Neuronale Objekt Erkennung für Android

von

Martin Böckenkamp

Inhaltsverzeichnis

1. Einleitung

Im Bereich der Mechatronik und Automatisierung lassen sich für die meisten Probleme Regelungen entwerfen, die unter der Voraussetzung perfekter Sensorik hervorragend funktionieren würden. Häufig kann diese Voraussetzung jedoch nur eingeschränkt und mit hochspezialisierten Sensoren erfüllt werden, welche zum Teil sehr teuer sind. So ist z.B. für die Bestimmung der Position und Geschwindigkeit eines Flugobjektes im Raum einiges an äußeren Mitteln wie GPS-Satelliten, LPS-Sendern und/oder sehr teuren Laserabstandssensoren notwendig.
Der Mensch hingegen extrahiert solche und noch viel mehr Informationen fast beiläufig und mit hoher Reichweite aus den Bildern, die er mit seinen Augen von seiner Umwelt empfängt.
Aufgrund des hohen Informationsgehalts von Bildern, lassen sich Bildquellen wie Kameras als Universalsensoren verstehen, die Dank der heute verfügbaren, sehr günstigen, energieeffizienten und hohen Rechenleistung ein enormes Potential haben.
Der Bereich des maschinellen Sehens hält bereits Einzug in die Automobilbranche z.B. im Bereich der Verkehrssicherheit, was man bereits an in Serienfahrzeugen integrierten Verkehrzeichen-Erkennungssystemen und Spurassistenten sehen kann.
Dieses Projekt hatte zum Ziel, ein einfaches Verkehrszeichen-Erkennungssystem in Form einer Android-App zu realisieren und so die grundlegenden Prinzipien des maschinellen Sehens zu erarbeiten. Die Umsetzung war erfolgreich und resultierte in einer Android-App, die (mit Einschränkungen bei der Erkennungsgenauigkeit und Einsatztauglichkeit) in der Lage ist, Geschwindigkeitsbegrenzungen der Schilder von 10 bis 100 km/h in 10er Schritten zu erkennen.
Das Programm wurde mit Processing 3.0a5 im Java-Mode mit Hilfe der Video-Library entwickelt und später mit Hilfe der Ketai-Library im Android-Mode auf Android portiert.

2. Initialisierung der Kamera

2.1 Webcam

Mit Processing 3.0a5 ließ sich die Webcam mit Hilfe der „Video“-Bibliothek über eine Instanz der Klasse „Capture“ ansprechen. Dazu muss das Objekt erzeugt und dem Konstruktor das PApplet „this“ und die Auflösung übergeben werden, über die Methode start() wird die Kamera schließlich eingeschaltet. Das Auslesen des Bildes erfolgt über die Methode read(), die am besten über den Eventhandler „captureEvent“ aufgerufen wird. Die Capture-Klasse erbt von Pimage und kann daher wie der Standard-Bildtyp in Processing behandelt werden.

```
import processing.video.*;
Capture video;

void setup() {
       video = new Capture( this, ResX, ResY );
}

void draw() {
       image( video, 0, 0 );
```

```
}

void captureEvent(Capture c) {
  c.read();
}
```

2.2 Handy-Kamera

Die Ansteuerung der Handy-Kamera unter Android ließ sich mit Hilfe der „Ketai“-Bibliothek auf analogem Wege wie bei der Webcam umsetzen. Nur die Bezeichnungen sind etwas anders.

```
import ketai.camera.*;
KetaiCamera video;

void setup() {
       video = new KetaiCamera( this, ResX, ResY, Framerate );
}

void draw() {
       image( video, 0, 0 );
}

void onCameraPreviewEvent() {
  video.read();
}
```

