Die Histogrammgenerierung liefert jeweils die Rot, Grün und Blau Verteilung in einer Datenstruktur und kann sie entsprechend auch optisch wiedergeben. Zusätzlich wird das Bild auch als Schwarzweiß-Histogramm angezeigt und mit entsprechender Equalisierung. Letzteres wurde jedoch im weiteren Verlauf der Arbeit nicht benutzt. Histogrammer.m wurde im folgenden auch dazu Verwendet um die verschiedensten RGB Histogramme für das Matching zu generieren. Diese Ergebnisse werden in Dateien abgelegt.

4.6.2 MakeHistograms.m

Diese Funktion bezieht vorher generierte Matching-Datein der Funktion histogrammer.m, wie vorhin angemerkt, werden diese in Dateien abgelegt und in dieser Funktion eingelesen. Die Anzahl der Dateien variiert natürlich anhand der Anzahl der Testcases. Das einlesen von fertig generierten Werten ist die performanteste Methode, da somit nicht unnötig Zeit verloren wird während der Laufzeit. In der MakeHistograms geschieht das schlussendliche Matching. Die zahlreichen Testwerte werden anhand ihrer R/G/B Bestandteile eingelesen und in entsprechenden Variablen abgelegt. Zuvor wird aber das Originalbild eingelesen. Dieses besteht aus zwei Komponenten. Dem Originalbild und dem Binärbild welches angibt wo sich dieses Kennzeichen genau befindet, anhand dessen wird es zuvor subtrahiert und danach entsprechend beschnitten. Am Ende des Vorgangs werden alle eingelesenen Werte der Methode compareValues() übergeben. Der zurückgegebene Wert (im folgenden genauer erläutert) wird in Variablen abgelegt und schlussendlich in eine Gesamtsumme umgerechnet. Anhand der Summe wird ein Prozentsatz errechnet inwiefern das Matching mit unseren bestehenden Kennzeichen erfolgreich war. Natürlich ist es auch gar nicht möglich ein 100 Prozent Matching zu erhalten, da einerseits perfekte Kennzeichen, aber auch eher schlechte Kennzeichen vorliegen um ein wohlgeformtes Matching zu erhalten. Im Normalfall ergab ein Wert über 30 Prozent auf jedenfall ein Kennzeichen.

4.6.3 compareValues.m

Hierbei handelt es sich um eine Vergleichfunktion. Die Übergabeparameter sind 6 Datenstrukturen mit jeweils den RGB Werten unseren Originalbildes und jenen eines Ver-

gleichsbildes aus den zuvor generierten Datensätzen. Ziel ist es die Ähnlichkeit zu bestimmen. Wie im Methodikteil erläutert, gibt es hierzu eine Vielzahl von Verfahren. Die Effizienz ließ jedoch bei fast allen Verfahren stark nach sobald die Histogramme niedriger Auflösung waren. Schlussendlich kam man zur Entscheidung sich auf die Variante über pdist2 zu verlassen, welche von allen angewandten Methodiken die genaueste Trefferquote erzeugte. Hierbei werden jeweils zwei Histogrammteile gleichen Farbtyps miteinander insofern verglichen, dass die Euklidische Distanz der jeweiligen Werte berechnet wird. Über pdist2 kann somit schlussendlich über eine Ähnlichkeit ausgesagt werden. In fast allen Testfällen war eine Ähnlichkeit dann gegeben wenn sich der Rückgabewert unter 1000 befand. Somit wird nur noch abgefragt ob für jeweils R,G und B der Matchingwert unter 1000 liegt und entsprechend ein Wert von 0-3 zurückgegeben.

4.6.4 Fremdquellen

Bein beschneiden des Originalbildes wird auf eine vorgefertigte Cropfunktion zurückgegriffen. Der Autor dieser Funktion ist Guilherme Coco Beltramini (guicocogmail.com)[4]

4.7 Geometrische Transformation

4.7.1 Punktrotation

- rotatePoints.m

Da für die geometrische Transformation die Reihenfolge der Eckpunkt wichtig ist, muss man dafür sorgen, dass die Eckpunkte in die richtige Reihenfolge gebracht werden. Als richtige Reihenfolge definieren wird, dass der links untere Eckpunkt der Erste ist und numerieren dann gegen den Uhrzeigersinn fortlaufend. Da es sich bei diesem Fall um Kennzeichen handelt, kann man von 2 Voraussetzungen ausgehen: Die linke und rechte Kanten sind kürzer als die obere und untere Kante und der Mittelpunkt von der linken Kante ist links vom Mittelpunkt der rechten Kante. Da setzt natürlich voraus, dass die Kennzeichen nicht auf dem Kopf stehen dürfen. Die Vorgehensweise ist, dass man die kürzen Kanten sucht und deren Endpunkte und die Kanten untereinander in die richtige Reihenfolge bringt.

