

Studienarbeit

Verkehrszeichenerkennung

Autoren: Alexander Melde, Stefan Schneider

Betreuer: Martin Kissel



Bearbeitungszeitraum: 16.10.2017 – 04.06.2018

Abgabedatum: 04.06.2018

Studiengang: B. Sc. Angewandte Informatik – TINF 15 K

Verkehrszeichenerkennung

Alexander Melde, Stefan Schneider

Studienarbeit

Kurzbeschreibung

Selbstfahrende Fahrzeuge sind schon lange keine Zukunftsmusik mehr. Eine zentrale Voraussetzung für das autonome Fahren ist ein Verständnis der Umgebung und der aktuell geltenden Beschilderung.

In dieser Arbeit werden zunächst Grundlagen, die zur Erkennung von Verkehrszeichen benötigt werden erläutert. Anschließend wird ein, im Rahmen dieser Arbeit entwickeltes, Programm dokumentiert, das Verkehrszeichen in Videos erkennt und klassifiziert. Für die Implementierung werden klassische Methoden der digitalen Bildverarbeitung mit modernen Machine Learning Frameworks kombiniert.

Das entwickelte Programm wird anschließend mit bereits verfügbaren Systemen verglichen, um die Vor- und Nachteile von verschiedenen Technologie-Ansätzen herauszuarbeiten.

Abstract

Self-driving vehicles are no longer a dream of the future. A central requirement for autonomous driving is an understanding of the environment and the currently effective signs.

This paper first teaches the essentials that are needed to detect traffic signs. Subsequently, a program, that was written as a part of this work is documented. The program recognizes and classifies traffic signs in videos. For this purpose, traditional methods of digital image processing will be combined with modern machine learning frameworks.

The developed program is then compared with systems released in the past to highlight the advantages and disadvantages of different technology approaches.

Informationen

| | | |
|------------------|---|-------------------------|
| Autoren | Melde, Alexander | 7939560 |
| | Schneider, Stefan | 5280242 |
| Betreuer | Kissel, Martin | martinkissel@sharing.de |
| Studiengang/Kurs | B. Sc. Angewandte Informatik – Kommunikationsinformatik TINF15K | |
| Titel der Arbeit | Verkehrszeichenerkennung | |
| Anlass | Studienarbeit, 3. Studienjahr | |

Ehrenwörtliche Erklärung

Wir versichern hiermit, dass wir die Studienarbeit mit dem Thema: „Verkehrszeichenerkennung“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt. *

** falls beide Fassungen gefordert sind*

Ort, Datum

Unterschrift Alexander Melde

Ort, Datum

Unterschrift Stefan Schneider

Inhaltsverzeichnis

| | |
|---|-----|
| Kurzbeschreibung | I |
| Abstract | I |
| Informationen | II |
| Ehrenwörtliche Erklärung | II |
| Inhaltsverzeichnis | III |
| Abbildungsverzeichnis | V |
| Tabellenverzeichnis | VI |
| Weblinkverzeichnis | VI |
| Abkürzungsverzeichnis | VI |
| | |
| 1 Einleitung | 7 |
| 2 Vorbetrachtung | 8 |
| 2.1 Marktanalyse | 9 |
| 2.1.1 Kommerzielle Systeme | 9 |
| 2.1.2 Akademische Projekte | 10 |
| 2.1.3 Vergleich und Auswertung | 11 |
| 2.2 Anforderungsdefinition | 11 |
| 3 Theoretische Grundlagen | 12 |
| 3.1 Standardisierung von Verkehrszeichen | 12 |
| 3.1.1 Farbnorm | 12 |
| 3.1.2 Verkehrszeichentypen | 13 |
| 3.1.3 Größe | 16 |
| 3.1.4 Sichtbarkeit und Regeln zur Aufstellung | 17 |
| 3.2 Digitale Bildverarbeitung | 18 |
| 3.2.1 Bildverarbeitung | 18 |
| 3.2.2 Bildanalyse | 18 |
| 3.2.3 Bilderkennung | 19 |
| 3.2.4 Bildklassifikation | 19 |
| 3.2.5 Tracking | 19 |
| 3.3 Künstliche Intelligenz | 20 |
| 3.3.1 Machine Learning | 21 |
| 3.3.2 Künstliche neuronale Netze | 22 |
| 3.3.3 Convolutional Neural Network | 23 |
| 4 Projektplanung | 25 |
| 4.1 Benutzeroberfläche | 26 |
| 4.2 Erkennung und Klassifikation | 27 |
| 4.2.1 Formerkennung | 28 |
| 4.2.2 Bildklassifikation | 28 |
| 4.3 Ergebnisdarstellung | 28 |
| 4.4 Risikoanalyse | 29 |
| 4.4.1 Risiken bei der Formerkennung | 29 |
| 4.4.2 Risiken bei der Bildklassifikation | 30 |
| 5 Projektdurchführung | 32 |
| 5.1 Benutzeroberfläche | 32 |
| 5.2 Rahmenprogramm | 33 |
| 5.3 Erkennung von Verkehrszeichen | 34 |

| | | |
|--------|---|----|
| 5.3.1 | Kreis-Detektion | 34 |
| 5.3.2 | Quadrat-Detektion | 36 |
| 5.4 | Klassifikation von Verkehrszeichen | 38 |
| 5.4.1 | Klassenstruktur | 39 |
| 5.4.2 | Gewinnung von Trainingsdaten | 40 |
| 5.4.3 | Trainieren der gewonnenen Trainingsdaten | 42 |
| 5.4.4 | Skript zur Klassifizierung eines Verkehrszeichens | 43 |
| 5.5 | Verfolgung von Verkehrszeichen | 43 |
| 5.6 | Software-Test | 44 |
| 5.6.1 | Detektionsrate | 44 |
| 5.6.2 | Fehlerrate der Detektion | 44 |
| 5.6.3 | Erfolgsrate der Klassifizierung | 44 |
| 5.6.4 | Schwellwert zur sicheren Klassifikation | 45 |
| 5.6.5 | Eindeutigkeit der Klassifikationsergebnisse | 45 |
| 5.6.6 | Zeitmessung | 45 |
| 5.6.7 | Funktionstest | 46 |
| 5.6.8 | Testergebnisse | 47 |
| 5.7 | Schwierigkeiten bei der Umsetzung | 48 |
| 5.8 | Veröffentlichung und Lizenz | 49 |
| 5.9 | Download, Bedienung und Installation | 50 |
| 5.9.1 | Einrichten der Entwicklungsumgebung | 50 |
| 5.9.2 | Ausführen | 51 |
| 5.9.3 | Kompilieren | 51 |
| 6 | Erweiterungsmöglichkeiten | 52 |
| 6.1 | Optimierungsmöglichkeiten | 53 |
| 6.1.1 | Parallellisierung und Einsatz von Grafikkarten | 53 |
| 6.1.2 | Verbessern der Kameraqualität | 54 |
| 6.1.3 | Berücksichtigung der Häufigkeit von Verkehrszeichen | 54 |
| 6.1.4 | Erhöhen der Trainingsdaten | 54 |
| 6.1.5 | Einsatz von kontinuierlichem Lernen | 54 |
| 6.1.6 | Beachten der Kameraposition | 55 |
| 6.1.7 | Einsatz eines leichtgewichtigeren Frameworks | 55 |
| 6.1.8 | Einsatz eines leichtgewichtigeren Klassifikations-Modells | 56 |
| 6.1.9 | Unterscheidung mehrerer Klassifikationsmodelle | 56 |
| 6.1.10 | Detektion mittels Machine Learning | 56 |
| 6.2 | Funktionserweiterungen | 57 |
| 6.2.1 | Gültigkeit von Schildern | 57 |
| 6.2.2 | Neue Verkehrszeichen für autonomes Fahren | 57 |
| 6.2.3 | Kennzeichenerkennung | 58 |
| 6.2.4 | Anwendung direkt im Fahrzeug | 59 |
| 7 | Vergleich | 60 |
| 8 | Fazit | 61 |
| | Literaturverzeichnis | 62 |
| | Anhang | 67 |

Abbildungsverzeichnis

| | |
|---|----|
| Abb. 1: Gefahrenzeichen | 13 |
| Abb. 2: Vorfahrtzeichen | 14 |
| Abb. 3: Verbots und Beschränkungszeichen | 15 |
| Abb. 4: Gebotszeichen..... | 15 |
| Abb. 5: Minimales neuronales Netz | 22 |
| Abb. 6: Convolutional Neural Network Layer..... | 23 |
| Abb. 7: Beispielhafte Berechnung der Kernel-Ausgangswerte | 24 |
| Abb. 8: Struktogramm zur praktischen Implementierung | 25 |
| Abb. 9: Erstes GUI Mockup..... | 26 |
| Abb. 10: Probleme bei der Erkennung von Verkehrsschildern | 29 |
| Abb. 11: Grafische Oberfläche des entwickelten „Traffic Sign Detection“-Programms | 32 |
| Abb. 12: HSV-Farbraum | 34 |
| Abb. 13: Verschiedene Bearbeitungsschritte der Kreiserkennung | 35 |
| Abb. 14: Erkennung von möglichen Verkehrsschildern anhand der Form | 36 |
| Abb. 15: Verschiedene Arbeitsschritte der Quadrat-Detektion..... | 36 |
| Abb. 16: Ablauf einer Bildklassifizierung | 38 |
| Abb. 17: Ordnerstruktur für Beispielbilder | 39 |
| Abb. 18: Zugeschnittene mögliche Verkehrszeichen vor der Klassifizierung..... | 40 |
| Abb. 19: Konvertieren der .ppm-Bilder zu .jpg..... | 41 |
| Abb. 20: Start des Modell-Trainings..... | 42 |
| Abb. 21: Meldung am Ende des Modell-Trainings | 42 |
| Abb. 22: Datenstruktur für Schilder | 43 |
| Abb. 23: Beispielszene aus dem Test-Video | 47 |
| Abb. 24: Aufnahme eines Verkehrsschildes aus verschiedenen Entfernungen..... | 48 |
| Abb. 25: Lizenz der Studienarbeit | 49 |
| Abb. 26: Aufruf der Verkehrszeichenerkennung..... | 51 |
| Abb. 27: Kompilieren der Verkehrszeichenerkennung | 51 |
| Abb. 28: Beispiel für einen Filter mit relevanten Regionen | 55 |
| Abb. 29: Installation des OpenALPR Docker-Containers | 58 |
| Abb. 30: Aufruf des OpenALPR Kommandozeilenprogramms | 58 |

Tabellenverzeichnis

| | |
|---|----|
| Tabelle 1: Marktanalyse – Kommerzielle Systeme..... | 9 |
| Tabelle 2: Marktanalyse – Akademische Projekte | 10 |
| Tabelle 3: Farbwerte von Verkehrszeichen | 12 |
| Tabelle 4: Größen von Verkehrszeichen | 16 |
| Tabelle 5: Zuordnung der Größen zur Höchstgeschwindigkeit | 16 |

Weblinkverzeichnis

| | |
|--|----|
| Link 1: appJar GUI-Framework http://appjar.info/ | 26 |
| Link 2: OpenCV Bibliothek https://opencv.org/ | 28 |
| Link 3: TensorFlow Framework https://tensorflow.org/ | 28 |
| Link 4: GTSRB Dataset: http://benchmark.ini.rub.de/Dataset/GTSRB_Final_Training_Images.zip | 41 |
| Link 5: GNU GPL v.3 Lizenz https://www.gnu.org/licenses/gpl | 49 |
| Link 6: Quelltext-Download https://github.com/AlexanderMelde/Verkehrszeichenerkennung | 50 |
| Link 7: Build-Download http://www.lehre.dhbw-stuttgart.de/~it15111/vze/ | 50 |
| Link 8: virtualenv-Tutorial https://www.geeksforgeeks.org/python-virtual-environment/ | 50 |
| Link 9: Tk-Framework Installationsanleitung http://www.tkdocks.com/tutorial/install.html | 50 |
| Link 10: Docker Installation: https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/ 58 | |

Zuletzt abgerufen am: 29.05.2018

Abkürzungsverzeichnis

| Abkürzung | Bedeutung |
|--------------|--|
| ALPR | Automatic License Plate Recognition |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit (Prozessor) |
| CV | Computer Vision |
| GPU | Graphic Processing Unit |
| GTSRB | German Traffic Sign Recognition Benchmark |
| GUI | Graphical User Interface |
| KI | Künstliche Intelligenz |
| RGB | Rot-Grün-Blau |
| SSO | Single Shot MultiBox Detector |
| StVO | (allgemeine Verwaltungsvorschrift zur) Straßenverkehrs-Ordnung |
| YOLO | You Only Look Once |

1 Einleitung

Die Autonomie zieht in immer mehr Bereiche ein, die Jahrzehnte lang Menschen und deren manuellen Methoden vorbehalten waren. Forschung, komplexe Produktionen, das Militär, der Verkehr oder auch die Medizin sind nur einige Bereiche, die von diesen Systemen profitieren.¹ Und der Boom in der Forschung der Künstlichen Intelligenz (KI) verspricht eine neue Technologie, welche vielleicht innerhalb von wenigen Jahren unsere aktuellen Systeme überholt:²



The next decade will be defined by the automation of the automobile, and autonomous vehicles will have as significant an impact on society as Ford's moving assembly line did 100 years ago.

- Mark Fields, damaliger Präsident und CEO der Ford Motor Company (Ford Motor Company, 2017)

Treibende Kräfte der Autonomie sind die Automobilhersteller, die schon seit vielen Jahren immer mehr Sicherheitstechnik und Assistenzsysteme für ihre Fahrzeuge entwickeln.³ Miteinander verbunden ermöglichen diese auch heutzutage schon ein teilautonomes Fahren.⁴ Dennoch sind die Systeme noch nicht komplett ausgereift, was sich beispielsweise anhand der immer wieder vorkommenden Unfälle mit autonomen Fahrzeugen beweisen lässt.⁵ Auch der als Visionär und Vorreiter bekannte CEO des Automobilherstellers Tesla, Elon Musk, korrigierte eine frühere Aussage von 2015, in der er die These aufstellte, dass bereits Ende 2017 eine komplette Autonomie erreicht werden würde. Heute geht er davon aus, dass erst 2020 oder 2021 Autos komplett autonom fahren.⁶

Für selbstfahrende Autos essentiell ist eine zuverlässige Wahrnehmung der Außenwelt um das Fahrzeug herum. Eine wichtige Rolle spielt hierbei die Erkennung von Verkehrszeichen, da diese dem Fahrzeug Geschwindigkeiten vorgeben, die Vorfahrt regeln und auf Gefahren hinweisen.

Ein weiteres Anwendungsgebiet für computergestützte Verkehrszeichenerkennungs-Systeme ist die automatisierte Kontrolle des Vorhandenseins von Schildern in Städten und auf Autobahnen.⁷

Diese Arbeit beschreibt den theoretischen Aufbau einer Verkehrszeichenerkennung und zeigt auf, welche Probleme bei der Erkennung auftreten können und an welchen Stellen heutige Erkennungssysteme an ihre Grenzen stoßen.

Die anschließend dokumentierte praktische Implementierung basiert auf einer Kombination von klassischen Methoden der Bildverarbeitung und modernen Machine Learning Techniken.

Es wird geprüft, ob durch diese Kombination nun weniger oder andere Probleme und Grenzen auftreten.

¹ (PwC - PricewaterhouseCoopers IL, 2017, S. 11)

² (PwC - PricewaterhouseCoopers IL, 2017, S. 9)

³ (Ford Motor Company, 2017)

⁴ (Audi AG, 17), (Auto Zeitung, 2017)

⁵ (Mortsiefer, 2016), (Wilkens, 2018), (dpa, Reuters, 2018)

⁶ (Bernau, 2017)

⁷ (de la Escalera, Armingol, & Mata, 2002, S. 1)

2 Vorbetrachtung

Um den Umfang der praktischen Implementierung an die Vorgaben der Hochschule anzupassen, wurde das weitgefasst vorgegebene Thema „Verkehrszeichenerkennung“ genauer untersucht, um den Aufwand für die Implementierung der Anforderungen abschätzen zu können.

Bei einer ersten Marktanalyse wurden sowohl kommerzielle Produkte als auch akademische Projekte auf deren Funktionsumfang, Funktionsweise und Forschungserfolge untersucht und miteinander verglichen.

Unter Miteinbezug der jeweiligen Teamgrößen konnten die erreichten Ergebnisse ausgewertet werden, um die Anforderungen an das Projekt dieser Arbeit zu definieren.

Um die Ergebnisse unseres Projekts hinsichtlich der Probleme und Grenzen der jeweiligen Funktionsweisen vergleichen zu können wurden zudem Stärken und Schwächen der Produkte recherchiert und festgehalten.