3. Beschreibung der Problemlösung

Theoretisch bräuchte man zur Erkennung von Objekten in einem Bild „einfach nur“ einen dem menschlichen Gehirn nachempfundenen visuellen Kortex aus künstlichen Neuronen aufzubauen; was aber, selbst wenn man den dessen genauen Aufbau kennen würde, an der begrenzten Rechenleistung des Handy-Prozessors scheitern würde.
In der Praxis ist es daher sinnvoll die Bildinformationen zuerst stark zu reduzieren, dann Kandidaten zu bestimmen und erst dann eine komplexere Mustererkennung anzuwenden.
Um das Problem zu lösen, wurde wie folgt vorgegangen:
Zunächst wird das Kamerabild komplett nach Rotwerten gefiltert, da die für uns interessanten Verkehrszeichen sich vor allem durch den roten Ring von ihrer Umwelt abheben. Im rot-gefilterten Bild wird dann durch eine Maximum-Suche der höchste Rotwert ermittelt, um daraufhin mit einem Schwellwert-Filter das Bild zu binarisieren - d.h. auf zwei Farben zu reduzieren. Der letzte Schritt ist notwendig, damit der anschließende Konturfolge-Algorithmus funktioniert. An dieser Stelle sind im Bild nur noch wenige große, zusammenhängende Strukturen vorhanden, welche nun über einen Kantenfilter auf ihre Konturen reduziert werden.
Nun wird das Bild in Zeilen mit Abständen von 20 Pixeln durchlaufen und nach Konturpixeln gesucht. Wird ein Konturpixel gefunden, merkt sich das Programm die Startposition und folgt der Kontur mit Hilfe eines binären Konturfolge-Algorithmus und zieht dabei ein rechteckiges Fenster auf, welches über das x,y-Minimum und x,y-Maximum definiert ist und den im folgenden zu untersuchenden Bildbereich markiert.

Nun wird ein Viertel der Bildfläche in der Mitte des Bildbereichs herausgeschnitten (die Ziffern), in Graustufen umgewandelt, auf 16x16 Pixel skaliert und in die Eingangsneuronen eines dreischichtigen Perzeptrons geladen, welches anschließend den Bildausschnitt klassifiziert. Jedes Ausgangsneuron steht für eine Geschwindigkeit, die nun durch Suche des aktivsten Neurons ermittelt wird. Um Fehlerkennungen zu reduzieren, wird neben dem aktivsten Neuron auch das zweitaktivste Ausgangsneuron gesucht. Überschreitet dessen Aktivität eine gewisse Schwelle, kann davon ausgegangen werden, dass die Erkennung nicht eindeutig war. Wurde nun ein Schild erkannt, wird der entsprechende Wert in einem zeitlichen Histogramm der erkannten Schilder um einen bestimmten Wert erhöht. Das Histogramm hat einen oberen Grenzwert und wird mit der Zeit immer wieder langsam abgebaut, wobei das letzte verbliebene Maximum erhalten bleibt, um das zuletzt gefundene Schild zu speichern und somit anzuzeigen. Nur wenn ein neues Maximum durch mehrere erfolgreiche Erkennungen eines anderen Schildes auftritt, fällt das letzte Maximum wieder ganz auf Null.

Abb. 1: Ergebnis einer erfolgreichen Kandidatensuche und Mustererkennung

4. Vorverarbeitung der Bilder

4.1 Rotwert-Filterung

Da die gesuchten Verkehrszeichen durch ihren roten Ring im Bild auffallen, wird das Bild nach roten Farbwerten gefiltert. Zum einen besteht die Möglichkeit das Bild vom (3x8 Bit) RGB- in den sogenannten HSV-Farbraum zu konvertieren, worin der Farbwert direkt als einzelne Komponente „H“ gegeben ist und so einfach per Intervall gefiltert werden könnte. Anschließend müsste eine Rückkonvertierung nach RGB erfolgen. Diese Vorgehensweise hat den Nachteil, dass sie sehr rechenintensiv ist. Daher wurde nach einem Verfahren gesucht, welches direkt auf die RGB-Werte angewendet werden kann. Eine Beobachtung des Verhaltens des RGB/HSV-Farbwählers des Grafikprogramms GIMP zeigte, dass sich Rottöne durch folgende Beziehung von den anderen Farbtönen abgrenzen lassen:

r_{neu} `= ((r > g) && (r > b)) ? ( (g >= b) ? (r-g)-(g-b) : (r-b)-(b-g) ) : 0;`

Die erste Bedingung bedeutet, dass etwas nur dann rot erscheint, wenn die Rot-Komponente größer als die Blau- und die Grün-Komponente ist. Dies garantiert jedoch noch nicht, dass es auch wirklich Rot ist - wie z.B. bei Orange. Die nächste Bedingung prüft lediglich, ob Grün oder Blau stärker ist und berechnet dann danach die Auswirkung auf den Rotwert.
Die Überlegung hinter der Rechnung besteht darin, beispielhaft für den Fall, dass Grün > Blau sei, dass zunächst die zweitstärkste Komponente Grün dem Rot entgegenwirkt `(r-g),` da ein zunehmendes Grün von Braun, über Orange zu Gelb führt. Der zweite Term -(g-b) rührt daher, dass die Differenz von Grün und Blau für eine zusätzliche Verschiebung des Farbwertes hin zu gelb sorgt und folglich zusätzlich dem Rotwert entgegenwirkt.
Da die Werte ins Negative gehen können, wird im Anschluss noch eine Intervallbeschränkung vorgenommen, sodass Ergebnisse stets zwischen 0 und 255 liegen. Das Bild weist nach der Filterung nur noch im Rotkanal Werte auf, der Grün- und Blau-Kanal sind für alle Pixel Null.

Abb. 2: Kamerabild (links) und nach Rotwert gefiltertes Bild (rechts, zum besseren Erkennen gammakorrigiert)

4.2 Schwellwert-Filterung

Die Binarisierung bzw. Reduktion des Bildes auf zwei Farbwerte wird erreicht, indem eine Grenze bei 0,25 · rot_{max} festgelegt wird. Alle Werte die gleich sind oder darüber liegen werden auf 255 gesetzt, alle darunter auf 0. Dem geht eine Maximumsuche von rot_{max} voraus. Beide Verfahren werden pixelweise auf den Rot-Kanal des zuvor nach Rotwert gefilterten Bildes angewendet.