Die kürzere Kante des Rechteckes sucht man, indem man mit dem ersten Eckpunkt beginnt und die euklidische Distanz zu allen anderen Eckpunkten berechnen. Die kürzeste Strecke verwendet man als linke Seite des Rechtecks. Die Strecke zwischen den verbleibenen zwei Eckpunkten ist dann automatisch die rechte Seite. Man vergleicht bei beiden Seiten die y-Werte der Eckpunkte und vertauschen die Eckpunkte gegebenfalls. Danach errechnet man sich den x-Wert des Mittelpunktes beider Strecken und vergleicht sie. Falls die linke Seite einen höheren x-Wert als die rechte Seite hat, vertauscht man sie. Nun muss man nur noch die Punkte in der richtigen Reihenfolge auslesen und zur Berechnung der Transformationsmatrix übergeben.

[4] http://www.mathworks.com/matlabcentral/fileexchange/42035-save-figure-to-file--choose-the-dimensions-and-crop-figure/content/crop_img.m

4.7.2 Transformationsmatrix

- calcGeometricTransformationMatrix.m

Die notwenigen Formeln hierzu wurden bereits im Kapitel Methodik vorgestellt, sodass hierzu nur noch zu sagen ist, dass die von der Punktrotation erhalten Eckpunkte in die Formeln eingesetzt werden und die Parameter a11 bis a32 und 1 als 3x3-Matrix zurückgegeben werden.

4.7.3 Transformation

- geometricTransformation.m

Jeder Pixel des Zielbildes wird durch die Breite/Höhe des Bildes dividiert und die erhaltenen x- und y-Koordinaten zusammen der homogenen Koordinate 1 mit der Transformationsmatrix multipliziert. Für den erhaltenen Punkt wird dann noch mittels Interpolation dessen Farbwert anhand der nähesten 4 Pixel berechnet. Dieser Farbwert wird dann dem Pixel im Zielbild mit Orginalbreite und –höhe zugewiesen. Für das um den Eurobereich nach links vergrößerte Nummernschild, wird von einem um einen Offset nach links verschobenen Bild ausgegangen und durch Breite-Offset dividiert. Dadurch werden auch Pixel links der 4 Eckpunkte ins Zielbild abgebildet. Gespeichert werden die Farbwerte in einem Zielbild ursprünglicher Breite ohne Offset.

4.7.4 Interpolation

- interpolation.m

Um ein etwas schöneres Ergebnis zu erhalten als einfach den nähersten Farbwert aus dem Originalbild zu nehmen, wird eine bilineare Interpolation durchgeführt. Sofern die gewünschte Koordinaten nicht ganzzahlig sind, liegt der Punkt zwischen vier Pixeln. Zunächst werden die Farben der beiden oberen und unteren Pixel jeweils entlang der X-Achse linear interpoliert. Die beiden so erhaltenen Farbwerte werden dann entlang der Y-Achse linear interpoliert.

Das Verfahren kann genauso genutzt werden, wenn eine oder beide Koordinaten ganzzahlig sind, das Ergebnis ist dennoch korrekt. Als Nachbarpixel werden dann einfach die rechten und unteren Pixel genutzt. Besondere Beachtung muss jedoch dem Fall geschenkt werden, dass der angegebene Punkt genau am rechten oder unteren Rand des Bildes liegt. Da es dann keine äußeren Nachbarn mehr gibt, wird als Farbwert der äußeren Nachbarpixel dann einfach schwarz genommen, dieser wird dann aber sowieso weginterpoliert.

4.7.5 Eurozeichenbereich-Farbtest

- euroTest.m

Der Eurozeichenbereich-Farbtest gehört eigentlich nicht zur geometischen Transformation, hat sich aber im Laufe des Projektes als eine notwendige Erweiterung herausgestellt, denn wenn der Eurozeichenbereich noch nicht im transformierten Kennzeichen enthalten ist, muss die Transformation mit nach links erweiterten x-Werten wiederholt werden. Dazu werden die linken 8% des Bildes getestet, wo sich der blaue Eurozeichenbereich befinden sollte. Davon wird der mittlere Bereich genommen mit 50% der Höhe und Breite. Von diesem Bereich wird der Durchschnittswert für alle 3 Farbkanäle berechnet, indem je Farbkanal alle Pixel aufsummiert und durch die Anzahl dividiert werden. Nun werden die 3 Durchschnittswerte verglichen. Der Durchschnittswert von Blau muss signifikant höher sein als der Durchschnittswert von Rot und Grün. Ist dies der Fall, ist das Eurozeichen bereits im Bild enthalten, anderenfalls muss der Bereich nach links vergrößert werden. Vergrößert wird der Bereich, indem die x-Werte des Zielbildes um 8% der Gesamtbreite im negativen Bereich beginnen lässt, um somit die gefundenen Eckpunkte des Nummernschildes entsprechend der Verzerrung nach links um den Eurozeichenbereich zu erweitern.