2.1 Marktanalyse

2.1.1 Kommerzielle Systeme

Für den Vergleich der kommerziellen Systeme wurden Testberichte des ADACs und unabhängiger Zeitschriften ausgewertet.

| Jahr | Produkt | Funktionalität | Funktionsweise | Probleme | Erkennungsrate |
|------|---|--|--|---|----------------|
| 2010 | BMW 740d ⁸ | Tempolimit anzeigen und an Frontscheibe darstellen | unbekannt, vermutlich Computer Vision | Hohe Fehlerrate, variable LED-Anzeigen und Ortseingangsschilder werden nicht erkannt | Note 1,9 |
| 2010 | Mercedes S 500 CGI ⁸ | | | | Note 2,0 |
| 2010 | VW Phaeton ⁸ | | | | 2,3 |
| 2010 | Audi A8 ⁸ | | | | 2,9 |
| 2010 | Opel Insignia ⁸ | | | | 2,8 |
| 2015 | Bosch MyDriveAssist ⁹ (Smartphone-App) | erkennt Überholverbote, Zusatzschilder und Tempolimits | Verarbeitung des Kamerabilds in der Cloud | Cloud-Upload erhöht Latenzzeit, erkennt auch Verkehrszeichen in Parallelspuren und auf LKWs | hoch |
| 2018 | BMW 7er ¹⁰ (Speed Limit Assistant) | Warnt vor hohen Geschwindigkeiten an Vorfahrtsstraßen | unbekannt, vermutlich Computer Vision ergänzt durch Machine Learning | Niedrige Erkennungsrate bei schlechten Witterungsbedingungen, keine Ampelerkennung | 83% |
| 2018 | Ford Galaxy ¹⁰ (Geschwindigkeitsregelanlage mit Geschwindigkeitsbegrenzer) | hohe Erkennungsrate, erkennt Zusatzbedingungen (Uhrzeit, Nässe) | | | 93% |
| 2018 | Mercedes S-Klasse (Aktiver Abstands-Assistent) ¹⁰ | erkennt Zusatzbedingungen (Uhrzeit, Nässe), Anpassung der empfohlenen Geschwindigkeit an den Straßenverlauf (Kurven, Kreuzungen) | | | 91% |
| 2018 | Audi A4 ¹⁰ (Prädiktiver Effizienzassistent) | | | | 90% |
| 2018 | VW Arteon ¹⁰ (Vorausschauende Geschwindigkeitsregelung) | | | | 88% |

Tabelle 1: Marktanalyse – Kommerzielle Systeme

⁸ (Buric, 2010), (autohaus24, 2013)

⁹ (Gallinge, 2015)

¹⁰ (ADAC e.V., 2018)

2.1.2 Akademische Projekte

Im Folgenden befindet sich eine Auswahl der nach unserer Einschätzung relevanten Forschungen. Einzelne Fachbegriffe der Tabelle werden erst im folgenden Grundlagenkapitel erklärt.

| Jahr | Titel | Funktionalität | Funktionsweise | Team- größe |
|------|---|---|--|----------------|
| 1991 | The robust recognition of traffic signs from a moving car ¹¹ | Erkennt verschiedene die Vorfahrt regelnde Schilder | Computer Vision mit räumlicher Merkmals-Klassifikation | 4 |
| 2002 | Traffic sign recognition and analysis for intelligent vehicles ¹² | Erkennt auch teils verdeckte Verkehrsschilder und deren Zustand | Computer Vision (Analyse von Form und Farbe) und neuronale Netze mit Farbton-Histogrammen als Merkmal | 3 |
| 2005 | A system for traffic sign detection, tracking, and recognition using color, shape, and motion information ¹³ | Erkennt runde Schilder an Tag und Nacht bei sonnigem oder bewölktem Wetter | Computer Vision (AdaBoost und Haar Wavelets) und Bayes-Klassifikator 100ms/Bild mit ~85% Erkennungsrate | 5 |
| 2011 | Traffic sign recognition with multi-scale Convolutional Networks ¹⁴ | Unterscheidet die 43 Zeichen des GTSRB Datensatzes | Machine Learning mittels CNN, Erkennungsrate ~98.97% | 2 |
| 2015 | Vision-Based Road Sign Detection ¹⁵ | Erkennt Freitext-Vorwegweiser-Schilder | Stereo Computer Vision 36ms/Bild mit ~90% Erkennungsrate | 5 |
| 2017 | SSD in TensorFlow: Traffic Sign Detection and Classification ¹⁶ | Erkennt Stopp- und Fußgänger-überweg-Schilder | Nutzt ein neuronales Netz, das mehrere Objekte je Bild klassifizieren kann | 1 |
| 2017 | Traffic signs classification with a convolutional network ¹⁷ | Unterscheidet die 43 Zeichen des GTSRB Datensatzes | Machine Learning mittels CNN, Erkennungsrate ~99.33% | 1 |
| 2017 | A Real-Time Chinese Traffic Sign Detection Algorithm Based on Modified YOLOv2 ¹⁸ | Unterscheidet zwischen verschiedenen Gebots-, Gefahren- und Verbots-Zeichen | Machine Learning mittels deep CNN / YOLO v2, bis zu 17ms/Bild und Erkennungsrate 96.69% | 4 |

Tabelle 2: Marktanalyse – Akademische Projekte

¹¹ (Seitz, Lang, Gilliard, & Pandazis, 1991)

¹² (de la Escalera, Armingol, & Mata, 2002)

¹³ (Bahlmann, Zhu, Ramesh, Pellkofer, & Koehler, 2005)

¹⁴ (Sermanet & LeCun, 2011)

¹⁵ (Kehl, Enzweiler, Froehlich, Franke, & Heiden, 2015)

¹⁶ (Sung, 2017)

¹⁷ (Staravoi, 2017)

¹⁸ (Zhang, Huang, Jin, & Li, 2017)

2.1.3 Vergleich und Auswertung

Die Marktanalyse hat gezeigt, dass zum Entwurf von vergleichsweise guten Algorithmen zur Erkennung von Verkehrszeichen in erster Linie ein hohes Fachwissen im Bereich der neuronalen Netze erforderlich ist. Während in den ersten Jahren vor allem auf klassische Algorithmen der digitalen Bildverarbeitung gesetzt wurde, verwenden viele moderne Algorithmen neuronale Netze.

Die Forschungsprojekte erzielten meist entweder hohe Erkennungsraten oder kurze Erkennungszeiten, während die kommerziellen Anbieter meist eine mittlere Erkennungsrate bei mittlerer Erkennungszeit erzielten. Wichtig zu beachten ist allerdings, dass die verschiedenen Erkennungsraten nur bedingt miteinander vergleichbar sind, da zum Test der Algorithmen verschiedene Datensätze verwendet wurden.

Die meisten Probleme traten bei der Erkennung von Schildern im echten Straßenverkehr auf, die teilweise verdeckt sind, aufgrund der Geschwindigkeiten verschwommen sind, oder zu weit entfernt sind. Auch die Erkennung in schwach beleuchteten Bildern war eine Herausforderung.

2.2 Anforderungsdefinition

Durch die Marktanalyse wurde gezeigt, dass hinsichtlich der Bearbeitungszeit und Erkennungsrate keine zu hohen Anforderungen gestellt werden sollten, wenn die Arbeit in der vorgegebenen Zeit abgeschlossen werden soll. Die Anzahl der bisherigen Forschungsprojekte und die Ergebnisse der unter hohen Investitionen entwickelten kommerziellen Produkte zeigen, dass in diese Hinsicht vermutlich keine schnellen Verbesserungen erreicht werden können.

Stattdessen wurde der Fokus dieser Arbeit auf die Untersuchung der Unterschiede zwischen den verschiedenen Funktionsweisen sowie die praktische Implementierung eines funktionsfähigen Programms zur Verkehrszeichenerkennung, das auf einer Kombination der zwei häufigsten Funktionsweisen basiert, also Computer Vision und Machine Learning kombiniert.

Gefordert ist eine Software, welche aus bei Tageslicht und guten Wetterverhältnissen aufgenommenen Videos einen Großteil der vollständig sichtbaren Verkehrszeichen einer Liste erkennt und korrekt klassifiziert. Eine Ausgabe soll dem Nutzer die Ergebnisse darstellen.

Primär sollen die Geschwindigkeits-, Gefahren- und Vorfahrtszeichen erkannt werden. Weitere Schildtypen, wie beispielsweise die Richtungszeichen, sind als optional vorgesehen.

Erreicht die Software sehr gute Ergebnisse, so ist eine Optimierung für nicht optimale Einsatzbedingungen möglich.

Eine zusätzliche Anforderung des Betreuers war, dass auch nicht aus der Fahrerperspektive aufgenommene Videos korrekt verarbeitet werden.

Hinsichtlich der Systemsicherheit und Verfügbarkeit wurden keine besonderen Forderungen gestellt. Bei einem Ausfall steht die Software nicht zur Verfügung. Die Software soll in bestehende Systeme integrierbar sein (mit verschiedenen Betriebssystemen kompatibel sein), Interaktiv bedienbar sein und eine hohe Usability bieten. Die Software soll sowohl für kurze und lange Videos geeignet sein (Skalierbarkeit) und einfach zu erweitern sein.

3 Theoretische Grundlagen

3.1 Standardisierung von Verkehrszeichen

Die ersten Verkehrsschilder wurden in Deutschland 1877 aufgestellt. Sie dienten als Warnung vor Bahnübergängen. Circa 20 Jahre später stellten Auto- und Fahrradvereine Warntafeln auf, die zuerst nur Text enthielten und später noch um Zeichen ergänzt wurden. 1908 wurden sechs Warntafeln amtlich genehmigt, die 2 Jahre zuvor beschlossen und entworfen wurden. Diese hatten weiße Symbole auf schwarzen Grund. Um international einheitlich zu sein, wurde ein Jahr später in Paris eine schriftliche Regelung festgehalten.

Da die Gusseisenschilder im ersten Weltkrieg eingeschmolzen wurden, durften vorerst auch Automobilclubs Schilder mit Werbung aufstellen. Erst 1927 ersetzte man die Warnzeichen durch neue Schilder. Schlussendlich wurde 1968 in Wien das Übereinkommen über Verkehrszeichen abgeschlossen. Dies sollte international zu weniger Verwirrung bei der Beschilderung führen.¹⁹

Heute sind Verkehrszeichen international genormt. Nachfolgend wird auf die relevantesten Normen eingegangen, welche für eine Erkennung von Verkehrszeichen notwendig sind.

3.1.1 Farbnorm

Das Übereinkommen über Verkehrszeichen gibt für jedes Schild die zu verwendenden Farben an. Diese werden jedoch nicht mit einem genauen Farbwert spezifiziert, sondern als rot, blau, gelb, orange, weiß oder schwarz angegeben.²⁰ Dies lässt jedem Staat die Freiheit, die Farbe seiner Verkehrszeichen eigenständig zu wählen, solange sie mit der groben Vorlage des Übereinkommens übereinstimmt. Weiter lässt das Übereinkommen oft Wahlmöglichkeiten bei der Farbe des Hintergrunds zu. So ist es erlaubt, ein Gefahrenwarnzeichen mit gelben oder mit weißem Grund herzustellen.






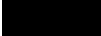
| Farbe | | RGB | HEX | RAL-F14 |
|---------|---|---------------|---------|------------------------|
| Gelb |  | 232, 191, 40 | #e8bf28 | RAL 1003 Signalgelb |
| Rot |  | 161, 38, 45 | #a1262d | RAL 3001 Signalrot |
| Blau |  | 0, 72, 115 | #004873 | RAL 5005 Signalblau |
| Grün |  | 0, 142, 94 | #008e5e | RAL 6032 Signalgrün |
| Weiß |  | 255, 255, 255 | #ffffff | RAL 9003 Signalweiß |
| Schwarz |  | 14, 19, 19 | #0e1313 | RAL 9004 Signalschwarz |

Tabelle 3: Farbwerte von Verkehrszeichen

Der deutsche Staat hat die Farbwerte der Verkehrszeichen genau festgelegt, damit verschiedene Produzenten dasselbe Schild einheitlich fertigen könne. Festgelegt sind diese in der ISO 7010 und ISO 3864-4. Orientiert wurde sich an den Farben des RAL-F14 Farbregisters.²¹

¹⁹ (Wikipedia-Autoren, Verkehrszeichen (Deutschland), 2018)

²⁰ (Die Vertragsparteien, 1968, S. Anhang 1, Muster 1)

²¹ (DIN EN ISO 7010, 2018)

3.1.2 Verkehrszeichentypen

Die verschiedenen Typen von Verkehrszeichen sind in Deutschland in der allgemeinen Verwaltungsvorschrift zur Straßenverkehrs-Ordnung (StVO) ²² beschrieben. Allgemein können die Verkehrszeichen der StVO in vier Klassen eingeteilt werden, die sich sowohl optisch als auch in ihrer Funktion unterscheiden.

In den nachfolgenden Abschnitten werden nur die am häufigsten verwendeten Verkehrszeichen der einzelnen Klassen aufgezählt, da eine ausführliche Beschreibung den Rahmen der Arbeit übersteigen würde.

Eine Untersuchung der Form und Funktion der einzelnen Schilder im Rahmen dieser Arbeit ist wichtig, damit bestimmt werden kann, welche Verkehrszeichen in der praktischen Implementierung primär erkannt werden sollten.

Die untenstehenden Beschreibungen fassen die Definitionen des Übereinkommens von 1968 sowie die Ergänzungen in der StVO zusammen.²³

3.1.2.1 Gefahrenzeichen

Die Gefahrenzeichen der StVO dienen zur frühen Warnung vor Gefahren, die auch von vorsichtigen Verkehrsteilnehmern nur schwer erkennbar sind.



Abb. 1: Gefahrenzeichen

(Korsch, 2017)

-
- a. Gefahrenzeichen des Typs A^a bestehen aus einem gleichseitigen Dreieck. Eine Seite ist waagrecht und die Spitze zeigt nach oben. Der Grund ist in Deutschland weiß, kann international jedoch auch gelb sein. Der Rand des Zeichens ist rot und in der Mitte ist optional ein schwarzes Gefahrensymbol.
 - b. Gefahrenzeichen des Typs A^b bestehen aus einem auf seiner Spitze stehendem Quadrat. Der Hintergrund ist weiß oder gelb und der schmale Rand schwarz. Die Symbole des Zeichens sind schwarz oder dunkelblau. Jeder Staat muss sich entscheiden, ob er Zeichen des Typs A^a oder A^b verwendet und diese daraufhin einheitlich durchsetzen
 - c. Eine Ausnahme bei den Gefahrenzeichen bildet das Andreaskreuz an Bahnübergängen. Es besteht aus rot-weiß gestreiften sich kreuzenden Rechtecken.

²² (Bundesregierung, 2017)

²³ (Die Vertragsparteien, 1968; Bundesregierung, 2017)

3.1.2.2 Vorfahrtszeichen

Vorfahrtszeichen dienen zur Regelung des Verkehrs an Kreuzungen, Einmündungen oder Straßenverengungen. Sie sollen Unklarheiten bei der Vorfahrt der Verkehrsteilnehmer reduzieren und zu einem flüssigen Verkehrsfluss beitragen.²⁴

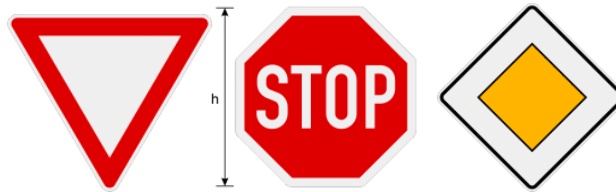


Abb. 2: Vorfahrtszeichen

(Korsch, 2017)

- a. Das Zeichen „Vorfahrt gewähren“ besteht aus einem gleichseitigen Dreieck. Eine Seite ist waagrecht und die Spitze zeigt nach unten. Das Design, die Farbe und die Größe orientiert sich an Gefahrenzeichen A^a. Es besitzt jedoch kein Symbol.
- b. Das Zeichen „Halt“ besitzt zwei Muster. In Deutschland wird fast ausschließlich Muster B2^a verwendet. Es besteht aus einem Roten Achteck mit einem weißem „STOP“ Schriftzug.
- c. Das weniger übliche Muster B2^b besteht aus einem weißen Kreis mit Roten Rand. In diesem ist ein rotes Dreieck und der Schriftzug „STOP“.
- d. Das Zeichen „Vorfahrtstraße“ besteht aus einem Quadrat, dessen Diagonale senkrecht steht. Es ist weiß mit einem schmalen schwarzen Rand. In der Mitte des Zeichens befindet sich ein gelb oder oranges Quadrat mit einem ebenfalls schwarzen Rand.
- e. Die Zeichen „dem Gegenverkehr Vorrang gewähren“ und „Vorrang vor dem Gegenverkehr“ (nicht abgebildet) bestehen aus weißem oder gelbem Grund und einem roten Rand. Der Vorrang bezeichnende Pfeil ist schwarz und der andere rot.

²⁴ (Die Vertragsparteien, 1968; Bundesregierung, 2017)

3.1.2.3 Verbots und Beschränkungszeichen

Verbotszeichen dienen zum Ausschluss von Verkehrsteilnehmern aus gewissen Bereichen. Beschränkungszeichen werden verwendet, um die erlaubte Höchstgeschwindigkeit anzugeben.²⁵



Abb. 3: Verbots und Beschränkungszeichen

(Korsch, 2017)

Die Verbots- und Beschränkungszeichen sind rund und besitzen einen roten Rand und einen weißen oder gelben Grund.

- Das Zeichen „Einfahrt verboten“ hat einem blanken Hintergrund und kann mit schwarzen Symbolen oder um eine Schrift erweitert werden, um bestimmte Fahrzeuggruppen zu verbieten.
- Das Zeichen für eine Geschwindigkeitsbeschränkung besteht aus dem blanken Hintergrund mit einer schwarzen Zahl, welche die zulässige Höchstgeschwindigkeit angibt.
- Das Zeichen, das Verbote und Beschränkungen aufhebt, besteht aus einem weißen oder gelben Kreis mit keinem oder schmalem schwarzem Rand. Von rechts oben nach links unten verläuft ein schwarzes oder dunkelgraues Band.

3.1.2.4 Gebotszeichen

Gebotszeichen weisen den Verkehrsteilnehmer auf Sachverhalte hin, denen er nachkommen muss. Dies kann beispielsweise die zu verwendende Fahrspur oder eine geforderte Mindestgeschwindigkeit sein.²⁵



Abb. 4: Gebotszeichen

(Korsch, 2017)

- Gebotszeichen bestehen aus einem blauen Kreis mit weißen Symbolen.
- Das Zeichen „Vorgeschriebene Seite des Vorbeifahrens“ besteht aus einem blauen Kreis mit weißem Pfeil.
- Das Zeichen „Mindestgeschwindigkeit“ beinhaltet eine weiße Geschwindigkeitsangabe.

²⁵ (Die Vertragsparteien, 1968; Bundesregierung, 2017)

3.1.3 Größe

Bei hohen Geschwindigkeiten müssen Schilder schon aus weiter Entfernung deutlich erkennbar sein. Die vorgeschriebene Größe von Verkehrszeichen ist daher abhängig von der für den jeweiligen Streckenabschnitt zugelassenen Höchstgeschwindigkeit. In der Allgemeinen Verwaltungsvorschrift zur Straßenverkehrs-Ordnung werden deshalb die in den folgenden Tabellen dargestellten Größen und Geschwindigkeiten definiert.²⁶

| Verkehrszeichen | Größe 1 | Größe 2 | Größe 3 |
|---------------------------------|-----------------|-----------------|------------------|
| Rund (Durchmesser) | 420 mm | 600 mm | 750 mm |
| Dreieck (Seitenlänge) | 630 mm | 900 mm | 1260 mm |
| Quadrat (Seitenlänge) | 420 mm | 600 mm | 840 mm |
| Rechteck (Höhe x Breite) | 630 mm x 420 mm | 900 mm x 600 mm | 1280 mm x 840 mm |

Tabelle 4: Größen von Verkehrszeichen

nach (Bundesregierung, 2017)

Die oben beschriebenen Größen sollen laut Vorschrift für die folgenden zugelassenen Höchstgeschwindigkeiten verwendet werden:

| Verkehrszeichen-Form | Geschwindigkeitsbereich für den Einsatz von ... | | |
|-----------------------------------|---|---------------|------------|
| | Größe 1 | Größe 2 | Größe 3 |
| Rund | 0 – 20 km/h | 20 – 80 km/h | > 80 km/h |
| Dreieck, Quadrat, Rechteck | 20 – 50 km/h | 50 – 100 km/h | > 100 km/h |

Tabelle 5: Zuordnung der Größen zur Höchstgeschwindigkeit

nach (Bundesregierung, 2017)

²⁶ (Bundesregierung, 2017)

3.1.4 Sichtbarkeit und Regeln zur Aufstellung

Dieser Abschnitt befasst sich mit den wichtigsten Regeln zur Sichtbarkeit und Aufstellung von Verkehrszeichen. Die für die Arbeit relevanten Regeln wurden herausgearbeitet und zusammengefasst. Alle Regeln beziehen sich auf die StVO-Paragraphen §39 bis §43.²⁷

Verkehrszeichen müssen gut sichtbar im rechten Winkel rechts von der Fahrbahn angebracht werden. Sie dürfen nur links oder nur über der Fahrbahn angebracht werden, wenn dies Missverständnisse vermeidet und sie auch bei Dunkelheit deutlich sichtbar sind. Wo es nötig ist, können sie jedoch auch auf beiden Fahrbahnseiten aufgestellt werden.