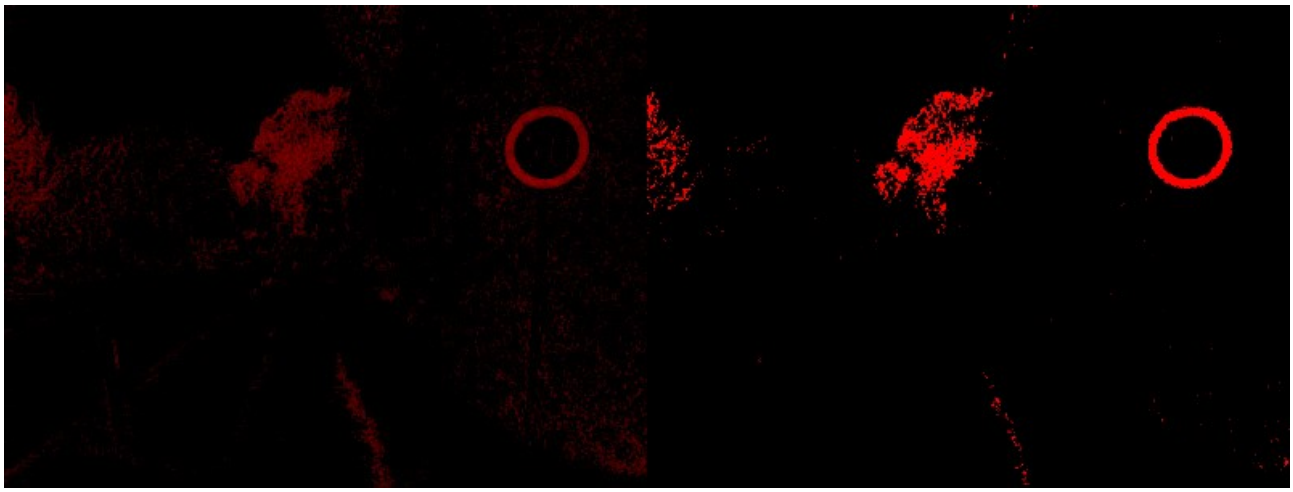

Abb. 3: Rotwert gefiltertes Bild (links), binarisiertes Bild (rechts)

4.3 Kantenfilter

Auf das binarisierte Bild wird nun eine sogenannte Kantenfilter-Faltungsmatrix angewendet.
Eine Faltungsmatrix bezeichnet in der Bildverarbeitung eine quadratische Matrix, die keine Matrix im üblichen mathematischen Sinne darstellt und als Filtermaske verwendet wird.
Sie wird mit ihrem Mittelpunkt über den zu berechnenden Pixel an der Stelle (x,y) des Bildes gelegt und alle Einträge mit dem jeweils darunter liegenden Pixelwert multipliziert und summiert, wodurch sich der neue Pixelwert ergibt. Mit Pixelwert ist der Helligkeitswert bzw. in diesem Fall nur der Wert des Rotkanals gemeint.

$$I^*(x,y)=\sum_{i=1}^{n}\sum_{j=1}^{n} I(x+i-a,y+j-a)\cdot k(i,j)$$

a - Abstand des Matrixmittelpunktes vom Rand
n - Zahl der Zeilen und Spalten

Eine Kantenfilter-Matrix entspricht der zweiten Ableitung der Pixelwerte nach allen Richtungen.
Für die Ableitung nach einer einzelnen Richtung betrachtet ergibt sich die erste Ableitung aus der Differenz der Werte zweier aufeinander folgender Pixel:

$$\frac{\partial f}{\partial x} = f(x+1)-f(x)$$

folglich ergibt sich die zweite Ableitung zu:

$$\frac{\partial^2 f}{\partial x^2} = (f(x+1) - f(x)) - (f(x) - f(x-1))$$
$$= f(x-1) + f(x+1) - 2 \cdot f(x)$$

Als Faltungsmatrix für die x-Richtung:

0	0	0
1	-2	1
0	0	0

Nimmt man die y-Richtung und die beiden Diagonalen Richtungen ergeben sich analoge Faltungsmatrizen, die zusammen überlagert die Kantenfilter-Faltungsmatrix ergibt:

1	1	1
1	-8	1
1	1	1

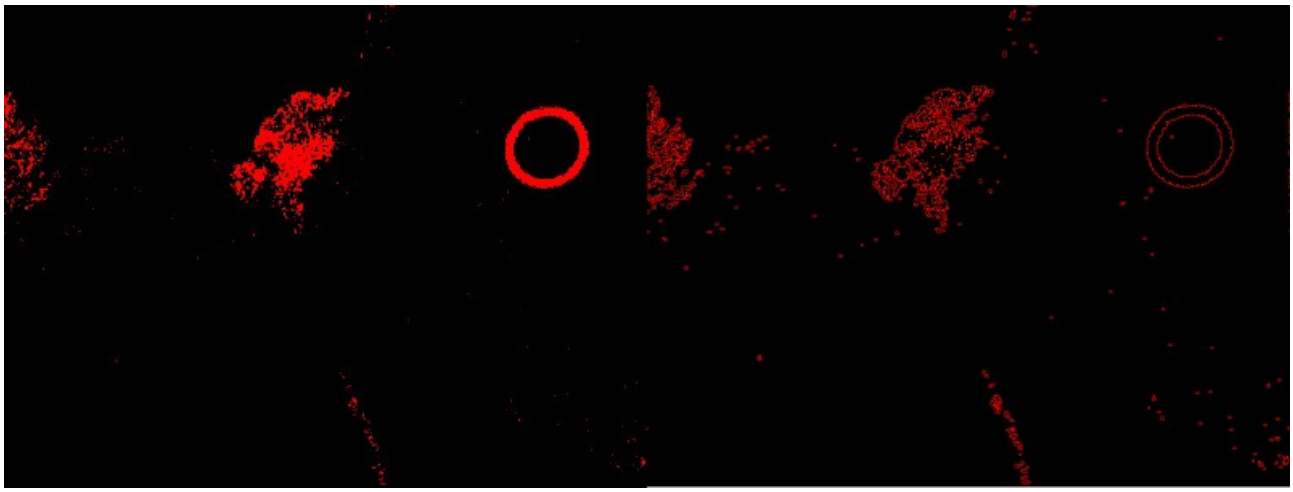

Abb. 4: binarisiertes Bild (links), Kanten gefiltertes Bild (rechts)

4.4 Kandidatensuche und Konturverfolgung

Zum Auffinden von Kandidaten wird das kantengefilterte Bild in Zeilen mit 20 Pixeln Abstand durchlaufen. Wird dabei ein Kantenpixel getroffen, beginnt der Konturfolge-Algorithmus.
Der binäre Konturfolge-Algorithmus benötigt nur zwei einfache Regeln, um einer geschlossenen Kontur zu folgen: (1) ist es ein Kantenpixel, dann drehe die Bewegungsrichtung um 90° im Uhrzeigersinn (2) ist es ein inaktiver bzw. dunkler Pixel, dann drehe die Bewegungsrichtung um 90° gegen den Uhrzeigersinn. Da sich das Verfahren an manchen Stellen bis zu 3 Schritte weit über dunklen Pixeln bewegen kann bis es wieder auf einen Kantenpixel der selben Kontur trifft, gibt es eine Lebenszeitvariable, die bei jedem Kantenpixel auf 5 gesetzt und auf jedem dunklen Pixel dekrementiert wird. Erreicht diese Variable den Wert 0, ist das Ende der Kontur erreicht.
Um geschlossenen Konturen nicht endlos zu folgen, wird in einem Abstand von 4 Pixeln die Kontur gelöscht. Auch hier wirkt sich die Lebenszeitvariable aus, da sie dafür sorgt, dass nach Erreichen des Endes der Kontur auch der letzte Pixel gelöscht wird.
Für das durchlaufene Gebiet werden während der Kantenverfolgung die Minimal- und Maximal-Werte der x- und y-Position ermittelt, welche so den rechteckigen Bildausschnitt enthalten, der die Kontur umschließt.
Wurde das Ende einer Kontur erreicht, werden mit drei Kriterien unsinnige Kandidaten aussortiert.
Ist der letzte Pixel innerhalb eines geringen Abstands zum Startpixel, wurde eine geschlossene Kontur gefunden - was die erste Bedingung für einen Kandidaten ist. Eine geschlossene Kontur ist z.B. am Bildrand nicht garantiert und muss daher überprüft werden. Das zweite Kriterium ist, dass das Seitenverhältnis des gefundenen Bildbereichs nicht größer als 2:1 bzw. nicht kleiner als 1:2 sein darf, da die Verkehrszeichen quadratisch sind und bei zu schräger Ansicht (und entsprechendem Seitenverhältnis) zunehmend schlecht erkannt werden. Als letztes Kriterium muss der gefundene Bildbereich mindestens 32x32 Pixel groß sein, um überhaupt noch etwas erkennen zu können. Sind diese Kriterien erfüllt, wird der Kandidat dem Perzeptron übergeben.