5 Evaluierung

5.1 Datensatz

Der Datensatz für das Testen der Kennzeichen-Erkennung besteht aus 43 Bildern auf denen zum Großteil Autos in verschiedener Größe, Marke und Position vorhanden sind. Des Weiteren ist der Datensatz auch so aufgebaut, dass sich auf den Bildern nicht nur einzelne Fahrzeuge befinden, sondern auch mehrere Fahrzeuge auf einem Bild. Die Lichtqualität wurde als gut angesetzt.
Auf diesen 43 Bildern befinden sich insgesamt58 Kennzeichen.
Die Größe der Bilder in Pixel beträgt bei allen Blder 2560 x 1920px, beziehungsweise 1920 x 2560px sofern es sich um ein Hochformatbild handelt. Alle Bilder des Testdatensatzes wurden durch Gruppenmitglieder erzeugt und bereitgestellt.

5.2 Sind alle gefundenen Objekte Schilder oder wurden falsche Objekte detektiert?

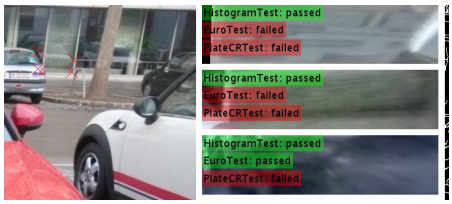


Abbildung 14

Am Ende des Algorithmus werden gefundenen Merkmale die nicht Kennzeichen entsprechen nicht als solches identifiziert. Daher ist gewährleistet dass es sich bei allen gefundenen Übereinstimmung auch um Kennzeichen handelt. Es werden jedoch alle Merkmale die gefunden wurden zur Überprüfung überreicht. Da der Algorithmus auf rechteckige Merkmale basiert, kommt es natürlich vor, dass rechteckige Objekte zur Überprüfung bereitgestellt werden - in Abb14 finden sich solche Objekte vor - in diesem Bild wird zum Beispiel eine Fensterfront erkannt.

Die anderen Überprüfungen sorgen jedoch dafür, dass derartige Objekte nicht als Kennzeichen erkannt werden. Die Bildnummer aus dem zugehörigen Datensatz ist Nummer 2.

5.3 Wurden alle vollständig sichtbaren Schilder detektiert?

Der Algorithmus erkennt einen Großteil des Inputs und kann eine Vielzahl von Kennzeichen finden. Im Testdatensatz wurden diverse Kennzeichen jedoch nicht erkannt. Es handelt sich hierbei um die Bilder 1,3,6,9,15,16,18,32. Einige von diesen Bildern enthalten mehrere Kennzeichen. Die Erkennung von mehreren Kennzeichen in einem Bild ist auf alle Fälle gewährleistet, jedoch nicht in allen Fällen möglich gewesen. Im nächsten Abschnitt wird darauf eingegangen warum die Erkennung bei diesen Kennzeichen fehlschlug.

5.4 Woran scheitern nicht gefundene Objekte?

Ein Grundstein des Algorithmus ist die Verwendung des Canny-Filters. In einigen Fällen kann es vorkommen, dass der Canny-Filter Löcher im Bereich der Kennzeichen lässt. Beim Füllen dieser Flächen werden diese Flächen im weiteren Verlauf natürlich inkorrekt gefüllt und entsprechen nicht mehr dem eigentlichen Kennzeichen. Dadurch verliert die Fläche sämtliche Rechtecksform und wird auch im weiteren Verlauf des Algorithmus aus der Betrachtnahme ausgeschlossen.

Abbildung 15

Dieses Problem wurde insofern versucht zu behoben, dass die gelassenen Lücken geschlossen werden. Dadurch können sich aber auch Flächen, die nicht zusammengehören verbinden und so falsche Flächen bilden.
Ein weiteres Problem kann durch die Buchstabenprüfung auftreten. Erscheint die Schrift im transformierten Bild etwas schräg, kann es sein, dass die Buchstaben teilweise nicht mehr erkannt werden, so dass das Objekt nicht als Kennzeichen erkannt wird.avel
Auch bei der Rechteckserkennung, welche jedoch eher kaum Probleme generierte, traten Interessante Fälle auch. Zum Beispiel wurde hier ein Logo der Marke Audi als Rechteck erkannt. Wenn man sich jenes Logo und die erzeugte Funktion genauer ansieht liegt es grundsätzlich nahe, dass es zu dieser Erkennung kommen kann.

