Bericht: Teil 1, Problemlösung

# Verkehrszeichen erkennen

AG_D04
Ortmayr Tobias (1026279)
Scalisi Riccardo (0626036)
Vorderegger Bernhard (9609556)
Machner Natascha Josephine (1027745)
Schönbauer Michaela (1026306)

## 1. Konzept

**Ziel**
Ziel ist es, auf einem Farbfoto eines von drei ausgewählten Verkehrszeichen zu finden und richtig zu erkennen.

**Eingabe**
Input ist ein Farbfoto mit bestimmter Auflösung, auf dem eines von drei ausgewählten Verkehrszeichen abgebildet ist (nähere Details: siehe <Voraussetzungen und Bedingungen>)

**Ausgabe**
Output ist der Name des gefundenen Verkehrsschildes (textuell) sowie eine Hervorhebung des gefundenen Verkehrsschildes durch Nachzeichnen der Konturen (visuell). Weiters wird im Command Window ausgegeben, welches Verkehrszeichen gefunden wurde sowie dessen Position anhand von x- und y-Koordinaten. Zusätzlich wird für ein Dreieck bzw. eine Raute das Seitenmaß ausgegeben, für einen Kreis der Radius.

**Voraussetzungen und Bedingungen**
- Folgende Verkehrszeichen sind erlaubt: Vorrangstraßen-Schild, Vorrang-Geben-Schild, Einfahrt-Verboten-Schild
- Es darf sich nur eines der drei oben genannten Verkehrszeichen im Bild befinden; allgemein darf sich nur ein Verkehrszeichen im Bild befinden
- Bild wurde möglichst frontal von vorne aufgenommen; es sind möglichst keine perspektivischen Verzerrungen entstanden
- Im Bild befindet sich als Hauptmotiv das Verkehrszeichen an sich (nicht zusätzlich die gesamte Stange, auf der es aufgestellt wurde)
- Verkehrszeichen muss vollständig im Bild sein; Kanten sollten nicht den Rand berühren
- Verkehrszeichen ist – soweit möglich – klar abgegrenzt vom Hintergrund
- Verkehrszeichen wird nicht verdeckt (z.B. von Ästen o. Ä.)
- gleichmäßige und gute Beleuchtung
- maximal 1024x1024 pix, minimal 512x128 pix, wobei das Verkehrszeichen mindestens 1/8 des Bildes einnehmen muss.

**Methodik**
1. Aufnahme eines Farbbildes mit einer Kamera
2. Filtern der potenziellen Verkehrszeichen anhand der Farbe

   a. Parallel werden ein Rot-Filter sowie ein Gelb-Filter angewandt (HSV-Farbmodell)
   b. Umwandeln des Bildes in ein Binärbild
3. Anwenden eines CCL-Filters
   a. Die größten zusammenhängenden Regionen werden gesucht und nur noch diese angezeigt, die restlichen Regionen werden ausgefiltert
   b. Löcher in Flächen werden gefüllt (wodurch bei jenen Verkehrszeichen, die nach dem Filtern einen weißen Rand und ein schwarzes Inneres aufweisen, dieses ausgefüllt werden soll)
4. Anwendung eines Gaußfilters zum Glätten der Kanten der übriggebliebenen Regionen
5. Anwendung eines Sobel-Kantenfilters (wodurch von einem Verkehrszeichen im Bild nur noch die Konturen übrig bleiben sollen)
6. Anwendung einer Hough-Transformation: Kreis- und Linienvariante
   a. Linien: Finden von Dreiecken und Rauten
   b. Kreise: Finden von Kreisen
7. Verifikation der eindeutig identifizierenden Form des Verkehrsschildes (Kreis, Dreieck, Raute): Deckungsgrad gefilterte Fläche und generierte Form
8. Zuordnen der gefundenen Form zu einem Verkehrschild und Ausgabe des Ergebnisses (Typ + Position)

**Evaluierung**

Ground Truth: Für jedes den Eingabebedingungen entsprechende Bild werden der Typ und die Position des Verkehrsschildes bestimmt.

Evaluierungskriterien:

- Wurde der Typ richtig erkannt?
- Stimmt die Position des Verkehrsschildes?
- Welche Bedingungen tragen zu einer erfolgreichen Erkennung bei/halten davon ab?

## 2. Arbeitsteilung

| Name | Funktionen |
|---|---|
| Ortmayr Tobias | - hough_lines<br>- hough_pixels<br>- hough_transform<br>- find_lines<br>- find_peaks |
| Scalisi Riccardo | - bresenham<br>- hough_circle<br>- hsv_filter<br>- ImFill<br>- intersectionPoint |
| Vorderegger Bernhard | - checkLinesInRaute<br>- checkLinesInTriangle<br>- evalRaute<br>- evalTriangle<br>- findsign |

| | |
|---|---|
| Machner Natascha | - CCLfilter<br>- label<br>- plot_rauten<br>- plot_triangle<br>- sobelF |
| Schönbauer Michaela | - CCLfilter<br>- gaussian_filter<br>- label<br>- plot_hough_lines<br>- sp_showImage |

**Weitere Aufgaben:**
Dokumentation der Arbeit sowie Evaluierung sind in der Gruppe erfolgt und lassen sich daher nicht auf bestimmte Personen aufteilen.
Die einzelnen Kapitel des Berichts wurden in der Gruppe besprochen; Konzept und die endgültige Implementierung gemeinsam durchdacht.
An der Evaluierung war ebenfalls jedes Gruppenmitglied beteiligt. Gemeinsam wurden Testdaten gesammelt, getestet und Ergebnisse diskutiert.