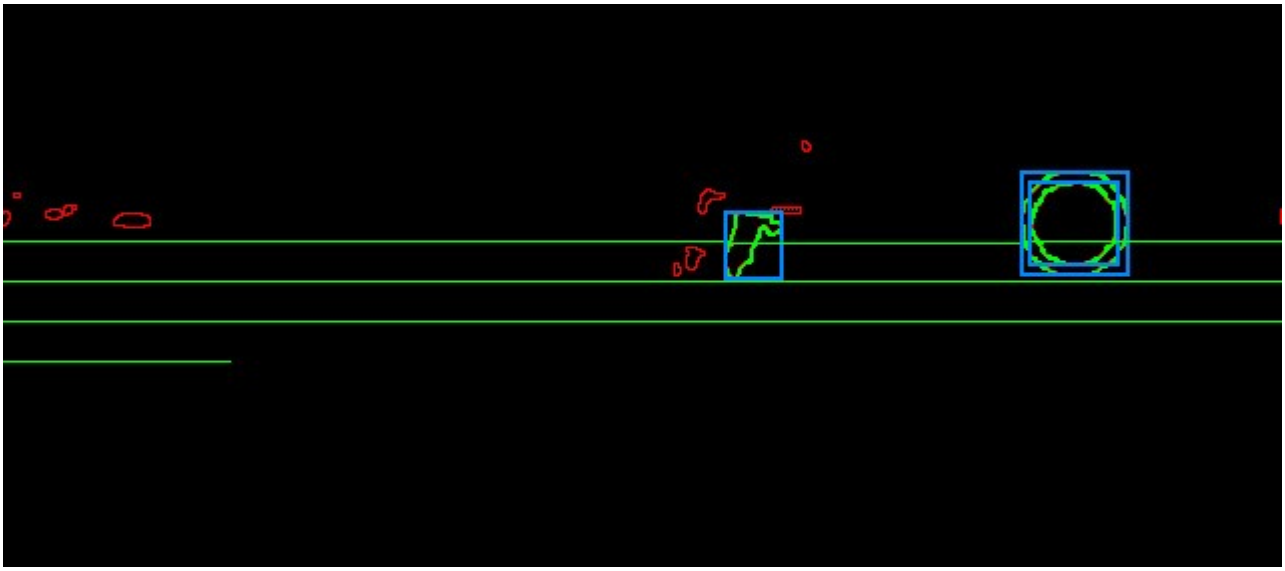

Abb. 5: Visualisierung des Konturverfolgungs-Algorithmus

5. Mustererkennung mit Perzeptron

Um das ggf. vorhandene Zahlenmuster des gefundenen Kandidaten zu analysieren, kommt ein dreischichtiges Perzeptron mit 256 Eingangs-Neuronen, 10 Zwischenschicht-Neuronen und 10 Ausgangsschicht-Neuronen zum Einsatz. Ein vorheriger Versuch mit einem zweischichtigen Perzeptron machte es unmöglich die 10 verschiedenen Schilder zu unterscheiden.

Das Perzeptron wurde mit jeweils 4 verschiedenen Bildern einer Klasse und mit drei Gegenbeispielen, bei denen der Ausgang jeweils komplett auf Null liegen sollte, mit dem Backpropagation-Algorithmus trainert. Als Lernfaktor wurde 1.0 · (1-sqrt(i / Trainingsdurchläufe)) verwendet (i - Schleifenzähler), was sich als sehr effektiv im Vergleich zu konstanten oder linear abnehmenden Lernfaktoren erwies. Das Training umfasste 10.000 Durchläufe. Die ermittelten Gewichte wurden als float-Array in Java-Syntax in Dateien geschrieben und anschließend direkt in das Programm eingebunden.