Ist es notwendig, Verkehrszeichen auch aus einem breiten vertikalen Winkel zu erkennen, so können diese gewölbt werden. Dies darf jedoch nicht die allgemeine Erkennbarkeit des Zeichens beeinflussen.

Eine Häufung von Verkehrszeichen ist zu vermeiden. Sind dennoch an einer Stelle mehrere Verkehrszeichen vorhanden, so sind die für den Verkehr wichtigen hervorzuheben. Dies kann durch Übergröße oder ein Warnlicht geschehen.

An gleichem Pfosten oder in unmittelbarer Nähe dürfen nicht mehr als drei Verkehrszeichen für den fließenden Verkehr angebracht werden. Gefahrenzeichen haben grundsätzlich allein zu stehen.

An spitzen Einmündungen ist es notwendig, die Verkehrszeichen eindeutig einer Fahrbahn zuzuschreiben. Dies ist gegebenenfalls mit Sichtblenden zu realisieren.

Auch der Abstand von der Straße zum Verkehrszeichen ist geregelt. Innerorts beträgt dieser im Normalfall 0,5 m und außer Orts 1,5 m. Die Unterkante des Verkehrszeichens darf im Regelfall nicht unter 2 m über dem Straßenniveau liegen. Bei Fahrradwegen ist die Grenze 2,2 m und bei Schilderbrücken 4,5 m. Auf Verkehrsinseln und Teilern sollten die Schilder eine Höhe von 0,6 m aufweisen.

²⁷ (Bundesregierung, 2017)

3.2 Digitale Bildverarbeitung

Bei der Verkehrszeichenerkennung geht es darum, Verkehrszeichen in einem Bild oder in Bild-Sequenzen (Videos) zu erkennen. Zur Entwicklung von Algorithmen, die solche Bilder maschinell verarbeiten, ist ein grundlegendes Verständnis über die verschiedenen Kategorien und Begriffe der digitalen Bildverarbeitung erforderlich.

3.2.1 Bildverarbeitung

Die reine Bildverarbeitung beinhaltet die **Bildverbesserung** und **Bildvorverarbeitung**. Sie nutzt Operationen, welche die im Bild vorhandenen Informationen aufwerten. Hierzu gehört beispielsweise die Transformation der Grauwertlinie, welche den Dynamikbereich eines Bildes auf den des kompletten Farbraums anpasst. Eine andere, oft eingesetzte, Methode ist die Binarisierung, welche Objekte aus einem Hintergrund extrahiert.

3.2.2 Bildanalyse

Die Bildanalyse dient der reinen **Informationsextraktion** aus Bildern, also die Technik hinter dem oft verwendeten Begriff „**Computer Vision**“. Auch hier existieren große Unterschiede zwischen den vorhandenen Verfahren.

3.2.2.1 Computer Vision

Computer Vision (CV) beschreibt die maschinelle Lösung von Aufgaben, die Menschen durch „sehen“ lösen würden. Computer-Vision-Algorithmen werden schon seit einigen Jahren eingesetzt, beispielsweise an Supermarktkassen, um Barcodes zu scannen, in Produktionsbetrieben, um Qualitätskontrollen durchzuführen und in Multifunktionsdruckern, um Text in eingescannten Bildern zu erkennen.²⁸

Auch die Wahrnehmung von Verkehrszeichen geschieht über den menschlichen Sehsinn, sie werden während der Fahrt visuell erkannt, und nicht etwa erfühlt, erhört oder anhand ihres Geruchs ermittelt. Es ist daher sehr naheliegend, auch die Verkehrszeichenerkennung in dieser Arbeit mithilfe von Computer-Vision-Algorithmen umzusetzen.

Zur Programmierung solcher Algorithmen stehen bereits zahlreiche Frameworks zur Verfügung. Zu den bekanntesten zählen unter anderem DLib und OpenCV. DLib ist eine Sammlung verschiedener, für C++ geschriebener Algorithmen, die die Erstellung umfangreicher Software vereinfachen sollen.²⁹ OpenCV ist eine Open Source Computer Vision und Machine Learning Bibliothek mit vielen hochoptimierten Algorithmen und Unterstützung von zahlreichen Programmiersprachen.³⁰ Die Python-Version von OpenCV wird im Rahmen dieser Arbeit eingesetzt, um Bilder des Videos einzulesen und in diesen Formen, Konturen und Muster zu erkennen, sowie um Objekte im Bild zu verfolgen. Details zur Umsetzung werden in Kapitel 5 beschrieben.

Die durch die Frameworks bereitgestellten Algorithmen basieren auf verschiedenen Techniken. Bei der **Farbklassifikation** werden Farben in einem Bild bestimmt oder gefiltert. Das erlaubt es beispielsweise, den Bildausschnitt mit dem größten Orange-Anteil zu bestimmen und über mehrere Einzelbilder zu verfolgen. Eine **Kontrastanalyse**, beispielsweise durch eine Hough-Transformation umgesetzt, dient der Erkennung von geometrischen Objekten im Bild.

²⁸ (Melde, 2018), (Sablatnig, Zambanini, & Licandro, 2014)

²⁹ (King, 2018)

³⁰ (OpenCV team, 2018)

Eine Analyse des optischen Flusses ermöglicht eine **Bewegungsextraktion**. Diese dient der Unterscheidung von bewegten Objekten im Vordergrund und einem Hintergrundbild. Auch eine **Kantenerkennung** kann mithilfe verschiedener mathematischer Modelle geometrische Objekte erkennen. Zum Einsatz kommen hierfür der Sobel-Operator, Wavelets, die Gauß-Laplace-Pyramide, Gabor-Wavelets und Laplacian-Of-Gaussian-Filter.³¹

3.2.3 Bilderkennung

Die Bild-**Detektion** beschränkt sich auf das reine Erkennen von Objekten oder ganzen Klassen. Hierfür wird das Bild systematisch nach Merkmalen abgesucht. Eine Detektion, die beispielweise Fahrzeuge im Querschnitt erkennt, könnte das Bild nach zwei beieinanderliegenden Reifen (Kreisen) untersuchen. Werden diese gefunden, war der Detektor erfolgreich und gibt die Position aus dem Bild zurück.³²

3.2.4 Bildklassifikation

Eine Klassifikation dient der **Einteilung** von Objekten in **verschiedene Klassen**. Ein Klassifikator kann beispielweise keine Autos erkennen, jedoch Bilder von Fahrzeugen in verschiedene Klassen einteilen. Der Detektor unterscheidet bei Fahrzeugen nicht zwischen Bus, Auto oder Lastwagen. Der Klassifikator untersucht das Bild jedoch nach spezifischen Merkmalen der einzelnen Klassen.

Sind die Merkmale einer Klasse besonders stark ausgeprägt, so befindet sich mit hoher Wahrscheinlichkeit ein Objekt dieser Klasse im Bild.

Für Klassifikatoren wird oft Machine Learning eingesetzt, das in Abschnitt 3.3.1 näher beschrieben wird.

Die **Detektion durch Klassifikation** ist ein rechenaufwändiges Verfahren, welches das Bild mehrfach in verschieden große Abschnitte unterteilt und diese versucht zu klassifizieren. Überschreitet der Klassifikator in einem Abschnitt einen definierten Schwellwert, gilt das dort dargestellte Objekt als detektiert und klassifiziert. Die verschiedenen Fenstergrößen, mit denen das Bild für jede Klasse untersucht werden muss, lässt die benötigte Rechenleistung stark ansteigen.

3.2.5 Tracking

Das Tracking beschäftigt sich mit der **Verfolgung von Objekten** in Videos. Dies soll in erster Linie eine ständige Detektion und aufwändige Klassifikation vermeiden. Hierfür wird ein Verfahren eingesetzt welches ein Muster aus dem zu verfolgenden Bereich generiert. Der Algorithmus geht davon aus, dass sich das Objekt von Einzelbild zu Einzelbild nur minimal bewegt. Mit diesem Wissen wird im darauffolgenden Einzelbild das nahe Umfeld des Objektes untersucht. Die Region des Umfelds, das die höchste Übereinstimmung mit dem Muster liefert, ist die neue Position des Objekts.³³

³¹ (Bruder, 2018)

³² (Gehrig, 2018)

³³ (Bruder, 2018)

3.3 Künstliche Intelligenz

Deutsche Straßenteilnehmer erlangen ihr Wissen über Verkehrszeichen meist zunächst theoretisch, beispielsweise in einer Fahrschule, in der sie unter anderem lernen, welche Typen von Verkehrsschildern es gibt, und was diese bedeuten.

Anschließend wird das Wissen anhand von praktischen Erfahrungen im Straßenverkehr vertieft, bei denen intelligenten Entscheidungen anhand des gelernten Wissens für eine sichere Fahrt unerlässlich sind.

Unter dem Begriff Künstliche Intelligenz (KI) versteht man ein Forschungsgebiet, in dem versucht wird, Mechanismen zu entwerfen, mit denen Maschinen oder Computer intelligentes Verhalten entwickeln können.³⁴

Künstliche Intelligenz scheint daher auch für die Erkennung von Verkehrszeichen relevant zu sein.

Die zentrale und auch für diese Arbeit relevante Methode der KI ist Machine Learning.

³⁴ (Wolski, 2016)

3.3.1 Machine Learning

Machine Learning beschreibt eine Technik, bei der Computeralgorithmen aus vorausgegangenen Situationen lernen und Rückschlüsse für neue Situationen schließen können.

Anwendung findet diese Technik in immer mehr Bereichen und wird beispielsweise zur Prognose von Aktienkursen oder zur Vorhersage von Produktionsausfällen eingesetzt.³⁵

Machine Learning besteht aus einer Lern- und Anwendungsphase. In dem Lern- oder auch Trainingsphase genannten Schritt lernt das System aus Daten („Erfahrung“), die es bei der späteren Anwendung zur Nachbildung von kognitiven Fähigkeiten des Menschen einsetzt.³⁶

Der Lernalgorithmus bestimmt die Parameter, mit dem das Modell des neuronalen Netzes aufgebaut wird. Bei der Anwendung werden die durch eine Merkmalsextraktion bestimmten Eigenschaften des zu klassifizierenden Bildes mit den Modellparametern zusammen an einen Klassifikator gegeben.

Konkret ermöglicht uns Machine Learning die Klassifizierung von Bildern nur anhand der dargestellten Objekte, ohne dass weitere Metadaten im Bild enthalten sein müssen.

Allgemein wird zwischen drei Lernmethoden unterschieden. Beim überwachten Lernen (Englisch: supervised) nutzt das Training beschriftete Beispielbilder. Beim unüberwachten Lernen (Englisch: unsupervised) wird eine unsortierte Menge von Daten genutzt, die selbstständig in verschiedene Klassen unterteilt wird. Eine letzte Methode ist das sogenannte bestärkende Lernen (Englisch: Reinforcement Learning), bei dem ein System aus Belohnungen und Bestrafungen aufgebaut wird, das den Trainingsalgorithmus bei guten Entscheidungen belohnt und bei schlechten Entscheidungen bestraft. Der stetige Versuch die Belohnung zu maximieren führt zu immer besseren Ergebnissen.³⁶

Zur Veranschaulichung des Reinforcement Learnings ein kleines Beispiel: Angenommen ein autonomes Fahrzeug fährt 1000-mal mit zufällig gewählten Parametern über eine Kreuzung. Eine Kollision mit anderen Fahrzeugen führt zu einer Bestrafung, eine makellose Überquerung zu einer Belohnung. Irgendwann, nach vielen Versuchen, weiß der Algorithmus, dass es sinnvoll ist, vor Ampeln langsamer zu werden und auf grüne Licht-Signale zu warten, da das die Anzahl der Kollisionen und damit die Bestrafungspunkte verringert.

Aus offensichtlichen Gründen ist dies für eine Verkehrszeichenklassifikation nicht geeignet. Anwendung findet Reinforcement Learning beispielsweise in Simulationen, die beliebig oft wiederholt werden können.

Für die Verkehrszeichenerkennung ist vor allem ein überwachtes Lernen sinnvoll, da es einen definierten Katalog an Verkehrszeichen gibt.³⁷

³⁵ (Tiedemann, 2017)

³⁶ (Noé, 2018)

³⁷ (Bundesregierung, 2017)

3.3.2 Künstliche neuronale Netze

Um Machine Learning zu realisieren, können künstliche neuronale Netze eingesetzt werden. Die Idee hierbei ist es, Strukturen des menschlichen Gehirns nachzubilden. Neuronale Netze bestehen hauptsächlich aus Neuronen und Synapsen, die in sogenannten Layern angeordnet sind. Komplexe Netze bestehen aus mehreren Millionen Neuronen, für ein einfaches Beispiel genügen bereits zwei Neuronen.³⁸



Abb. 5: Minimales neuronales Netz

Auf der linken Seite in Abb. 5 befindet sich der sogenannte Input-Layer. Das darin liegende erste Neuron nimmt die Eingangsdaten entgegen und leitet sie an alle Synapsen weiter. Jede Synapse hat ein bestimmtes Gewicht (Englisch: weight), mit dem alle Eingangsdaten multipliziert werden, bevor sie an die Output-Neuronen des Output-Layers weitergegeben werden.

Durch Verändern der Gewichte kann die Ausgabe modifiziert werden. Ein Machine Learning Algorithmus verändert während des Trainings die Gewichte der einzelnen Synapsen sehr oft und schaut sich am Ende an, welche Gewichte zu dem besten Ergebnis führen.

Ein einfaches Beispiel: Die Eingabewerte $x = \{1, 2, 5\}$ sollen auf die Ausgabewerte $f(x) = \{2, 4, 10\}$ abgebildet werden. Da unser Netz nur aus einer Synapse besteht, ist die Funktion einfach aufgebaut: $f(x) = S_1 * x$. Für die gegebenen Beispielwerte ist die Funktion genau dann wahr, wenn für das Synapsen-Gewicht $S_1 = 2$ gilt.

Der Machine Learning Algorithmus kennt diese Lösung nicht, weshalb während des Trainings die Gewichte des Netzes Schritt für Schritt in bestimmten Schrittgrößen erhöht werden, bis die Lösung erreicht wurde.

Für eine Bildklassifikation werden nun Bilder auf deren Beschriftungen abgebildet. Für jeden Pixel des Eingabebilds gibt es ein Eingabeneuron, das Werte zwischen 0 und 255 (für Graustufen) oder drei Eingabeneuronen, die die Farbwerte für Rot, Grün und Blau enthalten (bei einem RGB-Bild).

Die Anzahl der Ausgangsneuronen entspricht der Anzahl der zu unterscheidenden Klassen.

³⁸ (Rashid, 2017, S. 3-8, 37-59)

3.3.3 Convolutional Neural Network

Für Bildklassifikation effizienter ist eine Spezialform der neuronalen Netze, bei der die Neuronen der verschiedenen Layers in zweidimensionalen Matrizen angeordnet werden. Jede Synapse, in diesem Modell Kernel genannt, bedient nun mehrere Neuronen, die in der sogenannten Faltungsmatrix beieinander liegen. Bei den Berechnungen werden also nicht nur die einzelnen Neuronen, sondern auch deren Nachbar-Neuronen miteinbezogen.³⁹

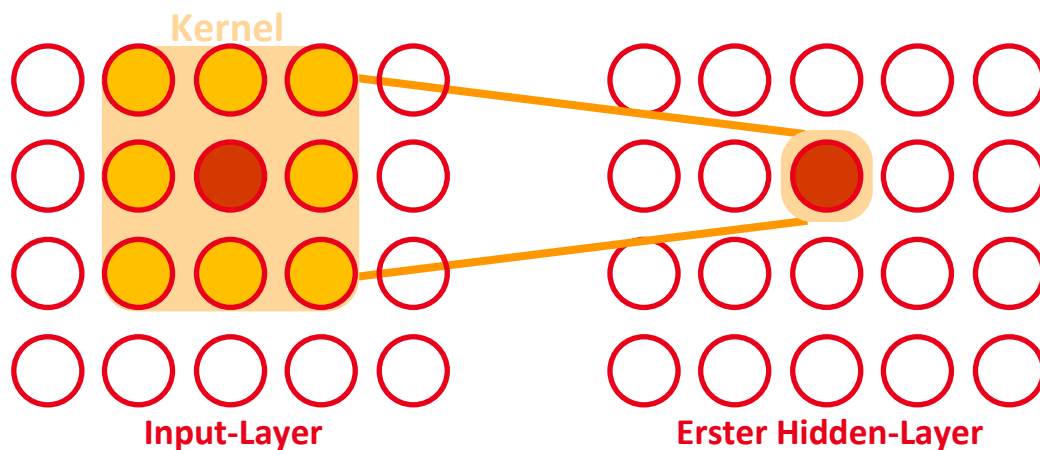


Abb. 6: Convolutional Neural Network Layer

In Abb. 6 ist neben der Neuronen-Matrix und dem Kernel auch ein erster Hidden Layer gezeigt. Mithilfe sogenannter Hidden Layers zwischen Input- und Output-Layer ist es möglich, mehrere Neuronen, in unserem Fall also Bildausschnitte, zusammenzufassen.

Der Kernel „fährt“ nun über das gesamte Bild (alle Neuronen des Input-Layers) und lässt alle, in diesem Fall 9 Nachbar-Neuronen in seine Berechnung einfließen. Das Ergebnis der Rechnung gibt er an den nächsten Layer weiter.