## 3. Methodik

**findsign(image,debugMode,debugMode2)**

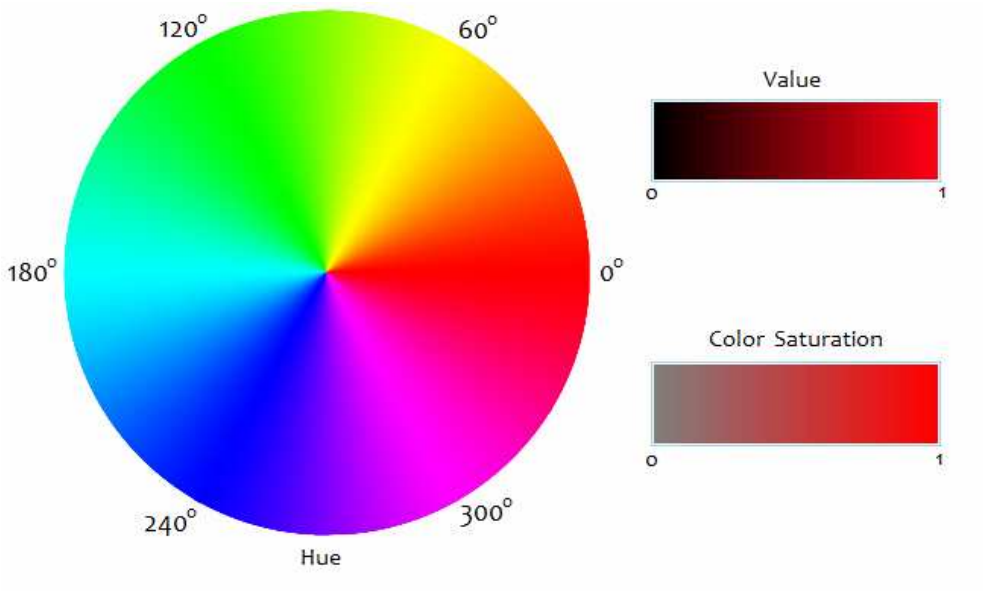
| | |
|---|---|
| @input image: Pfad des Testbildes | |
| @input debugMode: | wenn 1 → Bilder der Zwischenergebnisse ab Rotfilter werden ausgegeben<br>wenn 0 (oder weggelassen) → nur Endergebnis wird ausgegeben |
| @input debugMode2: | wenn 1 → Bilder der Zwischenergebnisse ab Gelbfilter werden ausgegeben<br>wenn 0 (oder weggelassen) → nur Endergebnis wird ausgegeben |
| @output: Ergebnisbild (mit Name des gefunden Verkehrszeichen sowie Markierung seiner Position) | |

Das ist die für den Benutzer/die Benutzerin relevante Funktion, die dieser/diese für ein Testbild aufrufen muss. Nähere Erklärung des Testablaufs bzw. der Verwendung dieser Funktion sowie ihrer Parameter → vgl. Kapitel „5. Implementierung“

 *Schritt 1*

Zu Beginn wird in findsign das Bild, das unter dem übergebenen Pfad liegt, eingelesen und in den HSV-Farmraum konvertiert.
Das HSV Farbmodell ist unterteilt in H Farbton, S Sättigung und V Hellwert,
daher ermöglicht der HSV Raum eine 1 dimensionale Trennung der Farbbereiche.

Anschließend wird die Funktion hsv_filter aufgerufen, einmal mit dem Parameter type = r und einmal mit type = y.



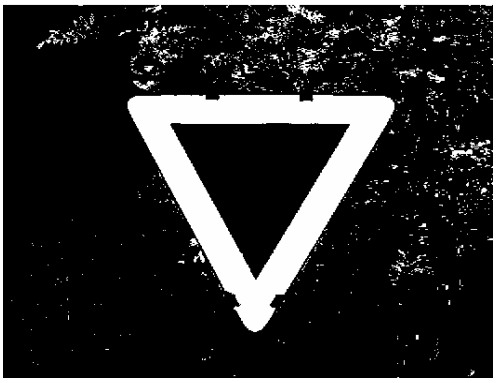

**hsv_filter(hsvimage,type)**

| |
|---|
| @input hsvimage: Ergebnisbild der HSV-Farbraum-Konvertierung<br>@input type: wenn r → nur Elemente im Bild mit Rotanteil bleiben übrig<br>wenn y → nur Elemente im Bild mit Gelbanteil bleiben übrig<br>@output: Binärbild (Pixel, die ausgefiltert werden, werden auf 0 gesetzt; die anderen auf 1) |

Diese Funktion lässt einen Rotfilter über das Eingabebild (hsvimage) laufen. Dadurch sollen alle Bereiche, die nicht bestimmte Farbton-, Sättigungs- und Helligkeitswerte aufweisen, ausgefiltert werden.
Als type kann r (=red) oder y (=yellow) übergeben werden.
Bei type = r sollen in erster Linie rotfarbige Bereiche im Bild übrigbleiben. Die Funktion ist für jene Testbilder wichtig, auf denen sich rotfarbige Verkehrsschilder („Einfahrt verboten“ und „Vorrang geben“) befinden. Überflüssige Elemente im Bild, die nicht zum Verkehrsschild gehören, sollen weitgehend ausgefiltert werden.
Bei type = y sollen in erster Linie gelbfarbige Bereiche im Bild übrigbleiben. Die Funktion ist für jene Testbilder wichtig, auf denen sich ein gelbes Verkehrsschild („Vorrangstraße“) befindet. Überflüssige Elemente im Bild sollen wieder weitgehend ausgefiltert werden.

Diese Funktion eignet sich gut als Vorfilterung. Verkehrszeichen zeichnen sich durch bestimmte Farben aus (in diesem Fall rot oder gelb). Durch das Anwenden eines Rot- bzw. Gelbfilters können Regionen im Bild, die keine dieser Farben aufweisen, leicht ausgefiltert werden.

Die H Werte wurden jeweils auf den Gelbbereich und den Rotbereich eingeschränkt. Um die Erkennungsrate zu erhöhen wurden die S und V Werte nahe dem Minimalwert gewählt.

*Beispiel: Vorrang_geben2*
*Durch den Rotfilter bleiben nur jene Regionen als weiße Pixel übrig, die einen gewissen Rotanteil aufweisen. Alle anderen Regionen werden ausgefiltert.*

*Schritt 2*

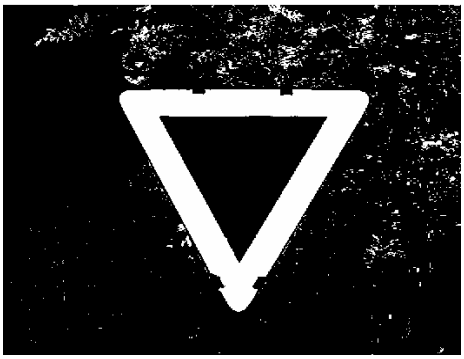
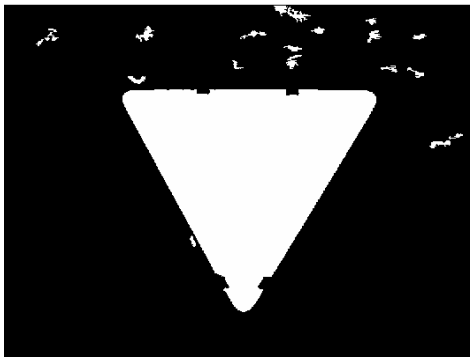
Im nächsten Schritt wird ein CCL-Filter aufgerufen.