5.5 Wurden alle Kennzeichen korrekt transformiert?

Alle Bilder aus dem Testdatensatz konnten korrekt transformiert werden und in sinngemäß ausgerichtet in der Endanzeige angezeigt werden.

5.6 Sind alle Bilder vollständig oder fehlt ein Teil?

In unserem Testdatensatz wurde nur ein Kennzeichen nicht vollständig erkannt. Uu diesem Problem führte war ein Blaustich innerhalb des Kennzeichens, wodurch der Algorithmus dachte, der blaue Abschnitt seie bereits vorhanden.

Abbildung 16

5.7 Wie viele Prozent der Testbilder führten zu einem korrekten Ergebnis?

Von allen 58 gegebenen Kennzeichen wurden wurden 50 Kennzeichen korrekt erkannt. Daher wurden 86.20 Prozent des Testdatensatzes korrekt erkannt. 8 Kennzeichen konnten aufgrund von oben genannten Problemen nicht erkannt werden.

5.8 Wie verhält sich die Laufzeit des Algorithmus, woran liegen Probleme?

Der Algorithmus hat eine langsame Laufzeit die auf den vielen vorgenommenen Operationen beruht. Ein Bild braucht durchschnittlich 90 Sekunden (auf einem i5 Quadcore mit 3.5 GHz) Dadurch, dass unser Canny-Filter eine große Menge an Kanten erzeugt, kann es sehr lange dauern, alle Flächen in dem Bild zu finden, insbesondere, wenn viele solche Flächen entstehen (z.B. bei Gras). Grund dafür ist vor allem die imfill-Methode von Matlab. Die selbstimplementierten Teile haben eine relativ stabile Laufzeit.

6 Schlusswort

Um Kennzeichen aus einem Bild maschinell zu erkennen muss man die zur Verfügung gestellten Informationen richtig interpretieren. Um die Daten korrekt zu behandeln bietet es sich an sich Gedanken über die menschliche Wahrnehmung im Bezug zur Erkennung von Objekten zu machen. Es stellt sich also die Frage: wie entscheiden wir ob wir ein Kennzeichen sehen oder nicht? So wirkt eine erste Analyse trivial und man kann rasch folgende Merkmale zusammenfassen: ein Kennzeichen ist meist eine rechteckige, mit Zahlen und Buchstaben beschriftete Fläche. Außerdem hat sie durch das Auftreten der Symbole ein gewisses Farb-/Kontrastverhältnis. Nachdem wir speziell nach europäischen Kennzeichen suchen, kann man als zusätzliches Merkmal die Europa Kennzeichnung hinzuzählen.

All die genannten Aspekte waren Herausforderungen, die mit Bravur bewältigt werden konnten. Die einzig größte Herausforderung lag tatsächlich darin die richtigen Flächen aus Kantenbildern zu erzeugen. Unglücklicherweise entstehen bei der Anwendung der Kantenerkennung Fehler, mit denen sich kaum arbeiten lässt. So kommt es manchmal vor, dass die Kanten des Kennzeichens nicht vollständig umschlossen sind und der Flächenfüllalgorithmus mehr füllt als gewollt. Auch die Größe des Bildes bereitet manchmal Probleme: wenn die äußeren Kanten des Kennzeichens mit den Symbolen auf

der Tafel in Verbindung kommen, so wird das Kennzeichen nicht richtig gefüllt und die gesuchte Fläche in weiterer Folge nicht erkannt.

Literatur

[1] Wilhelm Burger and Mark James Burge. *Digitale Bildverarbeitung Eine Einführung mit Java und ImageJ*, chapter 16, pages 363–373. Springer, 2 edition, 2006.

[2] J. Canny. A computaional approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8:679–698, 11 1986.

[3] M.A. Massoud, M. Sabee, M. Gergais, and R. Bakhit. Automated new license plate recognition in egypt. *Alexandria Engineering Journal*, 52:319–326, 2013.

[4] T. Moeslund. Canny edge detection. 3 2009.

[5] A. Nischwitz, M. Fischer, P. Haberäcker, and G. Socher. *Computergrafik und Bildverarbeitung Band II: Bildverarbeitung*, chapter 8, pages 238–250. Vieweg+Teubner, 3 edition, 2011.

[6] Robert Sablatnig and Werner Purgathofer. Einführung in visual computing: Skriptum zur vu.