Abb. 6: Kandidaten-Bild (links), vergrößertes Eingangsbild des Perzeptrons (rechts)

Zunächst wird nur ein Viertel der Bildfläche der Mitte des Kandidaten-Bildausschnitts ausgeschnitten, da dieser Bereich im Falle eines Geschwindigkeitsbegrenzungsschilds die Zahlen enthält. Um diesen neuen Bildausschnitt in die Eingangs-Neuronen des Perzeptrons zu laden, wurde eine spezielle Methode setzeEingang() geschrieben, die das übergebene Bild in ein invertiertes Graustufenbild umrechnet, wobei bunte Pixel, d.h. Pixel, bei denen sich r,g,b um mehr als „48“ unterscheiden, auf schwarz gesetzt werden. Dieses wird auf 16x16 Pixel mit Hilfe eines flächenbasierenden Verkleinerungsverfahrens skaliert. Das Verkleinerungsverfahren musste trotz der Existenz der PImage-Methode resize() implementiert werden, weil resize() unter Android stets zu Abstürzen führte. Nach dem Laden der Eingangs-Neuronen wird das Perzeptron durchgerechnet, wobei die 10 Ausgangsneuronen von 0 bis 9 den unterschiedlichen Geschwindigkeitsbegrenzungsschildern von 10 bis 100 km/h zugeordnet sind. Wird ein Schild sauber erkannt, nimmt das entsprechende Ausgangs-Neuron im Idealfall einen Wert von 1,0 an, wobei die restlichen Neuronen auf 0,0 liegen sollten.

6. Histogramm der Ausgangs-Neuronen

Um nun ein Schild zu identifizieren, wird eine doppelte Maximumsuche über die Ausgangs-Neuronen durchgeführt, wobei nur Neuronen mit einer Aktivität größer als 0,7 beachtet werden. Ist das zweitgrößte Maximum größer als 0,2 , kann davon ausgegangen werden, dass irgendein fremdes Muster diesen Ausgang hervorgerufen hat. Wurde nun ein Muster erkannt, wird im Histogramm-Array, in welchem jeder Eintrag einem Ausgangs-Neuron entspricht, die entsprechende Stelle um 3 bis zu einer oberen Schranke von 30 erhöht. Mit jedem

Programmzyklus wird jeder der Einträge des Histogramms jeweils um 1 vermindert, sodass ein zeitliches Histogramm der erkannten Muster entsteht. Der Maximal-Wert im Histogramm stellt somit das wahrscheinlichste Muster dar. Fällt der letzte Maximal-Wert des Histogramms unter 6, wird dieser wieder auf diesen Wert erhöht, solange bis ein anderes Maximum durch eine mindestens dreilmalige Erkennung diesen überbietet. Auf diese Weise bleibt das zuletzt erkannte Bild auch während der Zeit erhalten, in der keine weiteren Schilder erkannt werden.

7. Fazit

Das Verfahren funktioniert mit manchen Schildern (z.B. 20 km/h und 40 km/h) sehr gut und mit einigen weniger gut, d.h. nur bei sehr genauer Ausrichtung der Muster. Die Ursache liegt vermutlich zu einem nicht unerheblichen Teil im Training des Perzeptrons, da nur eine kleine Zahl von Trainingsmustern präsentiert wurde. Das Training müsste am besten mit der Kamera erfolgen, um möglichst unterschiedlichste Perspektiven der Muster zu erhalten. Es gibt noch einiges an Verbesserungspotential bei der Geschwindigkeit, so müsste der Kantenfilter nicht auf das gesamte Bild angewendet werden, sondern nur lokal auf dem jeweils vom Konturfolge-Algorithmus bearbeiteten Pixel. In der Methode zum Laden des Bildausschnitts in die Eingangs-Neuronen, müsste noch ein automatischer Weißabgleich implementiert werden, um den Kontrast der Eingangsmuster zu erhöhen.
Bei der Vorverarbeitung bzw. Kandidatensuche könnte noch eine Konturformanalyse eingebaut werden, die zumindest gerade Linien (konstante Steigungen) in den Konturen erkennt und so ein weiteres Ausschlusskriterium zur Verfügung stellt.