**CCLfilter(img)**

| |
|---|
| @input img: Ergebnisbild von hsvfilter<br>@output: Binärbild (Pixel, die ausgefiltert werden, werden auf 0 gesetzt; die anderen auf 1) |

Diese Funktion dient dazu, zusammenhängende Regionen im Bild zu finden. Für jeden Pixel, der den Wert 1 hat, wird gespeichert, zu welcher Region er gehört. Anschließend werden nur noch jene Regionen einer bestimmten Größe gezeichnet, der Rest wird auf 0 gesetzt.
Weiters werden Löcher durch diese Funktion gefüllt. Das ist wichtig, um Verkehrszeichen, die innen Löcher aufweisen (z.B. Vorrang-Geben-Schild → weißer Rand, innen schwarzes Dreieck), weiß auszufüllen. Das erleichtert später die Erkennung, um welches Verkehrszeichen es sich handelt, da weniger Kanten gefunden werden (→ keine Kanten mehr im „Innenbereich“ des Verkehrsschildes).

CCLfilter verwendet die beiden Funktionen **label(img)** und **ImFill(image)**. Während label(img) dafür zuständig ist, zusammenhängende Regionen zu finden und diese Information entsprechend zu speichern, werden in ImFill(image) die „Löcher“ im Bild weiß gefüllt.

*Beispiel: Vorrang_geben2*
*Kleine Regionen werden ausgefiltert, das Verkehrszeichen wird weiß ausgefüllt*

*Schritt 3*

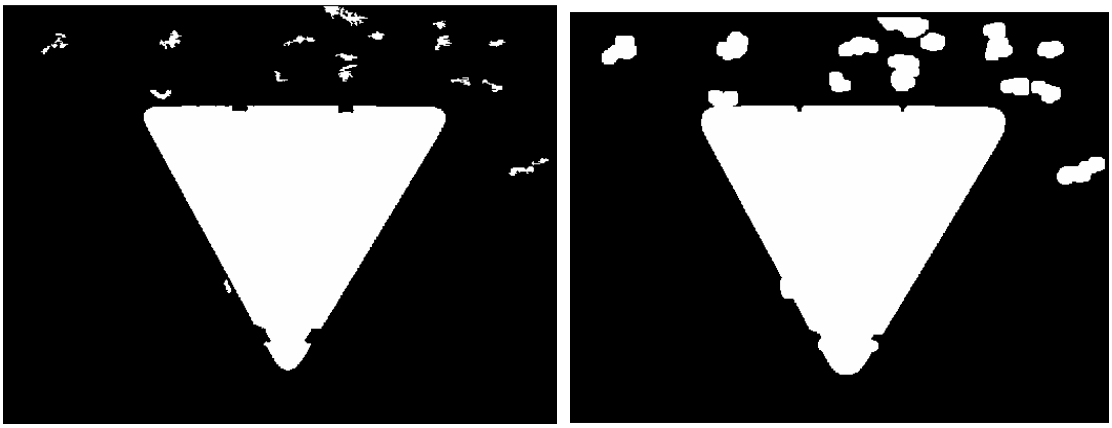
Im nächsten Schritt wird ein Gaußfilter aufgerufen.

**gaussian_filter(image,hsize)**

| |
|---|
| @input image: Ergebnisbild von CCLFilter<br>@input hsize: größe des Filterkerns (muss ungerade sein)<br>@output: geglättetes Binärbild |

Diese Funktion glättet Unebenheiten im Bild. Das soll später die Kantenfindung bzw. die Kreissuche erleichtern, da beispielsweise „ausgefranste“ Kanten geglättet werden. Ein ungerader Filterkern wird verwendet, da dieser ein eindeutiges Mitten-Pixel hat, das berechnet werden kann. Die Funktion findsign ruft gaussian_filter für jedes der zwei Ergebnisbilder von CCLFilter mit dem Parameter hsize = 15 auf. Das erzeugt einen 15x15 Filterkern. Pixel, die nahe zum Rand liegen und über den Filterkern nicht so einfach berechnet werden können, erhalten automatisch den Wert 0. Da die Testbilder eine Auflösung aufweisen sollen, die hoch genug ist, wirkt sich das Ignorieren weniger Pixelreihen nicht negativ auf das Endergebnis aus. Allerdings sollten nicht zu viele Pixelreihen beim Glätten ignoriert (und dadurch ausgefiltert) werden, weswegen der Filterkern eher klein gewählt wurde. Die Glättung ist für einen erfolgreichen Testweiterlauf aber ausreichend.

Anschließend an den Gauss-Filter werden alle Pixelwerte, die größer 0 sind, auf 1 gesetzt, wodurch wieder ein Binärbild entsteht. Das vergrößert zwar die Regionen etwas, da nun viele Pixel weiß werden, führt jedoch zu einer stärkeren Glättung und erleichtert die korrekte Verkehrszeichen-erkennung.



*Beispiel: Vorran*g_geben2
*Durch den Gauss-Filter und das anschließende Setzen aller Pixel > 0 auf 1 werden die weißen Regionen im Bild zwar etwas vergrößert, jedoch stärker geglättet.*

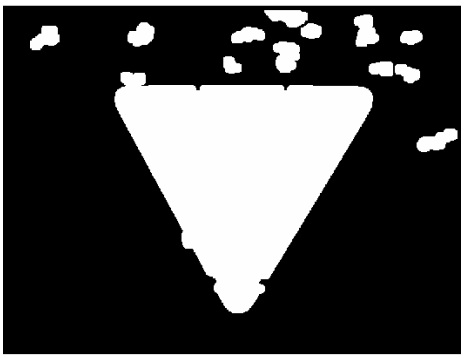
*Schritt 4*

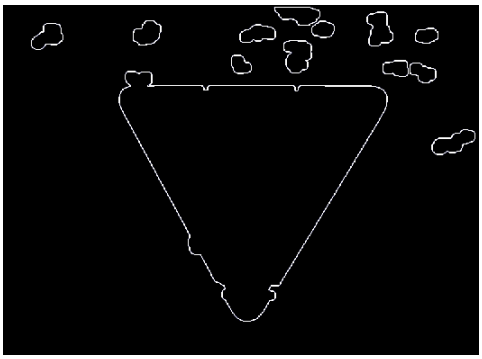
Im nächsten Schritt wird ein Sobel-Filter aufgerufen.