Die Aufgabe des Trainings ist erneut das Bestimmen von für die Berechnung geeigneten Parametern zur Gewichtung des Kernels.

³⁹ (Liang & Hu, 2015, S. 3367-3371)

Statt wie im vorherigen Abschnitt gezeigt ein Gewicht, besitzt der **Kernel** in diesem Fall nun für jede Neuronen-Position ein Gewicht, in unserem Fall also 9 Gewichte (siehe rote Matrix in Abb. 7). Am schwersten ist in diesem Beispiel das zentrale Neuron gewichtet.

Die Gewichte werden weiterhin mit den **Eingangswerten** multipliziert. Zur Bestimmung eines Wertes für den nächsten Layer wird anschließend beispielsweise das arithmetische Mittel der Werte genommen.

$$\frac{\text{sum} \left(\begin{pmatrix} 1 & 1 & 1 \\ 1 & 90 & 1 \\ 1 & 1 & 1 \end{pmatrix} * \begin{pmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{pmatrix} \right)}{9} = 94,4$$

Abb. 7: Beispielhafte Berechnung der Kernel-Ausgangswerte

Das Ergebnis dieser Berechnung wird nun also an den nächsten Layer weitergegeben, wobei alle Hidden Layers die oben genannten Schritte bis zum Erreichen des Output Layers wiederholen.

Die Gewichtung des Kernels ist abhängig vom Anwendungsfall. Für eine Gesichtserkennung werden beispielsweise die Matrix-Positionen in Augennähe besonders hoch gewichtet.

Auch die sogenannte Pooling-Funktion, die die Matrizen diskretisiert, ist vom Anwendungsfall abhängig. Die in der Praxis am häufigsten verwendete Funktion verwendet statt dem arithmetischen Mittel (Mean-Pooling) das Maximum der Matrix (Max-Pooling).

Eine weitere für die Bildklassifikation verwendete Methode ist Softmax-Pooling, bei dem die Softmax-Funktion (auch normalisierte Exponentialfunktion genannt) eingesetzt wird.⁴⁰

⁴⁰ Die Definition dieser Funktion kann in (Bishop, 2006, S. 198) nachgeschlagen werden.

4 Projektplanung

Nachfolgend wird die Planung der praktischen Implementierung beschrieben. Diese dient zur Einteilung von Arbeitspaketen und zur Zeitplanung. Sie beschreibt die vorläufigen Ideen, von denen die spätere Implementierung in Teilen abweichen kann.

Aufgrund des großen Projektumfangs wurde bereits zu Beginn festgelegt, dass ein modularer Aufbau benötigt wird. Dies sollte die Implementierung vereinfachen und Aufgabengebiete besser voneinander trennen. Das Programm zur Verkehrszeichenerkennung lässt sich in die folgenden Module aufteilen:

- 4.1 Benutzeroberfläche
- 4.2 Erkennung und Klassifikation
 - 4.2.1 Formerkennung
 - 4.2.2 Bildklassifikation
- 4.3 Ergebnisdarstellung

Zur Verknüpfung dieser Module gibt es noch ein Rahmenprogramm, das Ein- und Ausgaben koordiniert, bedingte Modulaufrufe regelt und als ausführbares Programm mit einem Klick gestartet werden kann.

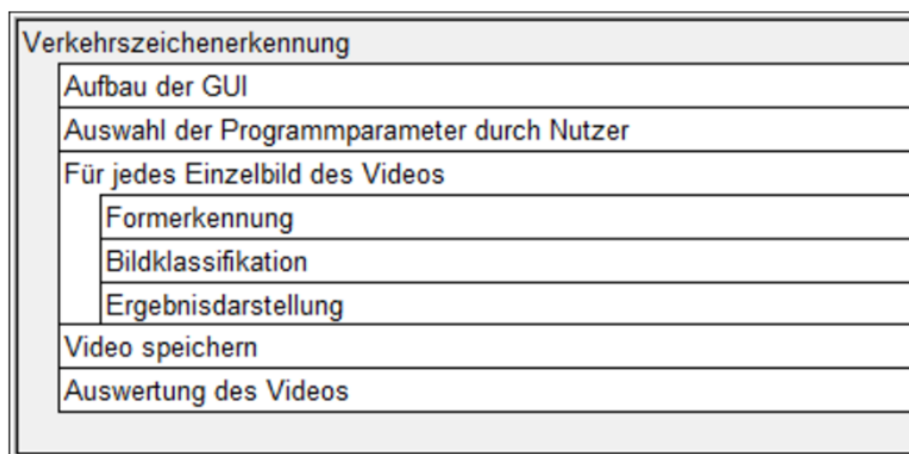


Abb. 8: Struktogramm zur praktischen Implementierung

Die Anordnung der einzelnen Module ist in der oberen Abbildung dargestellt.

Zur Programmierung soll die Skriptsprache Python in Version 3.6.3. eingesetzt werden, da die für die einzelnen Module benötigten Bibliotheken und Frameworks alle eine Python-Implementierung besitzen, Python plattformunabhängig ist und eine große Entwickler-Community besitzt.

4.1 Benutzeroberfläche

Die Schnittstelle zur Bedienung der Software ist ein grafisches Windows- und Linux- Programmfenster (auch: Graphical User Interface, GUI), das den Eintrittspunkt des Programms darstellt und die Auswahl der Ein- und Ausgabedateien sowie die Konfiguration der Verkehrszeichenerkennung erlaubt.

Es wurde festgelegt, dass für die Implementierung der Benutzeroberfläche das Framework appJar verwendet werden soll, ein Wrapper für das bekannte GUI-Toolkit Tk.

Link 1: appJar GUI-Framework <http://appjar.info/>

Neben diesem Start- und Einstellungsmenü gibt es noch ein zweites Modul mit einer grafischen Ausgabe, das allerdings nicht interaktiv bedient werden kann. Diese Ergebnisdarstellung ist in Abschnitt 4.3 beschrieben.

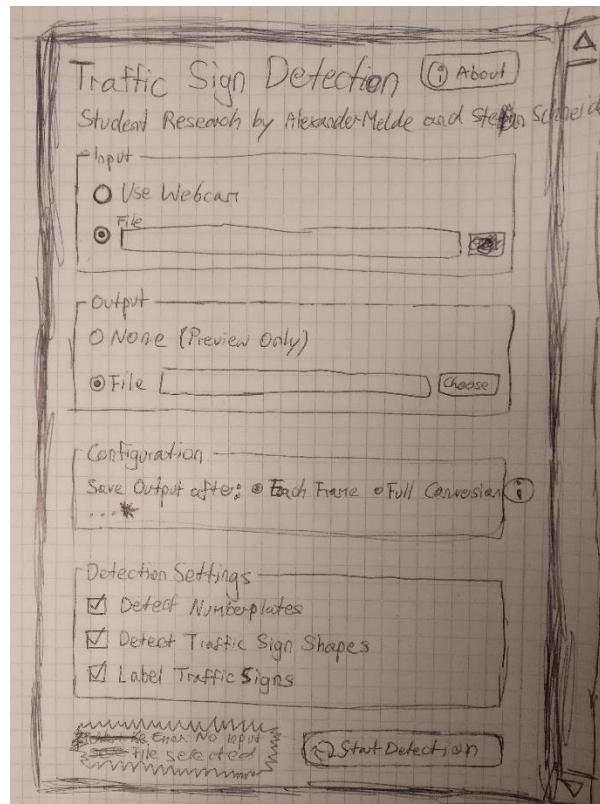


Abb. 9: Erstes GUI Mockup

Während der Planung wurde das Startmenü wie in Abb. 9 gezeigt bereits als Mockup entworfen.

4.2 Erkennung und Klassifikation

Zurückblickend können die bisherigen Ansätze zur Erkennung von Verkehrszeichen in zwei Kategorien eingeteilt werden.

Bereits Anfang der 90er Jahre wurde an Algorithmen geforscht, die Verkehrszeichen anhand ihrer Form, Farben und Konturen erkennen. Derartige Algorithmen stoßen vor allem bei unterschiedlichen Belichtungen an ihre Grenzen, weshalb oft aufwändige Farbraum-Konvertierungen von beispielsweise RGB (Red, Green, Blue) nach HSL (Hue, Saturation, Lightness) erforderlich sind. Bei HSV wird die Helligkeit durch den Parameter V bestimmt, was die Belichtungsschwankungen normalisiert.⁴¹

Modernere Forschungen nutzen KI-Ansätze, bei denen mithilfe von Machine Learning die zur Unterscheidung benötigten Features vom Algorithmus selbst bestimmt beziehungsweise gelernt werden.⁴²

Der in dieser Arbeit verwendete Algorithmus soll beide Techniken kombinieren, um in einem ersten Schritt alle Verkehrszeichen anhand ihrer Konturen zu bestimmen (Formerkennung) und anschließend mithilfe von Machine Learning die genaue Bedeutung des Verkehrszeichens (Bildklassifikation) zu ermitteln.

Die Erkennung von Verkehrszeichen in einem Video soll Bildweise geschehen und von vorherigen Ergebnissen zunächst unabhängig sein. Ein Einbezug vorheriger Einzelbilder, beispielsweise mithilfe einer Objektverfolgung, kann optional als Zusatzfunktion implementiert werden.

⁴¹ (Seitz, Lang, Gilliardt, & Pandazis, 1991)

⁴² (Zhang, Huang, Jin, & Li, 2017, S. 3-4)

4.2.1 Formerkennung

Bei der Formerkennung geht es darum, die Positionen von Verkehrszeichen anhand ihrer Form in einem Bild mit einer beliebigen Anzahl von Verkehrszeichen zu erkennen.

Die Realisierung soll mit der Bildverarbeitungs-Bibliothek OpenCV geschehen. Mithilfe eines Farbfilters soll zuerst nach für Schilder typischen Farben gefiltert werden. Anschließend wird mithilfe einer Formerkennung geprüft, ob in den farblich passenden Bereichen Konturen in Form von Verkehrsschildern vorhanden sind.

Link 2: OpenCV Bibliothek <https://opencv.org/>

Das „Shape Detection“-Modul gibt für eckige Schilder die Anzahl der Ecken und die Pixel-Koordinaten, die den Rahmen der einzelnen Formen beschreiben, zurück. Für runde Schilder werden deren Mittelpunkt und deren Radius zurückgegeben. Die Rückgabe der Koordinaten ist weniger Speicherintensiv als das Rückgeben von zugeschnittenen Bildausschnitten.

4.2.2 Bildklassifikation

Das Ziel der Bildklassifikation ist es, die genauen Bezeichnungen von Verkehrszeichen zu bestimmen.

Hierfür sollen Methoden des Machine Learning eingesetzt werden. Wie in Abschnitt 3.3.1 bereits erwähnt, werden zur Klassifikation beschriftete Beispielbilder von Verkehrszeichen benötigt.

Zur Gewinnung dieser Trainingsdaten können neben der Verwendung von öffentlichen Datensätzen auch eigene Videos aufgenommen werden, in dessen Einzelbildern Verkehrszeichen zugeschnitten und manuell beschriftet werden können.

Das Training des neuronalen Netzes und die Klassifizierung der Verkehrszeichen soll mithilfe des Machine-Learning Frameworks TensorFlow erfolgen.

Link 3: TensorFlow Framework <https://tensorflow.org/>

Die Bildklassifikation soll die wahrscheinlichste Typenbezeichnung sowie deren Wahrscheinlichkeit zurückgeben.

4.3 Ergebnisdarstellung

Nachdem die Schilder im Bild gefunden und beschriftet wurden, sollen diese Ergebnisse in das Originalbild eingezeichnet werden.

Es ist geplant, um jedes erkannte Verkehrszeichen einen beschrifteten Rahmen zu zeichnen. Zeichen, die nur mit sehr geringer Wahrscheinlichkeit beschriftet wurden können optional ausgeblendet oder mit einem andersfarbigen Rahmen dargestellt werden.

Das Zeichnen des Rahmens, das Einfügen der Beschriftungen und die Anzeige des modifizierten Bilds können ebenfalls mithilfe der OpenCV-Bibliothek umgesetzt werden.

Nach der Anzeige für den Benutzer sollte jedes Einzelbild auch innerhalb des Programms im Arbeitsspeicher oder direkt in die Ausgabedatei gespeichert werden, damit das vollständige Video nach Analyse aller Einzelbilder ohne erneute Berechnungen am Stück angesehen werden kann.

4.4 Risikoanalyse

Da viele große Projekte aufgrund von schlechtem Projekt-Management und den daraus resultierenden Problemen scheitern, sollten die Projekt-Risiken im Vorfeld erarbeitet werden.⁴³ Sind die Probleme und deren geschätzte Auswirkungen bekannt, so können entsprechende Gegenmaßnahmen definiert werden.

In Abschnitt 2.1 – Marktanalyse wurde bereits erarbeitet, welche Probleme und Grenzen bisherige Projekte hatten. Im Rahmen der Risikoanalyse werden deren Gefahren-Verursacher und -Auslöser mit unserem Projektplan verglichen, um potentielle Risiken zu ermitteln.

Klassische Risiken des Projektmanagements, die keinen direkten Einfluss auf die praktische Implementierung in dieser Arbeit haben, werden an dieser Stelle bewusst nicht beschrieben.

4.4.1 Risiken bei der Formerkennung

Für die Implementierung der Formerkennung wurden die folgenden Risiken definiert:⁴⁴



Abb. 10: Probleme bei der Erkennung von Verkehrsschildern

4.4.1.1 Schlechte Lichtverhältnisse

Oft passiert es, dass Verkehrszeichen im Schatten stehen oder direkt von der Sonne angestrahlt werden. Hierdurch werden die Kontraste verringert, was ein Erkennen schwer bis unmöglich macht.

Um auch diese Verkehrszeichen zu erkennen, kann der von der Helligkeit unabhängige HSV-Farbraum verwendet werden. In diesem kann man selbst bei schlechten Lichtverhältnissen noch Farbkontraste extrahieren. Eine weitere Möglichkeit zur Risikominimierung ist die Nutzung eines adaptiven Schwellwertverfahren. Diese Verfahren erkennen auch bei besonders dunklen oder hellen Stellen noch kleinste Konturen.

⁴³ (Heilck, 2016, S. 5)

⁴⁴ (Zhang, Huang, Jin, & Li, 2017, S. 2)

4.4.1.2 Partielle Verdeckung

Die teilweise Verdeckung von Verkehrszeichen tritt häufig in ländlichen Gebieten und Seitenstraßen auf, in denen Bäume oder Büsche vor Verkehrszeichen wachsen. Die Sicht auf Verkehrszeichen kann jedoch auch durch parkende Autos oder vorausfahrende LKWs eingeschränkt werden.

Um das Problem zu umgehen, müssen die Algorithmen so geschrieben werden, dass sie auch bei 50% eines Schildes einen Treffer liefern. Es muss jedoch beachtet werden, dass die Falscherkennungsrate hierdurch auch ansteigt.

4.4.1.3 Überladene Bilder

Vor allem in Innenstädten befinden sich im Kamerabild sehr viele Konturen und für die Erkennung unwichtige Details, wie zum Beispiel Werbeplakate. Auch im Bild-Hintergrund oder auf der Rückseite von LKWs können Verkehrszeichen vorhanden sein, die für die aktuelle Fahrstrecke keine oder nur eine geringe Rolle spielen.

Damit die überflüssigen Schilder die Verarbeitung nicht unnötig verzögern oder falsche Ergebnisse verursachen, sollte ein Filter verwendet werden, der Vordergrund und Hintergrund des Bildes separiert. Eine einfache Möglichkeit wäre eine Bewegungsextraktion, denn Objekte welche nah beim Betrachter sind bewegen sich schneller als der Hintergrund oder weit entfernte Objekte.

Eine weitere, jedoch aufwendigere Methode zur Vermeidung dieses Risikos ist die Verwendung eines Stereo-Kamerasystems. Mithilfe von zwei, in einem bestimmten Abstand und verschiedenen Winkeln in das Fahrzeug montierte Kameras kann zu jedem Bildpunkt eine Entfernung angegeben werden.⁴⁵

4.4.1.4 Kleine Objekte

Die, im Bild nur eine geringe Fläche bedeckenden, kleinen Verkehrszeichen können zwar detektiert, aber kaum klassifiziert werden, da in den wenigen Pixeln nur ein geringer Informationsgehalt enthalten ist.

Eine Methode der Risikominimierung ist die Verwendung von hochauflösendem Eingangsmaterial. Es sollte aber beachtet werden, dass die Verarbeitung von höheren Auflösungen auch mehr Rechenleistung benötigt.

4.4.2 Risiken bei der Bildklassifikation

4.4.2.1 Ungenügende Trainingsdaten

In der Anwendungsphase der Bildklassifikation kann es vorkommen, dass sehr schlechte Klassifizierungsergebnisse zurückgegeben werden. In diesem Fall sollte eine erneute Trainingsphase durchgeführt werden, in der weitere Bilder in das neuronale Netz eingefügt werden. Auch eine Spezialisierung hinsichtlich des Aufnahmewinkels und Beleuchtungssituationen kann Sinn machen, wenn sonst keine gute Erkennungsrate erreicht werden kann.

⁴⁵ (Gehrig, 2018)

4.4.2.2 *Overfitting*

Unter dem Begriff Overfitting (zu Deutsch: Überanpassung) versteht man ein Problem, das beim Trainieren von neuronalen Netzen auftreten kann.

Einfach gesagt beschreibt das Problem die Situation, bei der ein Eingabebild nicht richtig klassifiziert werden kann, weil das neuronale Netzwerk die falschen Merkmale zur Unterscheidung der Klassen generiert hat. So kann es beispielsweise vorkommen, dass der Machine Learning Algorithmus aufgrund der Trainingsdaten „denkt“, alle Schilder, die vor einer grünen Hecke stehen, seien Stoppschilder und alle Schilder, die nicht vor solchen Hecken stehen, seien beispielsweise „Achtung Baustelle“-Schilder.

Wenn nun ein „Achtung Baustelle“-Schild, das vor einer grünen Hecke steht, klassifiziert werden soll, so „denkt“ der Algorithmus aufgrund des Hecken-Merkmals, dass es sich um ein Stoppschild handelt.

Diese Überanpassung kann verhindert werden, indem in jeder Klasse abwechslungsreichere Bilder trainiert werden, im Beispiel also auch Bilder von Stoppschildern, die nicht vor einer grünen Hecke stehen.