**sobelF(img)**

| |
|---|
| @input image: Ergebnisbild von gaussian_filter<br>@output: binäres Kantenbild des Eingabebildes |

Diese Funktion erzeugt von dem Eingabebild ein Kantenbild und verwendet dazu den Sobel-Algorithmus. Dies soll die weitere Verarbeitung vereinfachen, indem das Bild auf die formgebenden Kanten reduziert wird.
Der Sobel-Algorithmus wurde hier gewählt, weil er ein relativ einfacher und wenig kostenintensiver Kantendetektor ist, der für die weitere Bearbeitung aber ausreichend brauchbare Ergebnisse liefert. Auch hier wurde wie beim Gauss-Filter ein ungerader Filterkern – nämlich der Standart-3x3-Filterkern für den Sobel-Operator jeweils in horizontaler und vertikaler Richtung – verwendet. Da in den zu verarbeitenden Bildern unter anderem Kreise gesucht werden, wäre eine Betonung in eine bestimmte Richtung wenig sinnvoll. Deswegen werden für jeden Pixel die Werte jeweils in x (GX) und y Richtung (GY) mit Hilfe der Filterkerne ermittelt, normalisiert und anschließend miteinander kombiniert (Formel: sqrt( GY^2 + GX^2) ), um eine Richtungsunabhängigkeit zu erreichen.
Es erfolgt eine primitive Randbehandlung, bei der die fehlenden Werde mit Nullen aufgefüllt werden.





*Beispiel: Vorrang_geben2*
*Durch den Sobel-Filter bleiben nur noch die Kanten aller Objekte im Bild übrig.*

*Schritt 5*

**hough_lines(BW)**

| |
|---|
| @input BW: binäres Kantenbild des Eingabebildes<br>@output lines: Structure-Array, das die gefundenen Linien enthält (Anfangs- und Endpunkt sowie Rho- und Theta-Werte) |

Finden von Linien im Kantenbild:

Diese Funktion transformiert zuerst das Kantenbild in den Hough-Raum (vgl. Houghtransformation im EBDV-Skript). Aus dem Akkumulatorarray, welches den Raum repräsentiert, werden die Geraden,

die am Häufigsten verkommen, herausgefiltert (max. 15). Im nächsten Schritt wird für jede Gerade untersucht, welche Pixel aus dem Kantenbild auf dieser liegen, und falls möglich, werden diese Pixel zu einer Linie zusammengefasst. Die gefundenen Linien werden im einem Structure-Array zurückgegeben und dienen dann als Grundlage für die Dreicks-/Rautenerkennung.

Die Funktion besteht aus 3 Unterfunktionen:

**- hough_tranform:**
Diese Funktion führt die eigentliche Hough-Transformation durch. Als Inputbild wird das Kantenbild verwendet. Der Output ist ein Vektor bestehend aus dem Akkumulatorarray H, einem Array mit den verwendeten Rho-Werten und einem Array mit den verwendeten Theta-Werten.

**- find_peaks:**
Diese Funktion such aus dem  Akkumulatorarray die größten Peaks (max. 15) und gibt diese zurück. Es werden erst Einträge des Arrays über einen bestimmen Schwellenwert als Peaks in Betracht gezogen (>0.5*dem größten Eintrag des Arrays; selber Schwellenwert wie im Matlab-Pendant)

**- find_lines:**
Diese Funktion sucht dann zu den Peaks entsprechende Linien. Zuerst werden für jeden Peak, alle Pixel des Kantenbilds, die auf der durch ihn dargestellten Geraden liegen, gesucht und zu Linien zusammengefasst. Nun hat man mehrere Liniensegmente. Segmente, die kleiner als der 'minLength' Paramtert (25) sind, werden verworfen und wenn der Abstand zwischen zwei Segmenten kleiner als 'fillGap' (20, Defaultwert) ist, so werden diese zu einer Linie kombiniert, also die Lücke zwischen den Segmenten wird gefüllt. Die gefundenen Linen werden in einem Structure-Array gespeichert, wobei jede Linie aus Anfangspunkt, Endpunkt, Theta-Wert und Rho-Wert besteht.

*Schritt 6 (nur im Debug-Mode)*

**plot_hough_lines(image, lines, count)**

| |
|---|
| @input image: Eingabebild, auf dem Linien markiert werden sollen<br>@input lines: zuvor gefundene Hough-Lines<br>@input count: Figure-Index<br>@output:  neuer Figure-Index (count+1) |

Im Debug-Mode wird diese Funktion gleich nach hough_lines aufgerufen. Sie zeichnet in dem Testbild alle Linien ein, die zuvor über hough_lines gefunden wurden. Die längste Linie wird dabei blau markiert, alle anderen Linien grün. Die Anfangspunkte der Linien werden durch ein gelbes Kreuz markiert, die Endpunkte der Linien durch ein rotes.

*Beispiel: Vorrang_geben2; Kanten im Bild werden eingezeichnet*

*Schritt 7*

**eval_triangle(lines, image)**

| |
|---|
| @input lines:  Hough Lines<br>@input image:  Input Image<br>@output:  Resultierende Dreiecke; Eckpunkte + Schwerpunkt |

Für die Dreieckserkennung wurde ein Brute Force Algorithmus gewählt. Dabei werden anhand der Hough Lines alle Schnittpunkte ermittelt. Schnittpunkte außerhalb des Bildes werden entfernt. Anhand zweier Punkte wird entlang der Normalen ausgehend vom Streckenmittelpunkt innerhalb eines Toleranzbereichs der dritte Punkt des Dreiecks evaluiert.
Kriterien für die Auswahl:

- Dreieck muss eine Mindestgröße haben
- Kantenüberdeckung durch Hough Linien muss auf allen 3 Kanten mit einem Mindestmaß erfüllt sein (ca. 30%)
- Dreieck mit bester Passung wird ausgewählt.
- Dreiecksüberdeckungen werden ausgefiltert.

Einschränkung für diese Methode ergeben sich hauptsächlich durch große Anzahl von Hough Linien (>20) bzw. viele gefunden Schnittpunkte. Des Weiteren ergibt sich durch die Normale ein Limit aufgrund perspektivischer Y Achsendrehungen.

Anschließend werden die Linien des gefundenen Dreiecks für
**plot_triangle(triangles,image,count,debugMode)** in das Bild blaut eingezeichnet.

*Schritt 8*

**eval_rauten (lines, image)**

| |
|---|
| @input lines: Hough Linien<br>@input image:  Input Image<br>@output: 4 Eckpunkte und Schwerpunkt |

Für die Vorrangstraßenerkennung wurde ebenfalls der Brute Force Ansatz wie für die Dreieckser-kennung gewählt. Hier werden im Wesentlichen wieder Punkte gesucht, die am genauesten eine Raute darstellen.
Um perspektivische X-Achsendrehungen zu erkennen wird nach einer Evaluierung aufgrund der Dreierkombinationen der  vierte Punkt mit der geringsten Abweichung vom berechneten gesucht.

Anschließend werden die Linien der gefundenen Raute über
**plot_rauten(rauten,image,count,debugMode)** in das Bild blau eingezeichnet.

*Schritt 9*

**hough_circle(image, signHeightMin, signHeightMax, precision)**

| |
|---|
| @input image: Kantenbild<br>@input signHeightMin, signHeightMax: Höhe (in % der Bildhöhe), die der gesuchte Radius annehmen kann<br>@input precision: es wird nur jeder n-te Radius untersucht<br>@output: Koordinaten und Radius des gefunden Kreises + Wahrscheinlichkeit, dass es sich um einen tatsächlichen Kreis handelt (unwahrscheinleich wenn < 0.9) |

Houghtransformation für Kreise:

Es wird eine 3D Houghtransformation durchgeführt, für Kreisposition (x, y) und Kreisradius. In einem Akkumulatorarray (Dimensionen: Bildhöhe x Bildbreite x Radii) werden für jeden untersuchten Kreisradius und für jedes Feature im Kantenbild (Wert > 0) ein Kreis gezeichnet. Dieser Kreis repräsentiert alle möglichen Kreismittelpunkte, die dieses Feature mit diesem Radius erzeugt haben könnte. Dort wo sich die meisten dieser potenziellen Mittelpunkte häufen wird ein tatsächlicher Kreis im Bild angenommen. Um das sicherzustellen wird der so akkumulierte Wert normiert und dann mit einem Schwellwert verglichen. Wenn das Maximum nur durch sehr wenige Punkte enstanden ist, wird das Egebnis verworfen.

Die Funktion bresenham(r) ist eine Hilfsfunktion und dient dem Erstellen von Kreispunkten mit einem gewissen Radius (=r). Der Algorithmus basiert auf dem "Midpoint Circle Algorithm" von Bresenham.



*Beispiele für ausgewertete Verkehrsschilder:*

- *Vorrang_geben2*
- *Vorrangstraße1*
- *Einfahrt_verboten1*

## 4. Literatur

@mastersthesis {
author = {Aino de Vries},
title = {Offline Verkehrszeichenerkennung für Geoinformationssysteme},
school = {Fachhochschule Oldenburg, Ostfriesland, Wilhelmshaven},
year = {2008},
month = {Juli},
type = {Diplomarbeit}
}

Kurzbeschreibung
Aino de Vries geht in seiner Diplomarbeit unter anderem darauf ein, wie man Verkehrszeichenerkennung über deren typische Farben sowie Formen in der Bildverarbeitung realisieren kann.
http://www.itwm.fraunhofer.de/fileadmin/ITWM-Media/Abteilungen/BV/Pdf/Diplomarbeit_Vries.pdf

@article {
author = {Markus Schmitt, Thorsten Lorenzen},
title = {Fahrerassistenz-Systeme mit Prozessoren:
Visuelle Verkehrszeichenerkennung mit dem Blackfin},
journal = {Hanser Automotive},
year = {2006},
month = {November}
}

Kurzbeschreibung:
Der Artikel beschäftigt sich mit dem Aufsetzen einer Verkehrszeichenerkennung auf einem Blackfin-Prozessor. Es wird auf Grundelemente in der Verkehrszeichenerkennung eingegangen. Dabei sind unter anderem wie in unserem Projekt der Sobel-Katenfilter sowie die Houghtransformation und die Kreissuche von Bedeutung.
http://www.hanser-automotive.de/fileadmin/heftarchiv/2004/14993.pdf

@book {
author = {Sergio Escalera, Xavier Baró, Oriol Pujol, Jordi Vitrià, Petia Redeva},
title = {Traffic-Sign Recognition Systems},
publisher = {Springer},
year = {2011}
}

Kurzbeschreibung:
Das Buch beschäftigt sich allgemein mit Systemen zur Verkehrszeichenerkennung. Dabei wird darauf eingegangen, dass spezielle Farben sowie Formen bei der Erkennung von Verkehrszeichen von Bedeutung sind. Eine Vorschau zum Buch befindet sich auf Google-Books:
http://books.google.at/books?id=QpqkgZ87Ic0C&pg=PA13&dq=new+results+on+traffic+sign+recognition&hl=de&sa=X&ei=0rjMUOKgAsXhtQbIiYHYDw&ved=0CDIQ6AEwAA

@misc {
	author = {Michael Morak, Johannes Perl, Ignaz Reicht, Thomas Rittler},
	title = {Verkehrszeichenerkennung durch Color Thresholding},
	year = {2010}
}

Kurzbeschreibung:
Wie der Titel bereits verrät, beschäftigt sich diese Ausarbeitung insbesondere mit der Erkennung von Verkehrsschildern über deren spezielle Farbinformation.
http://www.perlproductions.at/ref_docs/doc_1244434298.pdf

@article {
	author = {Prof. Dr.-Ing. Reiner Schmid, M.Sc. Tamas Liszkai, M.Sc. Tamas Bajcsi},
	title = {Verkehrszeichenerkennung},
	journal = {aktuelletechnik},
	year = {2012}
}

Kurzbeschreibung:
In diesem Artikel wird auf Verkehrszeichenerkennung im Zusammenhang mit deren Farben und Formen eingegangen. Dabei wird die Hough Transformation erwähnt sowie als Alternativvorschlag der RANSAC-Algorithmus.
http://www.aktuelletechnik.ch/Web/InternetAT.nsf/0/B7D45FAD2354FC62C1257AA2002093B8?OpenDocument&parm=parm

**Weiteres Interessantes:**

@misc {
	title = {Das Mammut-Praktikum},
}

Kurzbeschreibung:
Dieses Dokument geht auf die Ausarbeitung einer Praktikumsaufgabe ein, bei der Verkehrszeichen auf einem Kamerabild erkannt werden sollten.
http://www.tu-chemnitz.de/etit/nt/home/images/stories/lehre/Mammut_Praktikum/mammut_praktikum.pdf

@misc {
	title = {Zeichen der Zeit. Bildverarbeitungs-FPGAs in der Verkehrszeichenerkennung}
}

Kurzbeschreibung:
In diesem Dokument wird auf Bildverarbeitungstechniken zur Verkehrszeichenerkennung eingegangen. Dabei sind wie in unserem Projekt der Sobel-Katenfilter sowie die Houghtransformation von Bedeutung. http://www.silicon-software.de/download/archive/Zeichen_der_Zeit.pdf

## 5. Implementierung

Zum Ausführen des MATLAB-Codes wird nur die Funktion findsign(image, debugMode, debugMode2) benötigt.
Der Parameter ‚image' ist dabei der Pfad zu dem Testbild, mit dem der Code ausgeführt werden soll. Die Parameter ‚debugMode' und ‚debugMode2' können 0 oder 1 sein. Bei debugMode=1 werden die Ergebnisbilder der einzelnen Zwischenschritte ab dem Rotfilter angezeigt, bei debugMode = 0 nicht. Bei debugMode2=1 werden die Ergebnisbilder der einzelnen Zwischenschritte ab dem Gelbfilter angezeigt, bei debugMode2=0 nicht. Die Parameter lassen sich beliebig kombinieren. debugMode=1 und debugMode2=1 führen dazu, dass sowohl ab dem Rotfilter als auch ab dem Gelbfilter alle Zwischenergebnisse als Bilder ausgegeben werden. Sind beide Parameter 0, wird nur das Endergebnis mit dem gefundenen Verkehrszeichen ausgegeben, welches namentlich beschriftet und durch Nachzeichnen an den Konturen hervorgehoben wird.
Die debug-Parameter können auch weggelassen werden. Übergibt man nur einen debug-Parameter, wird debug2 automatisch auf 0 gesetzt. Übergibt man gar keine debug-Parameter, werden beide auf 0 gesetzt.