5 Projektdurchführung

In diesem Kapitel wird die Entwicklung eines Programms beschrieben, das Verkehrsschilder in Videos und Webcam-Streams erkennen, beschriften und verfolgen kann.

Das von uns entwickelte Programm lässt sich in vier unabhängige Abschnitte aufteilen. Diese sind die grafische Benutzeroberfläche, die Erkennung, die Klassifikation und das Rahmenprogramm, das alles steuert. In den Modulen Erkennung und Klassifizierung ist der Großteil von Logik und Arbeitsaufwand enthalten, weshalb auf deren spezifische Implementierung auch in mehreren Unterkapiteln eingegangen wird.

Neben den, das Programm beschreibenden, Abschnitten, werden im Folgenden auch die Entwicklungsumgebung, die Software-Tests und die Bedienung und Installation des finalen Programms beschrieben.

5.1 Benutzeroberfläche

Das Erste, das ein Nutzer von der Verkehrszeichen-Erkennung sieht, ist die in Abb. 11 gezeigte grafische Benutzeroberfläche (GUI).

Das GUI bietet die wichtigsten Einstellmöglichkeiten auf einen Blick. Dies soll auch fachfremden Personen ein einfaches Nutzen der Software ermöglichen.

Die wichtigste Einstellung ist hierbei die Eingangsquelle, die eine Videodatei oder ein Kamera-Stream sein kann. Anschließend kann gewählt werden, ob die Ausgabe nur angezeigt oder auch gespeichert werden soll. Die Auswahlmenüs sind hier möglichst schlicht gehalten und verwenden bereits bekannte Elemente, was ein schnelles und fehlerfreies Bedienen ermöglicht.

Die weiteren Menüpunkte der GUI sind optionale Einstellungsmöglichkeiten, mit welchen Features individuell hinzu oder abgeschaltet werden können.

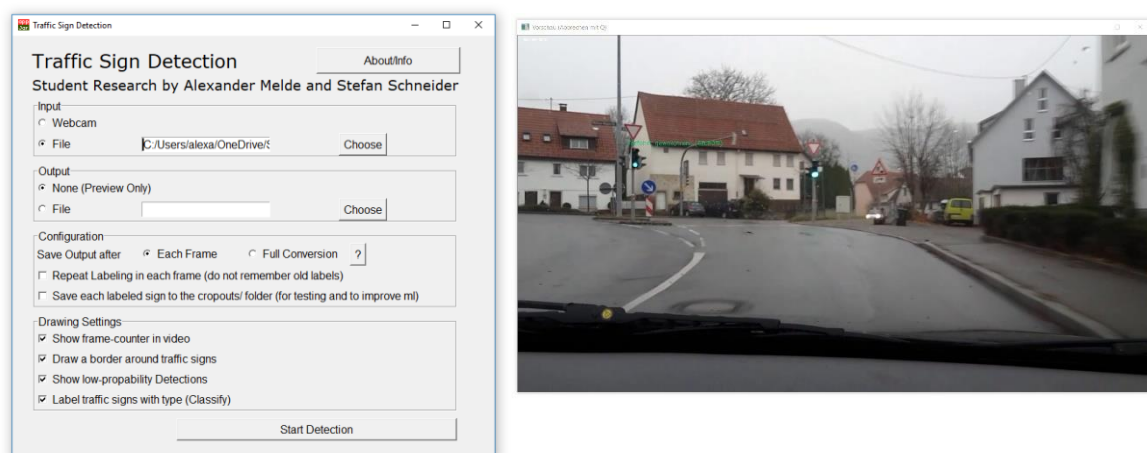


Abb. 11: Grafische Oberfläche des entwickelten „Traffic Sign Detection“-Programms

Wird die Verkehrszeichen-Erkennung nun gestartet, so werden alle eingestellte Parameter von der GUI an das Rahmenprogramm übergeben.

5.2 Rahmenprogramm

Das Rahmenprogramm prüft zuerst, ob sich die Eingangsdatei öffnen lässt und startet daraufhin das Einlesen der Datei. Soll die Ausgabe abgespeichert werden, so wird zusätzlich zu dem grafischen Fenster, in welchem die Einzelbilder angezeigt werden, auch noch die Ausgabedatei erstellt.

Nun wird das Video bildweise eingelesen und entsprechend der Konfiguration werden die Einzelbilder an Unterprogramme übergeben.

Bei einem Durchlauf mit Erkennung und Klassifikation sieht das wie folgt aus:

1. Das Hauptprogramm gibt ein Einzelbild an die Erkennung und diese wendet ihre Algorithmen an um nach Verkehrszeichen zu suchen.
2. Werden ein oder mehrere Verkehrszeichen erkannt, bekommt das Hauptprogramm eine diese enthaltende Liste zurück. Die Liste bezieht sich immer auf das entsprechende Einzelbild und enthält die Koordinaten, die Größe, sowie die Form der erkannten Verkehrszeichen.
3. Das Rahmenprogramm gibt die Koordinaten an den Klassifikator weiter. Dieser untersucht die Ausschnitte und gibt die Wahrscheinlichkeit für ein erkanntes Zeichen sowie dessen Name zurück.
4. Das Rahmenprogramm gibt diese Daten an die Ergebnisdarstellung weiter, bei der bei einem ausreichend hohen Prozentsatz ein Rahmen um das Verkehrszeichen gezeichnet und eine Beschriftung über den Typ eingefügt wird.
5. Das Einzelbild wird daraufhin live auf dem Desktop ausgegeben.

Um den Klassifikator zu entlasten, wird vom Hauptprogramm noch ein Tracker initiiert, der Verkehrszeichen nach ihrer ersten Detektion und Klassifizierung verfolgt und somit eine erneute Klassifizierung überflüssig macht. Diese Objektverfolgung wird in Abschnitt 5.5 näher beschrieben.

Da die Erkennung und Klassifizierung in Abschnitt 5.3 und Abschnitt 5.4 genauer beschrieben wird, soll hier nichtmehr weiter darauf eingegangen werden.

Bei der Implementierung wurden die Module in einer bestimmten Reihenfolge entwickelt.

Zuerst wurden die Erkennung und Klassifizierung parallel, jedoch unabhängig voneinander, entwickelt. Erst als beide Programme verwertbare Ergebnisse lieferten, wurden sie durch das Rahmenprogramm miteinander verbunden. Durch im Vorfeld fest definierte Schnittstellen führte dies zu keinen Problemen. Um die Nutzbarkeit zu erhöhen, wurde schlussendlich noch die geplante grafische Eingabemöglichkeit geschaffen. Ohne diese müsste sich jeder Nutzer in den Quellcode einarbeiten.

Gerade die fest definierten Schnittstellen ermöglichen eine hohe Modularität des Programms.

So könnte jederzeit eine andere GUI, ein anderer Klassifikationsalgorithmus oder eine andere Erkennung in das Rahmenprogramm integriert werden. Sie müssten lediglich die verwendeten Schnittstellen unterstützen.

5.3 Erkennung von Verkehrszeichen

Nachfolgend wird die Erkennung und Extraktion von Verkehrszeichen aus Video oder Bildmaterial beschrieben. Hierfür wird die Bibliothek OpenCV verwendet.

Wie im vorherigen Abschnitt beschrieben, erhält das Erkennungs-Modul einzelne Videobilder vom Rahmenprogramm. Die Einzelbilder werden auf Verkehrszeichen untersucht, wobei die gefundenen Schilder in einer Liste gespeichert werden, die die Koordinaten im Bild sowie die Form des Schildes enthält. Die Liste wird anschließend an den Klassifikations-Algorithmus übergeben, damit dieser die Verkehrszeichen korrekt beschriften kann.

Die Erkennung der Schilder besteht aus zwei Teilen, einer Kreis- und einer Quadrat-Detektion. Die Kreiserkennung versucht alle runden Schilder zu erkennen und die Quadraterkennung soll jegliche eckigen Schilder aus den Bildern extrahieren. Da die Algorithmen unabhängig voneinander arbeiten, werden sie parallel ausgeführt, was auf Multi-Core-Systemen zu einer höheren Performance führt.

Die Listen mit den Ergebnissen werden von einem speziellen Algorithmus zusammengeführt, um keine Duplikate zu enthalten.

5.3.1 Kreis-Detektion

Für die eigentliche Kreis-Detektion wird der HoughCircles Algorithmus der OpenCV Bibliothek genutzt. Daher ist es notwendig, alle falsch positiv als Schild erkennbaren Kreise aus dem Bild zu entfernen, da diese von dem Algorithmus sonst auch erkannt werden würden. Hierfür wird zuerst das komplette Bild einer Farbfilterung unterzogen.

Da das Bild im RGB-Farbraum vorliegt, wird dieses zuerst in den HSV-Farbraum konvertiert. Der HSV Farbraum legt ein Pixel nicht als Rot-, Grün- oder Blau-Wert, sondern als Wert auf einem Farbkreis ab. Die weiteren Parameter sind die Helligkeit und die Sättigung des Pixels.

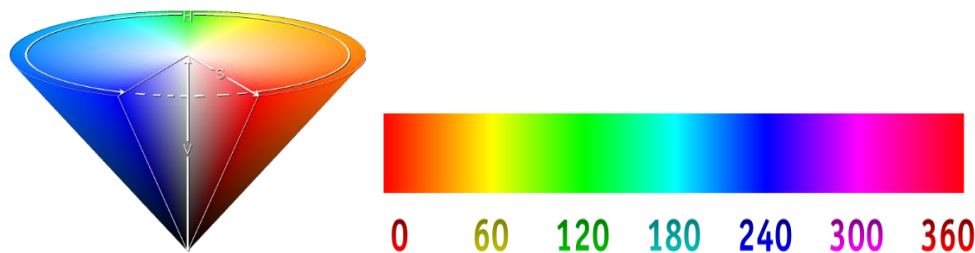


Abb. 12: HSV-Farbraum

(Wikipedia-Autoren, HSV-Farbraum, 2018)

Farbwert H , Sättigung S , Dunkelstufe V

Die Speicherung als Wert in einem Farbkreis ermöglicht ein leichtes Filtern von verschiedenen Farben, da geschaut werden kann ob sich die Farbe zwischen dem oberen und unteren angegebenen Winkel im Farbkreis befindet. Die Speicherung der Sättigung einer Farbe ermöglicht weiter auch das Erkennen von schon ausgebleichten Schildern. Und der Helligkeitswert lässt auch zu dunkle oder helle Schilder gut filtern, was den Algorithmus weniger anfällig für schlechte Lichtverhältnisse macht.

Nach der Konvertierung wird das Bild mit einem Filter bearbeitet, welches das Bild leicht verschwimmen lässt, um Rauschen zu unterdrücken. Nun kann mit der Farbfilterung begonnen werden. Hierfür wird das Bild nacheinander auf Rot, Gelb und Blau gefiltert. Die Masken werden nun logisch ODER verknüpft. Die Maske, die für alle drei genannten Farben gilt, wird nun mit dem Original Bild UND-Verknüpft. Das Ergebnis ist ein Bild welches nur aus Rot, Gelb und Blau besteht.



1) Originalbild



2) Maske des Farbfilters

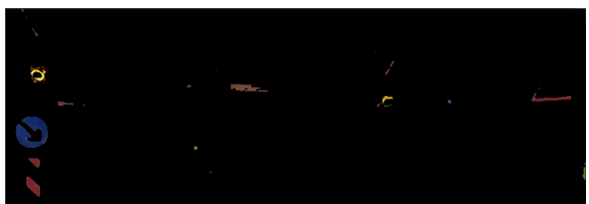
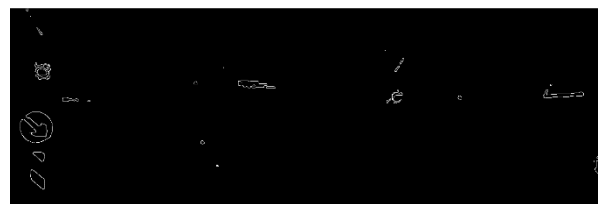

3) Maske des Farbfilters UND-Verknüpft
mit dem Originalbild

4) Extrahierte Kanten aus dem
gefilterten Bild 3

Abb. 13: Verschiedene Bearbeitungsschritte der Kreiserkennung

In diesem Bild werden nun die Kanten gefiltert. Aus dem Bild der Kanten extrahiert der HoughCircles Algorithmus alle zu entdeckenden Kreise. Hierbei wird schon eine Filterung der Kreisgrößen durchgeführt. Zu große oder kleine Kreise werden ignoriert.

Je nach verwendeter Kamera und Bildwechselfrequenz (Framerate), lässt sich das Ergebnis dieses Algorithmus noch weiter optimieren. So kann beispielsweise der Schwellwert für die Detektion von Kreisen erhöht werden. Das Ergebnis wird in eine für uns leichter zu verarbeitende Form transformiert, in der die Kreise mit der Pixelposition des Mittelpunkts und dem Radius angegeben werden.

Da bei Schildern oft ein innerer und äußerer Kreis erkannt wird, ist eine Filterung notwendig, damit diese Schilder nicht doppelt zum Klassifikator gelangen. Auch kommt es gelegentlich vor, dass Äste von Bäumen als eine Ansammlung von kleinen Kreisen erkannt werden. Äste überschneiden sich sehr oft, weshalb eine Funktion erstellt wurde, die prüft, ob sich die erkannten Kreise überschneiden.

Hierzu prüft die Funktion alle Kreise und vergleicht, ob die Entfernung von dem eigenen Mittelpunkt zum Mittelpunkt eines anderen Kreises kleiner ist als der eigene Radius. Sollte dies zutreffen, so wird der Kreis, mit dem verglichen wurde, nicht in die Ergebnisliste kopiert. Schlussendlich enthält das Ergebnis daher nur Kreise, die alleinstehen, sowie den größten von ineinander liegenden Kreisen.

Die Laufzeit des Algorithmus ist im schlechtesten Fall n^2 , was bei einer hohen Anzahl von Schildern n viel Zeit kosten kann. Dennoch ist das hinsichtlich der Rechenzeit günstiger, als viele falsch erkannte Bilder an den Klassifikator zu geben.

5.3.2 Quadrat-Detektion

Der zweite Teil der Schilder-Erkennung spezialisiert sich auf das Erkennen von Schildern mit Geraden und Ecken. Hierzu zählen Gefahren-, Vorfahrts- und Haltezeichen. Um diese Schilder zu erkennen wird die Form der Konturen überprüft.



Abb. 14: Erkennung von möglichen Verkehrsschildern anhand der Form

Da Farbinformationen für die weitere Bearbeitung in diesem Algorithmus nicht notwendig sind, und da Farbbilder einen höheren Rechenaufwand benötigen, wird das dem Modul übergebene Bild zunächst in Graustufen umgewandelt.



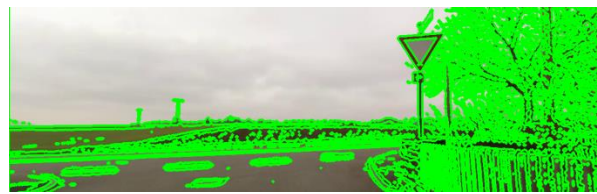
1) Originalbild



2) Bild in Grautönen



3) Bild nach dem Schwellwertverfahren



4) In das Original-Bild wurden alle gefundenen Konturen eingezeichnet.

Abb. 15: Verschiedene Arbeitsschritte der Quadrat-Detektion

Anschließend werden mit einem adaptiven Schwellwertverfahren Farbschwankungen hervorgehoben.

Schwellwertverfahren werden eingesetzt, um Bildausschnitte, die heller oder dunkler sind als ein definierter Wert, hervorzuheben oder auszublenden. Die eingelesenen Bilder sind meist nicht gleichmäßig ausgeleuchtet, weshalb ein adaptives Verfahren nach Gauß verwendet wird.

Das Verfahren schaut sich einen Bereich um einen Pixel an und berechnet anschließend einen einmaligen Wert. Dies wird für jeden Pixel wiederholt. Da Farbschwankungen bei fast jedem Schildtyp anzutreffen sind, wird so die Form des Schildes verdeutlicht.

Aus diesem Bild werden nun alle Konturen extrahiert und leicht approximiert. Das Approximieren fasst alle kleinen Konturen zusammen, was Effekte wie Unschärfe und Bildrauschen vermindert und die weitere Bearbeitung verbessert.

Gespeichert werden die Konturen in einer Baumstruktur, was die spätere Bearbeitung merklich effizienter macht, da nicht alle Konturen durchlaufen werden. Zuerst werden so die Konturen der ersten Stufen bearbeitet, und nur wenn bei diesen kein Verkehrszeichen dabei ist, werden die Konturen des nächsten Levels bearbeitet, also Konturen, die innerhalb der Konturen des vorherigen Levels liegen. Iteriert die Funktion zu tief, so wird automatisch abgebrochen.

Bei der Auswertung der Kontur wird zunächst ein Rechteckiger Bereich ermittelt, der die Kontur einschließt. Anschließend werden zu kleine Rechtecke, ähnlich wie bei der Kreis-Detektion, verworfen. Da das Rechteck in Idealfall quadratisch ist, werden im Anschluss die Seitenlängen miteinander verglichen. Weichen diese stark von einer quadratischen Form ab, so wird die Kontur verworfen. Erst nachdem dieser Tests bestanden wurden, wird die Kontur dem nächsten Schritt übergeben.

In diesem nächsten Schritt wird geprüft, ob es sich bei den Vektor-Objekten, welche aus der Kontur gewonnen werden, um Verkehrszeichen handelt. Hierfür wird die Kontur durchlaufen und von jeder Gerade werden Länge und Winkel in einem Vektor-Objekt gespeichert. Alle Vektor-Objekte einer Kontur werden anschließend Winkeln zugeordnet. Dies sind 0° , 45° , 60° und 90° . Vektor-Objekte mit anderen Gradzahlen werden in der Klasse separat gelistet.

Nach der Zuordnung wird die Auswertung gestartet. Diese prüft zuerst die Anzahl der Vektorobjekte um eine grobe Klassifizierung vorzunehmen. Bei drei Vektoren handelt es sich vermutlich um Gefahrenzeichen oder ein Vorfahrt-Achten-Zeichen, bei vier um Vorfahrtsschilder und bei acht Vektoren um Stoppschilder. Eine Plausibilitäts-Prüfung bei drei Konturen ist, ob zwei Objekte mit 60° und ein Objekt mit 0° vorhanden sind.

Da die Konturen von Schildern nicht immer korrekt eingelesen werden, gibt es noch weitere Fälle, die in dem Algorithmus zur Detektion eines Schilds führen. So reichen bei einem Dreieck auch drei gleichlange Konturen, von welchen mindestens zwei einen Winkel eines Dreiecks enthalten.