Wird findsign ausgeführt, ruft diese Funktion automatisch alle anderen nötigen Funktionen in der richtigen Reihenfolge auf und gibt dem Benutzer / der Benutzerin schließlich das Endergebnis aus. Für die Kreissuche wird ein Fortschrittbalken im Command Window anzeigt, wie viel Prozent der Kreissuche bereits durchgeführt wurden.
Ausgabe ist anschließend das Testbild mit Name des gefundenen Verkehrszeichens sowie Markierung dessen Konturen. Weiters wird im Command-Window ausgegeben

- Name des Verkehrszeichens, das gefunden wurde
- x- und y-Koordinaten der Position des Verkehrszeichens
- Seitenmaß des Verkehrszeichens

## 6. Evaluierung

**Verwendeter Datensatz:**


Vorrang_geben1


Vorrang_geben2

Vorrang_geben3

Vorrang_geben4

Vorrang_geben5

Einfahrt_verboten1

Einfahrt_verboten2


Einfahrt_verboten3

Einfahrt_verboten4


Einfahrt_verboten5

Einfahrt_verboten6

Einfahrt_verboten7

Vorrangstraße1

Vorrangstraße2

Vorrangstraße3

Vorrangstraße4

Vorrangstraße5

Vorrangstraße6

Vorrangstraße7

**Testfälle in Tabellenübersicht**

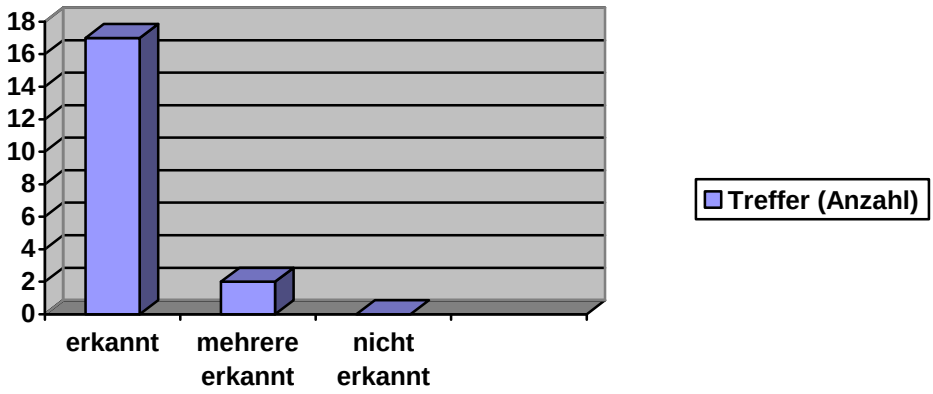
| Name | Typ richtig | Position richtig | Sonstiges |
|---|---|---|---|
| Vorrang_geben1 | Ja | Ja | |
| Vorrang_geben2 | Ja | Ja | |
| Vorrang_geben3 | Ja | Ja | |
| Vorrang_geben4 | Ja | Ja | |
| Vorrang_geben5 | Ja/Nein | Nach rechts verschoben, verkleinert | Der Typ des Verkehrszeichens wird richtig erkannt. Jedoch wird zusätzlich ein Kreis gefunden und als Einfahrt-Verboten-Schild erkannt. |
| Einfahrt_verboten1 | Ja | Ja | |
| Einfahrt_verboten2 | Ja | Ja | |
| Einfahrt_verboten3 | Ja | Nach rechts verschoben | |
| Einfahrt_verboten4 | Ja | Nach links verschoben | Bild leicht verzerrt |
| Einfahrt_verboten5 | Ja | Ganz leicht nach rechts verschoben | |
| Einfahrt_verboten6 | Ja | Leicht nach rechts verschoben | Unscharfes Bild |
| Einfahrt_verboten7 | Ja | Leicht nach unten verschoben | Bild leicht verzerrt |
| Vorrangstraße1 | Ja | Ja | |
| Vorrangstraße2 | Ja | Ja | |

| | | | |
|---|---|---|---|
| Vorrangstraße3 | Ja | Ja | Trotz perspektivischer Verzerrung |
| Vorrangstraße4 | Ja | Ja | |
| Vorrangstraße5 | Ja | Ja | |
| Vorrangstraße6 | Ja | Ja | |
| Vorrangstraße7 | Ja/Nein | Ja | Die Raute wird korrekt erkannt. Zusätzlich wird jedoch auch ein Dreieck rechts unten gefunden und als Vorrang-Geben-Schild erkannt. |

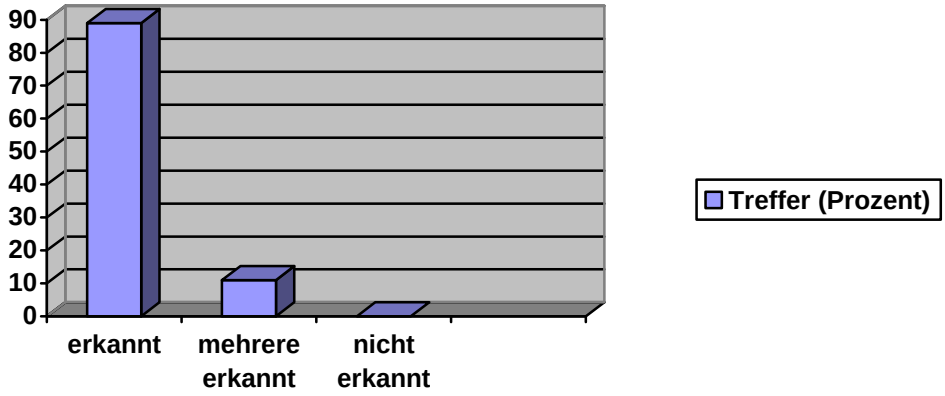
- **Wurde der Typ richtig erkannt?**

**Beschreibung**
Auf jedem Testbild soll der Typ des abgebildeten Verkehrszeichens richtig erkannt und im Bild ausgegeben werden. Unterschieden wird zwischen den Ausgaben „Vorrang geben“, „Einfahrt verboten“ und „Vorrangstraße“.
Beim Testen wurde darauf geachtet, dass immer der richtige Name des Verkehrschildes ausgegeben wird. Weiters wurde überprüft, ob wirklich nur ein Verkehrsschild (das abgebildete) erkannt wird oder ob zusätzliche Formen im Bild gefunden werden, die nicht gefunden werden sollen.

**Ergebnis**
Bei unserem Datensatz wurde auf jedem Testbild das abgebildete Verkehrszeichen gefunden und korrekt erkannt. Auf zwei unserer Bilder wurden jedoch zusätzliche Formen gefunden. In Vorrang_geben5 wird zusätzlich zum Dreieck ein Kreis eingezeichnet und als „Einfahrt verboten“ erkannt. In Vorrangstraße7 wird zusätzlich ein Dreieck rechts unten im Bild gefunden und als „Vorrang geben“ erkannt.




*erkannt: 17/19* *mehrere erkannt: 2/19* *nicht erkannt: 0/19*




*erkannt: 89%* *mehrere erkannt: 11%* *nicht erkannt: 0%*

**Diskussion des Ergebnisses**

Die Erkennung des richtigen Typs scheint gut zu funktionieren und kaum Probleme zu bereiten, insbesondere, wenn das Testbild genau den Vorgaben entspricht.

Dass in Vorrangstraße7 zusätzlich ein Dreieck und infolgedessen ein Vorrang-Geben-Schild erkannt wird, lässt sich möglicherweise damit erklären, dass der Hintergrund des Bildes durch die Äste stark strukturiert ist und so zusätzliche Linien gefunden werden.

In Vorrang_geben5 wird zusätzlich ein Einfahrt-Verboten-Schild erkannt. Hier ist naheliegend, dass der Fehler in der Kreissuche liegt.

- **Stimmt die Position des Verkehrsschildes?**

**Beschreibung**

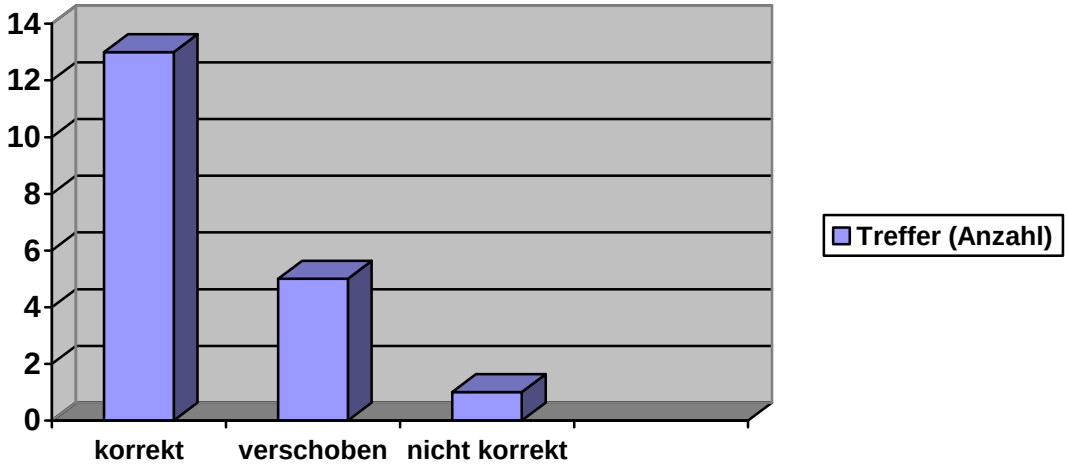
Beim Testen wurde überprüft, ob die Hervorhebung der Konturen des Verkehrsschildes durch Linien auch richtig erfolgt, d.h. ob die Position im Ergebnisbild richtig eingezeichnet wird. Weiters wurde darauf geachtet, ob die x- und y-Werte, die für das Testbild ausgegeben werden und die dessen Position angeben sollen, mit der Lage des Verkehrsschildes im Bild übereinstimmen.

**Ergebnis**

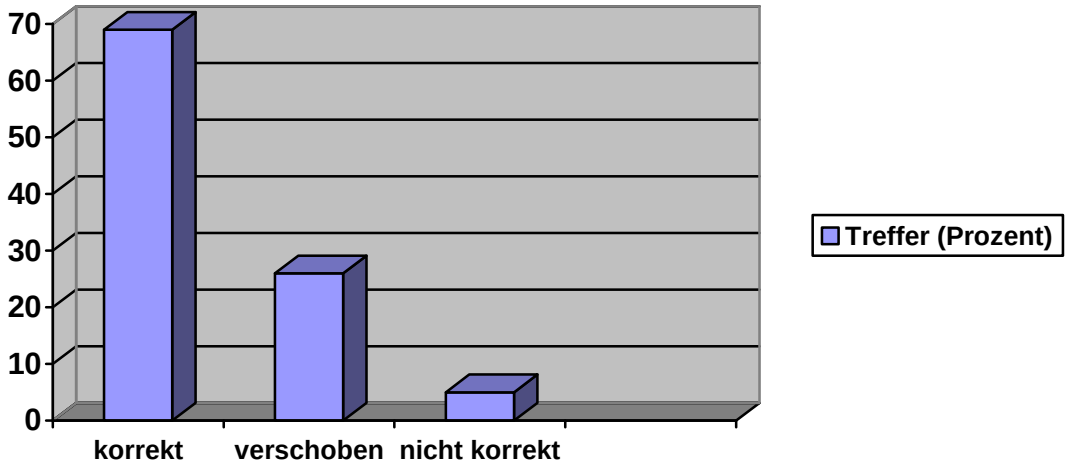
Grundsätzlich hat die Erkennung der Position des Verkehrsschildes mit unseren Testdaten gut funktioniert. Die x- und y-Werte entsprechen der Position des Verkehrsschildes im Bild. Weiters wurde die Form des gefundenen Verkehrsschildes in unseren Testbildern korrekt nachgezeichnet (Dreieck, Raute oder Kreis).

Probleme ergaben sich eher dabei, dass die Konturen des Verkehrsschildes nicht immer exakt um dieses herum gezogen wurden. Insbesondere bei den Kreisen wurde die Kreisposition häufig leicht verschoben eingezeichnet: Einfahrt_verboten3 (eingezeichneter Kreis nach rechts verschoben), Einfahrt_verboten4 (Kreis nach links verschoben), Einfahrt_verboten5 und Einfahrt_verboten6 (leicht nach rechts verschoben), Einfahrt_verboten7 (leicht nach unten verschoben). Die Abweichungen sind jedoch gering, weshalb die Verschiebung von uns nicht als großes Problem gewertet wurde.