5.4 Klassifikation von Verkehrszeichen

Nach der im vorherigen Kapitel beschriebenen Erkennung eines möglichen Verkehrszeichens ist nur bekannt, an welcher Stelle im Bild sich ein Schild in welcher Größe und mit wie vielen Ecken befindet, aber nicht, um welches Schild es sich genau handelt. In diesem Abschnitt wird eine Möglichkeit beschrieben, wie die Art des Verkehrszeichens bestimmt werden kann.

Hierfür kommt eine Bildklassifizierung zum Einsatz, die auf Machine Learning basiert und das Framework „TensorFlow“ verwendet. Mithilfe des TensorFlow-Frameworks können beliebige Arten von Bildern klassifiziert werden. Es ist nicht nur möglich, zwischen allgemeinen Objekten wie Tieren, Menschen und Autos zu unterscheiden, sondern es ist auch möglich, verschiedene Katzenrassen oder Verkehrsschilder zu unterscheiden.

Damit eine solche Unterscheidung möglich ist, werden mindestens 30 Beispielbilder für jede Klasse benötigt. Der Klassifizierungsalgorithmus prüft dann, wie viele Ähnlichkeiten ein Bild mit den Beispielbildern der verschiedenen Klassen hat und berechnet zu jeder Klasse die Wahrscheinlichkeit, mit der das Bild in dieser Klasse ist.



Abb. 16: Ablauf einer Bildklassifizierung

Der Aufruf des Klassifikationsalgorithmus ist nach einer kurzen Einarbeitung in das Framework vergleichsweise einfach. Das Erstellen einer richtigen Klassenstruktur sowie das Sammeln und Beschriften von geeigneten Beispielbildern stellt eine größere Herausforderung dar.

Die einzelnen Teilschritten werden in den folgenden Unterkapiteln beschrieben.

5.4.1 Klassenstruktur

Zur Klassifikation von Verkehrszeichen ist es wichtig zu definieren, welche Klassen unterschieden werden sollen. Diese Arbeit soll speziell zwischen den verschiedenen deutschen Verkehrszeichen unterscheiden können. Daher wurden für die Klassifizierung die offiziellen Verkehrszeichen-Nummern und -Namen der Straßenverkehrsordnung (StVO) verwendet.⁴⁶

Das TensorFlow-Framework und das zugrundeliegende Convolutional Neural Network (CNN) benötigt die geordneten Beispieldaten in der folgenden Ordnerstruktur:

```

signs/
| - /101000/                # Gefahrstelle
|   | -img0001.jpg
|   | -img0002.jpg
|   | -img0003.jpg
| - /283020/                # Haltverbot (Ende)
|   | -img0001.jpg
| ...

```

Abb. 17: Ordnerstruktur für Beispieldaten

In den mit Nummern beschrifteten Ordnern liegen jeweils alle Bilder eines Verkehrszeichen-Typs, beispielsweise Bilder, die aus verschiedenen Winkeln, bei unterschiedlicher Belichtung oder vor anderen Hintergründen aufgenommen wurden.

Die Nummer des Ordners leitet sich von der in der Straßenverkehrsordnung (StVO) festgelegten Zeichenummer der auf den jeweiligen Bildern abgebildeten Verkehrszeichen ab.

Hierfür wurde die folgende Formel verwendet:

$$\text{Ordernummer} = \text{Zeichenummer} * 1000 + \text{Zusatznummer}$$

Aus dem Zeichen „Ende einer Fußgängerzone“ mit der Nummer 242.2 wird beispielsweise der Ordnername 242002.

Die Klassifizierungsfunktion nimmt, wie bereits erwähnt, ein auf ein erkanntes Verkehrszeichen zugeschnittenes Bild entgegen und gibt für jede Ordernummer zurück, wie wahrscheinlich es ist, dass das übermittelte Bild dem gleichen Verkehrszeichentyp angehört.

In der Ausgabe, die der Benutzer sieht, wird die Ordernummer mit der höchsten Wahrscheinlichkeit ausgewählt und anhand einer einfachen Zuordnungstabelle wird diese Ordernummer dann durch die offizielle Bezeichnung laut StVO⁴⁶ ersetzt.⁴⁷

⁴⁶ Wie im Laufe dieser Arbeit bemerkt wurde, unterscheiden sich die Namen und Nummern der Verkehrszeichen in verschiedenen Versionen der StVO. Als bebilderte Übersicht wurde zunächst (Barthl, 2013) verwendet, was sich später aber als nicht-vollständig und veraltet herausgestellt hat. Eine aktuellere bebilderte Übersicht mit dem Verkehrszeichenkatalog der StVO von 2017 befindet sich in (Korsch, 2017). Als Referenzwerk bei Unklarheiten wurde der offizielle Gesetzestext der StVO (Bundesregierung, 2017) verwendet.

⁴⁷ Beim Einzeichnen der Beschriftungen ins Video wurden Umlaute und Sonderzeichen soweit nötig durch im ASCII-Zeichensatz verfügbare Zeichen ersetzt.

5.4.2 Gewinnung von Trainingsdaten

Für jeden Verkehrszeichentyp müssen mindestens 30 Bilder zum „Trainieren“ der Klassifizierung vorliegen. In einem ersten Test haben wir hierfür einfach Bilder einer Internet-Bildsuchmaschine in die passenden Ordner heruntergeladen. Nach einigen Versuchen hat sich herausgestellt, dass die Klassifizierung besser funktioniert, wenn Bilder aus selbstaufgenommenen Videos für das Training verwendet werden, da diese die bei einer Autofahrt typischen Bedingungen hinsichtlich Belichtung, Rauschen und Verzerrung am ehesten erfüllen.

Hierfür wurden einige private Fahrten im Projektzeitraum mithilfe von Smartphones oder Action-Kameras aufgezeichnet. Auf diese Weise sind über vier Stunden Videomaterial entstanden. Bei der Auswahl der Strecken wurde darauf geachtet, dass sowohl Stadt-, Überland- und Autobahn-Fahrten sowie Fahrten bei verschiedenen Wetter- und Lichtbedingungen aufgezeichnet wurden.

Aus diesen Videos müssen nun Verkehrszeichen extrahiert werden. Hierfür wurde die in Abschnitt 5.3 beschriebene Verkehrszeichenerkennung um eine Funktion erweitert, die alle erkannten Zeichen als zugeschnittenes Bild in einen Ordner abspeichert. Mit dem oben erwähnten Videomaterial konnten rund 16.000 mögliche Verkehrszeichen zugeschnitten gespeichert werden (siehe Abb. 18).



Abb. 18: Zugeschnittene mögliche Verkehrszeichen vor der Klassifizierung

Die zugeschnittenen Bilder sind noch nicht klassifiziert, weshalb eine händische Einordnung in die im vorherigen Abschnitt beschriebene Struktur notwendig ist. Nach dem Entfernen aller „False-Positives“, also Zuschnitten, in denen sich gar kein Verkehrszeichen befindet, wurden rund 4.000 Bilder von Verkehrszeichen per Hand klassifiziert.

Für insgesamt 39 verschiedene Klassen konnten mehr als 30 Beispielbilder klassifiziert werden, sodass diese nun für die Klassifizierung genutzt werden können. Noch nicht für die Klassifizierung geeignet sind 61 Typen von Verkehrszeichen, bei denen weniger als 30 Beispielbilder vorliegen.

Die händische Klassifizierung ist mit einem hohen zeitlichen Aufwand verbunden. Im Laufe der Arbeit wurde die Funktion zum Abspeichern der Bild-Zuschnitte erweitert, sodass diese auch den wahrscheinlichsten Verkehrszeichentyp in den Namen der Bilddatei schreibt. So ist eine Filterung möglich, die die Klassifizierungs-Arbeit ein wenig erleichtert. Aufgrund der während der Entwicklung hohen Rate von Falscherkennungen, beispielsweise weil viele Verkehrszeichentypen noch keine eigene Klasse hatten, gab es aber trotz diesem Ansatz noch viele falsch einsortierten Schilder.

Eine Alternative zu dieser manuellen Methode der Ermittlung von Trainingsdaten ist das Nutzen von öffentlich verfügbaren sortierten Datensätzen. Das Institut der Neuroinformatik der Ruhr-Universität Bochum hat im Jahr 2010 eine aus über 50.000 Bildern bestehende Sammlung von 43 verschiedenen Verkehrszeichentypen veröffentlicht.⁴⁸ Dieser Datensatz ist Teil des Wettbewerbs „German Traffic Sign Recognition Benchmark“ (GTSRB), bei dem verschiedenen Forschergruppen eine Verkehrszeichen-erkennung mit den damals verfügbaren Technologien umsetzten. Das hier genutzte TensorFlow Framework erschien im Jahr 2015, kann aber ebenfalls mit diesem Datensatz arbeiten.

Für unsere Zwecke relevant ist die Datei „Training dataset – Images and annotations“, die unter folgendem Link heruntergeladen werden kann:

Link 4: GTSRB Dataset: http://benchmark.ini.rub.de/Dataset/GTSRB_Final_Training_Images.zip

Die einzelnen Bilder der Verkehrsschilder sind in den fortlaufenden Unterordnern des Verzeichnisses /GTSRB/Final_Training/Images enthalten. Zum Erreichen der in Abb. 17 gezeigten Ordnerstruktur müssen die Ordner lediglich umbenannt werden.

Beim Betrachten der in diesem Datensatz enthaltenen Bilder fällt schnell auf, dass sich viele der Bilder sehr stark ähneln. Einige Bilder sind aus nur leicht anderen Winkeln, bei anderer Belichtung oder aus einer anderen Entfernung aufgenommen wurden. Bei unserer Klassifizierungsmethode sind diese ähnlichen Bilder nur wenig hilfreich. Um das Trainieren der Klassifizierung zu beschleunigen und um Overfitting⁴⁹ zu verhindern, wurden sehr ähnlich aussehende Bilder entfernt.

In einem letzten Schritt müssen die im .ppm Format vorliegenden Bilder in das modernere .jpg Format umgewandelt werden, damit Sie von unserem Trainingsskript genutzt werden können. Hierfür kann das in vielen Linux-Distributionen standardmäßig installierte Kommandozeilentool mogrify genutzt werden:

```
> cd GTSRB/Final_Training/Images/
> mogrify -format .jpg */*.ppm
> find . -name "*.ppm" -type f -delete
```

Abb. 19: Konvertieren der .ppm-Bilder zu .jpg

Die oben genannten Befehle erstellen zunächst eine .jpg Datei zu jedem .ppm Bild. Mit dem letzten Befehl werden die alten .ppm Dateien gelöscht. Um lediglich eine Vorschau über die zu löschenden Dateien zu bekommen kann die -delete Option am Ende des Befehls zunächst weggelassen werden.

In dieser Arbeit wurden die selber aufgenommenen Bilder mit den heruntergeladenen kombiniert, um schneller auf die 30 erforderlichen Beispielbilder pro Klasse zu kommen. Die GTSRB-Bilder erhöhen durch den anderen Bildzuschnitt und die anderen Belichtungen zudem die Varietät der Trainingsdaten.

⁴⁸ (Stallkamp, Schlipfing, Salmen, & Igel, 2011)

⁴⁹ Eine Erklärung dieses Begriffs befindet sich in Abschnitt 4.4.2.2.

5.4.3 Trainieren der gewonnenen Trainingsdaten

Die gesammelten Trainingsdaten können vom neuronalen Netzwerk nur genutzt werden, wenn aus diesen ein Modell trainiert wurde. Hierfür kommt ein von den TensorFlow-Entwicklern geschriebenes Python Skript zum Einsatz.⁵⁰

Die in Abschnitt 3.3.2 gezeigten theoretischen Grundlagen werden für den Einstieg mittels Framework nicht benötigt, dennoch soll nicht unerwähnt bleiben, dass ein auf Inception v3 trainiertes künstliches neuronales Faltungsnetz (CNN) mit einer SoftMax-Pooling-Funktion verwendet wurde.

Die Erstellung des Modells kann je nach Anzahl der Trainingsbilder und deren Bildgröße zwischen fünf Minuten und mehreren Stunden in Anspruch nehmen. Ruft man das Skript ein zweites Mal auf, so werden nur neu hinzugefügte Bilder betrachtet, was die Erstellung des Modells beschleunigt.

```
> python classification/retrain.py \
--bottleneck_dir     =${PWD}/tf_files/bottlenecks \
--how_many_training_steps=500 \
--model_dir          =${PWD}/tf_files/inception \
--output_graph       =${PWD}/tf_files/retrained_graph.pb \
--output_labels      =${PWD}/tf_files/retrained_labels.txt \
--image_dir          =${PWD}/data/signs
```

Abb. 20: Start des Modell-Trainings⁵¹

Für das Training werden die vorsortierten Beispielbilder aus Abschnitt 5.4.2 zu gleichen Teilen in Trainings- und Testbilder aufgeteilt. Am Ende des Trainings mithilfe der Trainingsbilder wird das Modell auf die Testbilder angewendet, um zu ermitteln, wie oft die Klassifizierung Erfolg hatte. Es erscheint eine Meldung der folgenden Art:

```
>>> INFO:tensorflow:Final test accuracy = 80.6% (N=474)
```

Abb. 21: Meldung am Ende des Modell-Trainings

Die Angabe 80.6% bedeutet, dass 80.6% der getesteten Bilder korrekt klassifiziert wurden. Die Genauigkeit kann durch Vergrößerung des Trainingsdatensatzes erhöht werden.

Das neue Modell liegt anschließend im Unterordner „tf_files“. Wenn die Klassenstruktur aus Abschnitt 5.4.1 geändert (und nicht nur ergänzt) wird, sollte der gesamte Ordner „tf_files“ vor dem erneuten Trainieren gelöscht werden, damit nicht mehr benötigte Daten und irrelevante Teile des neuronalen Netzes gelöscht werden und Speicherplatz gespart wird.

⁵⁰ (The TensorFlow Authors, 2018)

⁵¹ (Raval, 2016)

Hinweis: Der gezeigte Code kann nur ausgeführt werden, wenn die Entwicklungsumgebung bereits erfolgreich eingerichtet wurde. Eine Anleitung hierzu befindet sich in Abschnitt 5.9.1.

5.4.4 Skript zur Klassifizierung eines Verkehrszeichens

Die Beschriftung der Einzelbilder erfolgt durch die Funktion `classification/analyse_image`. Diese nimmt ein OpenCV-Bild entgegen und gibt ein Array von Klassen und Wahrscheinlichkeiten zurück (siehe Abb. 16).

In der Ausgabe des Skripts finden sich nun die Nummern der verschiedenen Klassen des Trainingsdatensatzes sowie die Wahrscheinlichkeit, dass es sich bei dem Objekt im Bild um ein Objekt dieser Klasse handelt.

Eine dahinterliegende Funktion im Rahmenprogramm übersetzt diese Nummern und schneidet nicht mehr benötigte Ergebnisse ab.

5.5 Verfolgung von Verkehrszeichen

Mit den bisher beschriebenen Methoden wird für jedes erkannte Schild eine Klassifikation durchgeführt. Leider sind Klassifikationen der rechenaufwendigste Teil des Algorithmus, weshalb eine Möglichkeit eingeführt wurde, bereits klassifizierte Schilder über mehrere Einzelbilder hinweg zu verfolgen und die bestimmte Klasse beizubehalten.

Für jedes erkannte mögliche Verkehrszeichen wird ein Tracker gestartet, der versucht, den Bildausschnitt in den nächsten Einzelbildern des Videos wiederzuerkennen. Gelingt dies, so ist keine vollständige neue Klassifikation notwendig.

Ein Schild wird intern durch eine Folge von mit der jeweiligen Einzelbild-Nummer verknüpften Positionen angegeben, es kann also in verschiedenen Einzelbildern an verschiedenen Positionen vorkommen, und dennoch wird der Zusammenhang gespeichert, dass es sich um ein und dasselbe Schild handelt.

In weiteren Feldern der Datenstruktur werden, wie in Abb. 22 gezeigt, auch die Form des Verkehrszeichens, was zum Zeichnen des Rahmens um das Schild verwendet wird, und auch die in vorherigen Frames bestimmten Klassifizierungen (Labels) des Schilds gespeichert.

```
class Sign:
```

```

    pos          = (12: (102,240,100,100),          #einzelbildnr: (x,y,w,h)
                    13: (105,245,99,99), ...)

    edgecount    = 3                                #triangle=3, square=4, circle=0

    labels       = (12: (142010, 0.84353), ...)      #einzelbildnr: signid, probability

    tracker      = None                             #cv2.TrackerKCF_create()
```

Abb. 22: Datenstruktur für Schilder

Dank der Verkehrszeichen-Verfolgung ist es nun auch möglich, Statistiken zur Verteilung von Verkehrszeichentypen im Video zu messen, ohne auf einzelne Einzelbilder eingehen zu müssen.

So kann beispielsweise die Anzahl der Schilder im Video ausgewertet werden oder die Häufigkeit von Schildern einer bestimmten Klasse in zwei Videos verglichen werden.

5.6 Software-Test

Die Implementierung wurde hinsichtlich verschiedener Kriterien kontinuierlich getestet. Die Kriterien sowie die jeweiligen Testergebnisse werden in den folgenden Unterkapiteln erläutert.

5.6.1 Detektionsrate

Die Erkennungsrate des Detektions-Algorithmus ist wie folgt definiert:

$$\text{Detektionsrate} = \frac{\text{Anzahl detektierte Verkehrszeichen}}{\text{Anzahl Verkehrszeichen im Video}}$$

Hieraus lässt sich ableiten, wie viele Schilder aus dem Video prozentual erkannt werden.

Während der Entwicklung der Implementierung wurden die Detektion zunächst mithilfe ausgewählter Beispielbilder getestet. Hierzu wurde nach jeder Code-Änderung überprüft, wie viele Schilder korrekt erkannt werden. In späteren Entwicklungsphasen wurden die Beispielbilder durch Videos ersetzt.

Bei Videos müssen Schilder nicht im allerersten Einzelbild, in dem sie auftreten, detektiert werden, da die Schilder noch für einige Einzelbilder im Bild bleiben. In der Praxis verbessert dies die Detektionsrate deutlich, da einige Einzelbilder für eine Detektion selbst für Menschen eine echte Herausforderung sind.

5.6.2 Fehlerrate der Detektion

Mithilfe der folgenden Formel kann bestimmt werden, in prozentual wie vielen Einzelbildern des Videos es zu „false-positives“ kommt, also Objekte als Verkehrszeichen erkannt werden, obwohl diese gar keine Verkehrszeichen sind.

$$\text{Fehlerrate}_{\text{Detek.}} = \frac{\text{Anzahl Einzelbilder mit fälschlich als Verkehrszeichen detektierten Objekte}}{\text{Anzahl Einzelbilder des Video}}$$

Die Einzelbilder mit Fehlerkennungen wurden analysiert, sodass die Algorithmen so angepasst werden konnten, dass die jeweiligen Fehler nicht mehr auftreten.

Wird ein Tracker zur Objektverfolgung eingesetzt, so berechnet sich die Fehlerrate anhand der Anzahl der verfolgten Objekte, und nicht anhand der Anzahl der Einzelbilder.

5.6.3 Erfolgsrate der Klassifizierung

Beim Klassifizierungsalgorithmus wurde getestet, in wie vielen Fällen das „wahrscheinlichste“ Ergebnis des Algorithmus mit dem getesteten Verkehrszeichen übereinstimmt.

$$\text{Erfolgsrate}_{\text{Klassifizierung}} = \frac{\text{Anzahl erfolgreich klassifizierte Verkehrszeichen}}{\text{Anzahl klassifizierte Verkehrszeichen}}$$

Wie bereits im Abschnitt 5.4.2 („Gewinnung von Trainingsdaten“) erwähnt, wurden alle nicht erfolgreich klassifizierte Verkehrszeichen, sowie auch einige korrekt erkannten Testbilder anschließend dem Trainingsdatensatz hinzugefügt, sodass das getestete Verkehrszeichen-Bild nach dem nächsten Trainieren des neuronalen Netzes korrekt erkannt wird.

Es sollte beachtet werden, dass nach dieser Definition auch die nicht im Trainingsdatensatz vorhandenen getesteten Schilder in die Erfolgsrate einfließen. Diese Schilder kann der Algorithmus nicht bestimmen, da er diese noch nicht „gelernt“ hat.

5.6.4 Schwellwert zur sicheren Klassifikation

In das dem Benutzer präsentierte Video werden nur Verkehrszeichen als erkannt eingezeichnet, die mit einer hohen Wahrscheinlichkeit einer bestimmten Klasse angehören.

Wenn ein Bildausschnitt nur mit einer geringen Wahrscheinlichkeit einer bestimmten Verkehrszeichen-Klasse angehört, dann ist es wahrscheinlich, dass der Bildausschnitt kein oder ein nicht im Trainingsdatensatz enthaltenes Verkehrszeichen enthält.

Der Erfolg ist gegeben, wenn nicht vorhandene Schilder als falsch erkannt werden. Die Erfolgsrate des Schwellwerts sollte daher möglichst groß sein:

$$Erfolgsrate_{Schwellwert} = \frac{\text{Anzahl verworfene Bildausschnitte}}{\text{Anzahl der kein Verkehrszeichen enthaltenden Bildausschnitte}}$$

Neben der Änderung des Schwellwerts führt auch die Erweiterung der Klassenvielfalt des Trainingsdatensatzes zu einer veränderten Erfolgsrate.

5.6.5 Eindeutigkeit der Klassifikationsergebnisse

Ein weiterer Parameter zur Bestimmung der Qualität des Klassifizierungsalgorithmus ist die Eindeutigkeit der Klassifikationsergebnisse, also die Differenz zwischen den Wahrscheinlichkeiten der ersten beiden Ergebnisse.

$$Eindeutigkeit_{Klassifikation} = \text{Wahrschk. erstes Ergebnis} - \text{Wahrschk. zweites Ergebnis}$$

Eine hohe Eindeutigkeit führt zu einer Verbesserung der Erfolgsrate der Klassifikation.

5.6.6 Zeitmessung

Die Berechnungszeit meint in diesem Test die Gesamtdauer zur Analyse eines Einzelbilds. Diese setzt sich aus den Laufzeiten der Erkennung und Klassifikation zusammen. Als Messkriterium kann neben der Berechnungszeit für definierte Beispielbilder auch der Durchschnitt mehrerer Messungen herangezogen werden.

$$\text{Durchschnittliche Berechnungszeit} = \frac{\text{Summe aller Berechnungszeiten}}{\text{Anzahl Messwerte}}$$

Darüber hinaus wurde auch die Gesamtlaufzeit des Programms für ein bestimmtes Video getestet, die in Abhängigkeit von der Videolänge als Messkriterium genommen werden kann:

$$\text{relative Verarbeitungsdauer} = \frac{\text{Dauer zur Verarbeitung eines Videos}}{\text{Videolänge}}$$

Diese relative Verarbeitungsdauer ist nicht von der Videolänge, dafür aber vom Videoinhalt, also zum Beispiel von der Anzahl der enthaltenen Verkehrszeichen abhängig, und deshalb nur bedingt zur Qualitäts-Messung einsetzbar.

Durch regelmäßigen Messungen konnten die zeitlichen Auswirkungen von Änderungen am Code nachgewiesen werden. Mit besonderem Interesse wurden auch langsame Programmabschnitte untersucht, um gezielt Teile des Algorithmus zu optimieren.

5.6.7 Funktionstest

Die Funktionalität wurde primär durch die Entwickler überprüft, indem beispielsweise die Auftrittshäufigkeit von Programmfehlern in der aktuellen Version oder die Korrektheit des Ergebnisses überprüft wurden.

Die Funktionsfähigkeit ist von den Eingangsparametern abhängig, weshalb verschiedene Konfigurationen, Videolängen und Videoformate getestet wurden.

Um Teile des Programms genauer zu analysieren, wurden verschiedene Funktionen zur Eingrenzung von Fehlern programmiert und eingesetzt, die beispielsweise zusätzliche Ausgaben in der Konsole ermöglichen oder Rahmen und Beschriftungen in das Bild einzeichnen. Einzelne Funktionen des Programms können auch über die Benutzeroberfläche vor Start der Verarbeitung ein- und ausgeschaltet werden, um potentielle Fehlerquellen ausschließen zu können.

5.6.8 Testergebnisse

Nachdem die Entwicklung der Verkehrszeichenerkennung im Rahmen dieser Arbeit abgeschlossen wurde, konnte ein finaler Test zur Bestimmung der Algorithmus-Performance durchgeführt werden.

Der Test fand auf einem Computer mit Windows 10, einem Intel Core i5 2500K @ 3.30GHz Prozessor, einer NVIDIA GeForce GTX 1070 8 GB Grafikkarte und 8 GB DDR3 Arbeitsspeicher statt. Es wurde die kompilierte Version des Programms in seiner Standardkonfiguration verwendet.

Für den Test wurde ein selbstaufgezeichnetes Video einer regnerischen und stark verschwommenen Überland-Fahrt verwendet. Dieses Video stellt bewusst keinen Idealfall dar, sondern eine echte Herausforderung.



Abb. 23: Beispielszene aus dem Test-Video

Wie in dem Beispielbild Abb. 23 zu sehen, sind die meisten Frames unscharf, und die Schilder auch für Menschen nicht gut zu erkennen.

Das Video hat das Format .mp4, eine Framerate von 30 FPS und ist 6:16 Minuten lang.

In dem getesteten Video kamen insgesamt 45 für die Erkennung relevante Verkehrsschilder vor, von denen 7 erkannt und 5 korrekt klassifiziert wurden. Dank dem Schwellwert-Algorithmus wurden keine Objekte fälschlicherweise als Verkehrsschilder identifiziert.

Die Detektionsrate beträgt folglich $\frac{7}{45} = 15,55\%$, die Fehlerrate der Detektion $\frac{0}{45+0} = 0\%$ und die Erfolgsrate der Klassifikation $\frac{5}{7} = 71,42\%$. Kombiniert ergibt das eine Erfolgsrate von:

$$15,55\% * 71,42\% = 11,11\%$$

Die Verarbeitung des Videos wurde in 15:16 Minuten abgeschlossen, was einer relativen Verarbeitungsdauer von 243% entspricht.

Beim Test von hochauflösten Videos unter besseren Wetter- und Licht-Bedingungen wurden deutlich bessere Detektionsraten und eine damit verbundene höhere Erfolgsrate erzielt. Auch die Verwendung der in Abschnitt 6.1 gezeigten Optimierungsmöglichkeiten führt zu besseren Ergebnissen.

5.7 Schwierigkeiten bei der Umsetzung

In diesem Abschnitt sollen die Probleme und Grenzen der Implementierung aufgezeigt werden. Primär ist uns aufgefallen, dass die Einzelbilder der getesteten Videos verschwommen sind.



Abb. 24: Aufnahme eines Verkehrsschilds aus verschiedenen Entfernungen

Bei der Aufnahme einer Fahrt aus einer Dashcam-Perspektive bewegen sich die Schilder, wie in Abb. 24 gezeigt, von der Bildmitte zum Bildrand, wobei die Größe des Schilds im Bild zunimmt. Bei gängigen Smartphone-Kameras sind die Einzelbilder bei hohen Geschwindigkeiten vor allem in den sich schnell wechselnden Randbereichen unscharf und perspektivisch verzerrt. Schilder in der Ferne sind weniger verschwommen, allerdings können diese aufgrund der geringen Pixelgröße schlecht gelesen werden. Außerdem können diese, beispielsweise durch Regentropfen oder kleine Flecken auf der Windschutzscheibe ganz oder in Teilen überdeckt werden.

Auch eine irreführende oder fehlerhafte Beschilderung in der Straßenverkehrsordnung⁵² (StVO) führt zu Problemen. Die StVO verfolgt das Ziel, den Straßenverkehr in Deutschland und die dort verwendeten Beschilderungen möglichst eindeutig zu regulieren. Während dieses System in der Theorie keine offensichtlichen Widersprüche aufweist, sind Verkehrsschilder in der Praxis oft irreführend oder widersprüchlich aufgestellt, und es ist eine Portion „gesunder Menschenverstand“ und ein vorausschauendes Fahren erforderlich, um die Intention hinter den aufgestellten Schildern zu verstehen.

Beispiele für fehlerhaft aufgestellte Beschilderungen finden sich vor allem an Baustellen.

⁵² (Bundesregierung, 2017)

5.8 Veröffentlichung und Lizenz

Die Studienarbeit inklusive der praktischen Implementierung und dem erhobenen Trainings-Datensatz darf unter den Bedingungen der *GNU GPL v3* weitergegeben und modifiziert werden. Die vollständigen Lizenzbedingungen können unter der folgenden Adresse und in der Datei `LICENSE.txt` im Quelltext nachgelesen werden.

Link 5: GNU GPL v.3 Lizenz <https://www.gnu.org/licenses/gpl>

Die Lizenz erlaubt die private und kommerzielle Nutzung, Weitergabe und Modifikation der Software und die Nutzung dieser in Patenten unter den folgenden Bedingungen: Die Modifikation darf nur unter derselben Lizenz veröffentlicht werden und muss einen Hinweis auf die Autoren der ursprünglichen Software sowie eine Übersicht über die Änderungen enthalten. Es wird explizit keine Haftung oder Garantie übernommen.

Verkehrszeichenerkennung – Studienarbeit

© Copyright 2018 Alexander Melde, Stefan Schneider

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Abb. 25: Lizenz der Studienarbeit

Die praktische Implementierung dieser Studienarbeit verwendet Teile von Open Source Software. Die für diese geltenden Lizenzen sind in der Datei `NOTICE.txt` im Quelltext aufgeführt.

5.9 Download, Bedienung und Installation

Die Studienarbeit inklusive der praktischen Implementierung wurde veröffentlicht und kann von der Internetplattform GitHub heruntergeladen werden:

Link 6: Quelltext-Download <https://github.com/AlexanderMelde/Verkehrszeichenerkennung>

Es gibt zwei Möglichkeiten zur Installation. Entweder kann das Projekt anhand des Python Sourcecodes selber kompiliert werden oder als lauffähiges Programm heruntergeladen werden. Wird letztere Methode gewählt, muss im komplett heruntergeladenen build-Ordner lediglich der für das Betriebssystem passende Ordner geöffnet und die darin liegende ausführbare Datei `start_vze` mit einem Doppelklick gestartet werden.

Die bereits kompilierte Version kann unter dem folgenden Link heruntergeladen werden:

Link 7: Build-Download <http://wwwlehre.dhbw-stuttgart.de/~it15111/vze/>

5.9.1 Einrichten der Entwicklungsumgebung

Da es sich bei dem Quelltext um Skripte der Skriptsprache Python 3.6.3 handelt, kann das Programm auf verschiedenen Betriebssystemen gestartet werden.

Für Start und Entwicklung sind neben einer Python3-Installation die folgenden Module sowie deren Abhängigkeiten („Dependencies“) erforderlich:

- `apjar` (0.82.1)
- `cx-Freeze` (5.1.1)
- `numpy` (1.14.2)
- `opencv-contrib-python` (3.4.0.12)
- `opencv-python` (3.3.1)
- `pip` (9.0.2)
- `setuptools` (38.5.1)
- `tensorflow` (1.4.0)
- `virtualenv` (15.1.0)
- `wheel` (0.30.0)

Die Verwendung des Moduls `virtualenv` ist optional, wird aber zur besseren Trennung mehrerer Python-Projekte auf einem System empfohlen. Eine Einführung in Virtual Environments ist kein Teil dieser Arbeit, eine gute Anleitung zur Einrichtung findet man unter folgendem Link:

Link 8: `virtualenv`-Tutorial <https://www.geeksforgeeks.org/python-virtual-environment/>

Wurde die `virtualenv` eingerichtet können alle weiteren Module innerhalb der Environment (Umgebung) installiert und das Skript innerhalb der Umgebung gestartet werden.

Das Modul `cx-Freeze` wird lediglich zum Erstellen von ausführbaren Dateien benötigt, eine Installation ist also nur notwendig, wenn der Python Code später ohne Interpreter ausgeführt werden soll.

Für die Darstellung der GUI wird in der nicht-kompilierten Version zudem das GUI-Framework Tk benötigt. Dieses wird bei den meisten Python-Installationen bereits als Tkinter mitgeliefert. Sollte Tk noch nicht vorhanden sein, kann die folgende Anleitung genutzt werden:

Link 9: Tk-Framework Installationsanleitung <http://www.tkdocks.com/tutorial/install.html>

5.9.2 Ausführen

Vor der ersten Ausführung des Programms zur Verkehrszeichenerkennung muss, nachdem die Entwicklungsumgebung wie im vorherigen Abschnitt beschrieben, eingerichtet wurde, das neuronale Netz für die Klassifikation trainiert werden. Dies geschieht mit dem auf Seite 42 in Abb. 20 beschriebenen Befehl.

Nach dem Trainieren des neuronalen Netzes kann die Verkehrszeichenerkennung mit dem folgenden Befehl (innerhalb der virtualenv) gestartet werden:

```
> python main.py
```

Abb. 26: Aufruf der Verkehrszeichenerkennung

In der Konsole erscheint nach dem Aufruf der Hinweis, dass die grafische Oberfläche nun geladen wird.

Die Bedienung der grafischen Oberfläche wird als selbsterklärend angesehen. Kleine Buttons und Hilfetexte unterstützen den Anwender innerhalb der grafischen Oberfläche.

5.9.3 Kompilieren

Um das entwickelte Programm auch ohne die Einrichtung einer Entwicklungsumgebung starten zu können, kann eine ausführbare Datei kompiliert werden.

Hierfür wird das Modul cx-Freeze verwendet, das die mitgelieferte Konfigurationsdatei `setup.py` beim Aufruf einliest und aus den Skripten eine passende Datei kompiliert.

Das Build-Skript kann mit dem folgendem Befehl aufgerufen werden:

```
> python setup.py build
```

Abb. 27: Kompilieren der Verkehrszeichenerkennung

Hierbei wird ein neuer Ordner `build` erstellt, der die ausführbare Datei `start_vze` enthält.

6 Erweiterungsmöglichkeiten

Im Rahmen dieser Arbeit wurden die geforderten Basisanforderungen erreicht. Darüber hinaus wurden bereits zahlreiche Optimierungen vorgenommen, wie beispielsweise die Objektverfolgung von Schildern über mehrere Einzelbilder hinweg oder alternative Eingabemöglichkeiten (Webcam) und Konfigurationsmöglichkeiten über die GUI.

Dennoch wurden bei weitem nicht alle möglichen Funktionalitäten in das Programm integriert. Während der Entwicklung wurden Ideen für zahlreiche Erweiterungsmöglichkeiten erarbeitet.

Neben neuen Features gibt es auch Möglichkeiten, den bisherigen Programmcode performanter zu gestalten.

6.1 Optimierungsmöglichkeiten

Gegen Ende der Arbeit stellte sich heraus, dass das von uns entwickelte Programm nicht mit kommerziellen Produkten der Autoindustrie mithalten kann. Dies gilt sowohl für die Laufzeit als auch für die Erkennungsrate der Schilder. Deshalb werden in diesem Abschnitt mögliche Optimierungen aufgezählt, welche die Erkennungsrate verbessern und die Geschwindigkeit erhöhen.

6.1.1 Parallelisierung und Einsatz von Grafikkarten

Ein möglicher Punkt, um die Verarbeitungsgeschwindigkeit zu erhöhen, wäre die Verwendung von leistungstärkerer Hardware. Dies verbessert nicht das Programm, kann jedoch zu einem besseren Nutzer-Erlebnis führen. So testen wir die Erkennung auf verschiedenen Prozessoren (CPUs), mit verschiedenen Taktraten und Anzahl von Kernen.

Hochleistungs-CPUs konnten hierbei bis zu 10 Bilder pro Sekunde mehr erreichen als Mittelklasse CPUs. Es stellte sich auch heraus, dass primär nicht die Anzahl der Kerne oder Threads die Bearbeitung beschleunigte, sondern lediglich die Taktrate und die Singlecore-Performance des Prozessors.

Trotz der nativen Multicore Unterstützung der OpenCV Bibliothek wird meist nur ein Kern voll ausgelastet und zusätzliche Kerne laufen zwischen 25 und 50 Prozent. Um eine bessere Auslastung der CPU zu erreichen gibt es mehrere Möglichkeiten.