Weiters gab es auch bei einem anderen Verkehrszeichen eine leichte Verschiebung. In Vorrang_geben5 wurde das Dreieck nach rechts verschoben und kleiner eingezeichnet. Zusätzlich wurde in diesem Testbild ein Kreis eingezeichnet.




*korrekt: 13* *verschoben: 5* *nicht korrekt: 1* [1]




*korrekt: 69%* *verschoben: 26%* *nicht korrekt: 5%* [1]

[1] Es wurde zwar ein Dreieck eingezeichnet, aber nach rechts verschoben und kleiner als das Verkehrsschild. Zusätzlich wurde ein Kreis eingezeichnet.

**Diskussion des Ergebnisses**

In unseren Testdaten wird jede Form richtig erkannt und eingezeichnet. Die Einzeichnung erfolgt jedoch teilweise mit leichten Verschiebungen. Bei den Einfahrt-Verboten-Schildern ist der eingezeichnete Kreis zwar großteils leicht verschoben, jedoch nicht besonders stark. Das Nachzeichnen der Kreiskonturen hebt immer noch das gefundene Verkehrszeichen hervor und weicht meist nicht weit von der ursprünglichen Position ab, weshalb dieses Verhalten nicht als problematisch eingestuft wurde.

Ein eher größeres Problem stellt das Testbild Vorrang_geben5 dar, in dem die linke Linie des Dreiecks nach rechts verschoben eingezeichnet wird und sich das Dreieck somit verkleinert. Grund dafür sind vermutlich schlechte Lichtverhältnisse. Der Rotfilter filtert bereits Teile des Verkehrsschildes im linken oberen Rand aus. Durch spätere Verarbeitungsschritte fehlt bei der Liniensuche schließlich das linke obere Eck des Vorrang-Geben-Schildes, weswegen später nicht die linke äußere Linie als Rand erkannt wird, sondern die linke innere Linie. Dadurch wird die Dreiecksform verkleinert eingezeichnet.
Problematisch erweist sich auch, dass zusätzlich ein Kreis im Bild gefunden wird.
Da es sich hierbei jedoch um nur ein fehlerhaftes Bild handelt, lässt sich sagen, dass die Testfälle mit korrekter Positionserkennung überwiegen.

- **Welche Bedingungen tragen zu einer erfolgreichen Erkennung bei/halten davon ab?**

**Beschreibung**
Im Zuge unserer Tests stellten wir fest, dass manche Bilder schneller bzw. leichter korrekt verarbeitet wurden als andere. Bei manchen musste erst der Code überarbeitet werden, damit das Verkehrsschild korrekt erkannt wird.

**Ergebnis**
Schwierigkeiten bereiteten vor allem Bilder, die nicht ausreichend ausgeleuchtet waren sowie Bilder mit perspektivischen Verzerrungen. Vorrangstraße3 bereitete erst Probleme, da es mit perspektivischen Verzerrungen aufgenommen wurde und so eigentlich nicht alle Vorrausetzungen eines geeigneten Testbildes erfüllte. Durch Anpassen des Codes wurde jedoch erreicht, dass auch dieses Testbild korrekt erkannt wurde.
Unterschiedliche Beleuchtungen auf den Testbildern führten anfangs dazu, dass der Farbfilter mal besser, mal weniger gut funktionierte. Oftmals wurden Farbbereiche ganz weggefiltert, die für die Verkehrszeichenerkennung wichtig gewesen wären. Durch Anpassen der Parameter im hsv_filter konnte dieser Fehler behoben werden.
Dass schlechte Beleuchtung jedoch nach wie vor Probleme verursachen kann, zeigt Testbild Vorrang_geben5. Hier dürfte eine unzureichende Beleuchtung der Grund sein, warum die linke obere Ecke des Dreiecks ausgefiltert wird, was später zu einem falschen Einzeichnen der Dreieckskonturen führt.

**Diskussion des Ergebnisses**
Zusammenfassend lässt sich hier sagen, dass jene Testbilder, die die von uns festgelegten Vorgaben gut erfüllen, auch mit hoher Wahrscheinlichkeit korrekt von unserem Programm verarbeitet werden können. Achtet man darauf, dass das Testbild gut ausgeleuchtet ist und keine perspektivischen Verzerrungen aufweist (so wie auf die anderen von uns aufgelisteten Punkte in Kapitel 1), sollte das Testbild gut verarbeitet werden können.

# 7. Schlusswort

**Zusammenfassung**

Zusammenfassend lässt sich sagen, dass ein Großteil unsere Testdaten vom Programm korrekt verarbeitet werden konnte. Einzelne Fehler, die aufgetreten sind, ließen sich verbessern durch

- bessere Aufnahmequalität der Bilder
- oder durch Perfektionierung des Codes (siehe „Verbesserungsvorschläge“)

Grundsätzlich haben die von uns gewählten Methoden in den Testläufen zu gewünschten Ergebnissen geführt und sich damit bewährt.
Auch laufzeitmäßig konnten wir unseren Code stark verbessern, wodurch jetzt im Normalfall nur noch wenige Minuten auf das Endergebnis gewartet werden muss.

**Offene Probleme und Fragen**

Offen bleibt das Problem, dass Kreise im Testbild meist verschoben eingezeichnet werden. Auch, wenn es die Auswertung und somit die korrekte Erkennung des Verkehrsschildes in unseren Testdaten nicht beeinflusst hat, kann hier noch etwas verbessert werden. Dazu müsste der Ursache auf den Grund gegangen werden, warum die Kreise verschoben eingezeichnet werden und der Code entsprechend angepasst werden.

**Verbesserungsvorschläge**

Perspektivische Verzerrungen könnten durch ein erweitertes Evaluierungsmodell besser kompensiert werden.
Die Parameter  für das HSV Modell könnten durch eine Histogrammanalyse zielgenauer dynamisch gewählt werden.
Probleme treten auf wenn Hough Lines als zu lang erkannt werden, dazu müsste vor allem die Überprüfung der Kantenüberdeckung ausgebaut werden, die Perfektion der Evaluierung wäre für sich ein unendliches Thema.
Weiteres können durch nahe liegende Schnittpunkte Detektionsregionsverzerrungen entstehen, hier müsste ebenfalls eine verbesserte Evaluierung wie zum Beispiel  eine Flächen Kantenzuordnung implementiert werden.