- Der komplette Algorithmus könnte zerteilt und in eigenen Threads ausgeführt werden. Dies wurde in kleinem Umfang schon bei der Erkennung von runden und eckigen Schildern gemacht. Da diese Unabhängig voneinander laufen werden sie parallel ausgeführt und ihr Ergebnis wird in einer Liste zusammengeführt. Es konnte jedoch nur ein minimaler Performance Zuwachs gemessen werden.
- Das eingelesene Bild könnte auf die Anzahl der Kerne aufgeteilt werden. Hierzu müsste die Anzahl der Kerne ausgelesen werden und das Bild in entsprechend viele Quadrate unterteilt werden. Jeder CPU Kern untersucht daraufhin nur einen Ausschnitt des Bildes. Hierbei ist jedoch zu beachten, dass Ausschnitte sich überlappen müssen, da sonst Verkehrszeichen in Randbereichen nicht immer erkannt werden.
- Jedem CPU Kern/Thread könnte parallel je ein Einzelbild des Videos zugeteilt werden. Dies könnte die CPU maximal auslasten, kann aber nicht bei der Auswertung von Echtzeitdaten eingesetzt werden. Außerdem wird zusätzliche Logik benötigt, die garantiert, dass ausgewertete Daten in der richtigen Reihenfolge zurückgegeben werden und Funktionen wie der Bildübergreifende Tracker weiterhin funktionieren.

Auch das Zusammenspiel von Soft- und Hardware kann optimiert werden. Werden Grafikkarten (GPU) und zugehörige Software-Frameworks und Treiber verwendet, so können mehrere Operationen parallel auf großen Datenmengen ausgeführt werden. Vor allem in der Bildverarbeitung werden GPUs deshalb gern zur Beschleunigung von Abläufen verwendet.

Wenn eine Operation an die GPU übergeben wird, so wird diese parallel auf mehreren Pixeln des Bildes ausgeführt. Das ist um einen Faktor schneller als die serielle Abarbeitung aller Pixel auf der CPU.

Mit dem Einbinden einer Grafikkarte und den entsprechenden Befehlen im Code könnte sich die Laufzeit der Erkennung verbessern, denn Operationen wie die Farbfilterung, die Konvertierung

zwischen Farbräumen, das Berechnen der Kanten und das Verwaschen des Bildes könnten auf einer GPU ausgeführt werden.⁵³

Auch der Klassifikationsalgorithmus kann sich mit einer GPU beschleunigen lassen, da ein Großteil der Algorithmen für maschinelles lernen für GPUs entwickelt wurden und TensorFlow Grafikartenframeworks wie NVIDIA CUDA unterstützt. Hiermit sind laut NVIDIA bis zu 50% schnellere Berechnungen möglich.⁵⁴

In einem ersten Versuch wurde CUDA sowie die GPU-kompatible Version von TensorFlow installiert. Außer einem Anstieg des belegten Grafikartenspeichers wurde aber keine erhöhte GPU-Auslastung oder Performance festgestellt, hier bedarf es scheinbar noch einer weiteren Konfiguration des Frameworks.

6.1.2 Verbessern der Kameraqualität

Für eine gute Detektionsrate sind gute Bilder notwendig. Hierzu wird eine leistungsstarke Kamera benötigt, die hochauflösende und scharfe Bilder mit guten Kontrasten generiert. Damit die Rechenzeit bei der hohen Auflösung nicht ansteigt, könnten diese Bilder für Teile des Algorithmus auf niedrigere Auflösungen skaliert werden. Dies könnte die Rechenzeit deutlich verringern.

6.1.3 Berücksichtigung der Häufigkeit von Verkehrszeichen

Die Häufigkeitsverteilung der einzelnen Verkehrszeichen in der realen Welt ist nicht gleichmäßig. Es gibt beispielsweise mehr aufgestellte auf 50km/h begrenzende Schilder als solche für 120km/h.⁵⁵

Wenn zwei Kennzeichen-Klassen also eine ähnliche per TensorFlow berechnete Wahrscheinlichkeit haben (nur wenige Prozentpunkte zwischen zwei Zeichen), dann sollte bei Berücksichtigung der Häufigkeit das auf (deutschen) Straßen öfters vorkommende Schild als Ergebnis angezeigt werden.

6.1.4 Erhöhen der Trainingsdaten

Allgemein lässt sich die Klassifikation auch durch das Erhöhen der Trainingsdaten verbessern. Eine Methode, die zu einer höheren Anzahl an Bildern führt ist beispielsweise das Spiegeln oder Drehen von Bildern mit richtungsunabhängigen beziehungsweise drehbaren Zeichen.⁵⁶

6.1.5 Einsatz von kontinuierlichem Lernen

Um die Erfolgsrate der Klassifikation weiter zu steigern, könnte der Algorithmus auch während der Auswertung von Videos dazulernen. Eine Methode, die die Anzahl der Trainingsdaten kontinuierlich erhöht, ist das Senden von Nutzerbildern an eine zentrale Plattform, die die Trainingsdaten für das neuronale Netz bereitstellt. Hierfür wäre neben einem ausreichend dimensionierten Server auch eine Datenschutzerklärung und das Einverständnis der Nutzer erforderlich.

Hochwertigere Ergebnisse werden erreicht, wenn der Nutzer oder der zentrale Plattformbetreiber die Bilder vor dem Weiterverarbeiten überprüft, um die Korrektheit der erkannten Beschriftungen zu bestätigen. Zeitintensiver aber noch besser wäre es, auch die richtige „Lösung“ anzugeben. Zur Motivation der Nutzer können unter anderem Gamification-Elemente und Belohnungssysteme eingesetzt werden.

⁵³ (Wilms, 2013), (Hangün & Eyecioğlu, 2017)

⁵⁴ (NVIDIA Corporation, 2018)

⁵⁵ (Zanetti, 2016, S. 31)

⁵⁶ (Staravoiu, 2017)

6.1.6 Beachten der Kameraposition

Als eine Anforderung an diese Arbeit wurde definiert, dass das Bildmaterial nicht immer von einem fahrenden Auto stammt. Beachtet man diese Anforderung nicht, so ist eine Optimierung für den Einsatz in Fahrzeugen möglich.

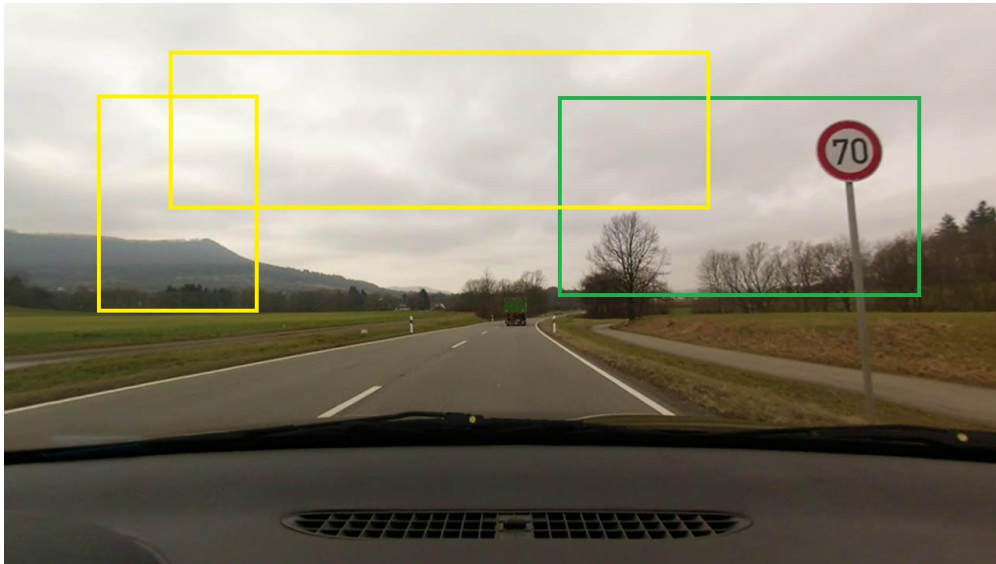


Abb. 28: Beispiel für einen Filter mit relevanten Regionen

Wäre das Bildmaterial immer von einer fest im Fahrzeug verbauten Kamera aufgenommen, könnte ein akkuraterer Farbfilter oder eine bessere Filterung anhand der Schildergröße realisiert werden. Mit dem Wissen über die Einbauhöhe und den Erfassungswinkel der Kamera, sowie dem Wissen über Sichtbarkeit und Aufstellung, aus Kapitel 3.1.4, könnten relevante Abschnitte definiert werden. Diese Regionen wären die Bereiche in welchen die Schilder vorkommen.

In vielen Fällen könnte die komplette untere Hälfte des Bildes ignoriert werden, da dort oft nur die Straße dargestellt ist. Von der oberen Hälfte wäre besonders der rechte Abschnitt interessant (grünes Rechteck in Abb. 28), da sich dort die meisten Schilder befinden. Mittig oben und in der linken oberen Hälfte (gelbe Rechtecke in Abb. 28) kann mit wenig Priorität gesucht werden, da dort seltener Schilder vorkommen.

Dieses Verfahren könnte an der Fläche gemessen einen theoretischen Performance-Gewinn von 50% – 70% mit sich bringen.

6.1.7 Einsatz eines leichtgewichtigeren Frameworks

Da die Klassifikation von einzelnen Bildern im Vergleich zur Erkennung viel Zeit in Anspruch nimmt, besteht auch hier Optimierungspotential im Code. Eine Möglichkeit ist, das umfangreiche TensorFlow-Framework durch eine leichtgewichtige Alternative zu ersetzen, wie beispielsweise TensorFlow Lite⁵⁷ oder Torch⁵⁸.

⁵⁷ (The TensorFlow Authors, 2018)

⁵⁸ (What is Torch?, 2018)

6.1.8 Einsatz eines leichtgewichtigeren Klassifikations-Modells

Modelle für neuronale Netze werden heutzutage nur noch selten von Grund auf selber trainiert. Oft werden bereits trainierte Modelle verwendet, die schon ein grundlegendes Verständnis von alltäglichen Objekten bieten und dann auf den jeweiligen Anwendungsfall angepasst werden.

In der aktuellen Implementierung wurde ein sehr präzises, aber auch sehr langsames und speicherintensives Modell verwendet, da wir die Präzision als wichtiger als die Klassifikationsdauer angesehen haben.

Ein Wechsel zu einem Ressourcenschonenderem Framework sollte zu deutlich schnelleren Bearbeitungszeiten und einer nur geringfügig schlechteren Klassifikation führen. Für den Einsatz in Realtime-Systemen empfehlen wir daher den Wechsel des Klassifikations-Modells.

Ein Wechsel des Klassifikations-Modells ist mithilfe des Parameters `--architecture` im Skript `classification/retrain.py` möglich.⁵⁹

6.1.9 Unterscheidung mehrerer Klassifikationsmodelle

Um die Klassifizierung zu entlasten, könnten verschiedene Modelle trainiert werden, die jeweils nur eine Auswahl von Verkehrszeichen erkennen. Je nach Anzahl der Kanten wäre es beispielsweise möglich, ein passendes Modell nur für runde, rechteckige oder dreieckige Schilder zu verwenden. Hierzu müsste jedoch die Ecken-Erkennung noch besser werden, da es hier während des Tests zu falschen Einschätzungen kam. Für eine weitere Unterteilung bietet es sich an, ein weiteres Programm in Open CV zu erstellen, welches nicht nur die grobe Form, sondern auch die Farbe des Verkehrszeichens an die Klassifikation übergibt.

Wenn dieses Programm gute Ergebnisse erzielt, ist es auch möglich, die Klassifikation in OpenCV durchzuführen. Die Erkennung könnte stufenweise gestaffelt werden, und nur Verkehrszeichen, die von OpenCV nicht erkannt werden an den aufwändigeren Klassifikator übergeben.

6.1.10 Detektion mittels Machine Learning

Eine weitere Methode der Optimierung ist das Verwenden von Machine Learning Frameworks, die in einem Bild mehrere Objekte und deren Positionen erkennen können, wie beispielsweise YOLOv2.⁶⁰

Die Schwierigkeit dieser „Single Shot MultiBox Detectors“ (SSD) besteht dabei primär im Beschriften des Trainingsmaterials. Im Gegensatz zu den hier verwendeten Beispielbildern müssten die Bilder für diese Methode nicht zugeschnitten sein, da der Kontext des gesamten Bildes bei der Klassifikation miteinbezogen wird. Zu jedem Bild muss daher zusätzlich eine Datei angelegt werden, die die Position der relevanten Objekte im Bild beinhaltet.⁶¹

Die Verwendung solcher Klassifikatoren macht eine separate Erkennung überflüssig, was die Performance steigern könnte.

⁵⁹ (The TensorFlow Authors, 2018)

⁶⁰ (Zhang, Huang, Jin, & Li, 2017)

⁶¹ (Liu, et al., 2015, S. 4-5)

6.2 Funktionserweiterungen

6.2.1 Gültigkeit von Schildern

Der entwickelte Algorithmus zur Verkehrszeichenerkennung ist für autonomes Fahren nicht ausreichend, da zahlreiche Regelungen durch Fahrbahnmarkierungen (beispielsweise ein Überholverbot) und Straßenverläufe (wie Einmündungen und Kreuzungen) sowie Lichtzeichen (Ampeln) geregelt sind, und eine Erkennung dieser Situationen im Rahmen dieser Arbeit nicht implementiert wurde.

Eine weitere Möglichkeit zur Erweiterung der Funktionalität des Programms ist die Interpretation der Bedeutung von Schildern, was beispielsweise besonders beim autonomen Fahren relevant ist. In einem ersten Schritt könnten Schilder die Zonen einleiten und beenden („Tempo 30“, Parkverbote, ...) analysiert werden, damit dem Autofahrer zu jedem Zeitpunkt die aktuell gültigen Verkehrsregeln angezeigt werden können.

Hierfür ist neben einer sehr hohen Erkennungsrate aber noch ein weiteres Feature notwendig, das bisher noch nicht implementiert wurde: Die Erkennung von Ein- und Ausfahrten sowie Fahrspuren.

Laut StVO werden einige Verkehrsregelungen wie beispielsweise Tempolimits an Einmündungen auf die jeweils gültigen Standardregelungen zurückgesetzt.⁶²

Selbst bei einer zuverlässigen Erkennung all dieser Situationen ist aber Vorsicht geboten: Wie bereits in Abschnitt 5.7 beschrieben, ist die Beschilderung nicht immer eindeutig, sodass eine Interpretation der Gesamtsituation erforderlich ist.

Ein Beispiel für irreführende Regelungen findet sich an Einmündungen, an denen Geschwindigkeitsbegrenzungen erst einige Meter nach der Auffahrt wiederholt werden, obwohl bisherige Geschwindigkeitsbeschränkungen an der Einmündung aufgehoben werden.

Wer sich hier strikt an die StVO hält, könnte auf dem kurzen Stück auf die, für den jeweiligen Straßentyp maximal zugelassene Geschwindigkeit beschleunigen, auch wenn das in den meisten Fällen hinsichtlich der hiermit entstandenen Gefahren und dem höheren Kraftstoff-Verbrauch nur selten effizient ist.

6.2.2 Neue Verkehrszeichen für autonomes Fahren

Neben den in der StVO beschriebenen Schildern gibt es zahlreiche weitere nichtamtliche Zeichen, die ebenfalls erkannt werden könnten. Ein Beispiel sind die für autonomes Fahren eingeführten Verkehrsschilder, die genaue Positionsangaben enthalten.⁶³

Die Implementierung dieser Schilder ist, ausreichend Beispielbilder vorausgesetzt, ohne weiteres möglich, in dem im Ordner `signs` sowie in der Übersetzungstabelle in `main.py` ein neuer Zeichentyp ergänzt und das Modell des neuronalen Netzes neu trainiert wird.

⁶² (Bundesregierung, 2017)

⁶³ (Bundesministerium für Verkehr und digitale Infrastruktur, 2016)

6.2.3 Kennzeichenerkennung

Neben den Verkehrszeichen kommen in den getesteten Videos oft auch Autokennzeichen (Nummernschilder) vor. Es wurden daher bereits erste Schritte unternommen, ein Framework zur Erkennung von Kennzeichen zu integrieren. Aufgrund der einfachen Bedienung, der zuverlässigen länderübergreifenden Erkennung und aufgrund des offenen Quellcodes wurde die *Automatic License Plate Recognition library OpenALPR* ausgewählt.

Im Folgenden werden Installation und Test von OpenALPR als selbstständige Version beschrieben, eine Integration in das entwickelte Programm ist noch nicht erfolgt. OpenALPR bietet zur Installation einen fertigen Docker-Container an, der nach dem Herunterladen und Kompilieren dessen direkt gestartet werden kann.

Falls noch nicht geschehen muss zur Installation zunächst das Virtualisierungstool Docker installiert werden. In dieser Arbeit wurde die Community Edition anhand der folgenden Anleitung installiert:

Link 10: Docker Installation: <https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/>

Download und Installation des Docker-Containers erfolgen anschließend über den folgenden Befehl:⁶⁴

```
> docker build -t openalpr https://github.com/openalpr/openalpr.git
```

Abb. 29: Installation des OpenALPR Docker-Containers

Für eine grundlegende Erkennung von Nummernschildern kann das mitgelieferte Kommandozeilenprogramm genutzt werden. Der Aufruf erfolgt direkt über Docker, sodass kein vorheriger Wechsel in die Docker-Umgebung notwendig ist. Das zu analysierende Bild, in diesem Fall `testbild.jpg` liegt in dem aktuell in der Kommandozeile genutzten Ordner, der über den Parameter `-v` im Docker-Container verfügbar gemacht wird.

Die erste Pfadverknüpfung dient der Unterdrückung einer Warnmeldung bezüglich einer veralteten Bibliothek, die die Auswertung der im JSON-Format zurückgegebenen Bildanalyse erschwert. Wird der Wert des Parameters `-n` erhöht, steigt die Anzahl der zurückgegebenen, möglicherweise im Bild vorhandenen Kennzeichen. In unserem Fall wird nur das Kennzeichen, das mit der höchsten Wahrscheinlichkeit im Bild ist zurückgegeben.

```
> docker run -it --rm -v /dev/null:/dev/raw1394 -v $(pwd):/data:ro openalpr \
  -j -n 1 -c eu testbild.jpg
```

Abb. 30: Aufruf des OpenALPR Kommandozeilenprogramms

Eine beispielhafte Ausgabe des Programms befindet sich in Anhang 1. Neben dem Kennzeichen in Textform wird auch die Position des erkannten Kennzeichens zurückgegeben, was beispielsweise das Einzeichnen eines Rahmens um das Kennzeichen in OpenCV ermöglicht.

⁶⁴ (OpenALPR Technology, Inc., 2017)

6.2.4 Anwendung direkt im Fahrzeug

Der in Python geschriebene Anwendungscode wurde plattformübergreifend unter Windows und Linux getestet. Die Kernkomponenten sollten auch auf anderen Plattformen wie beispielsweise Android funktionieren, lediglich eine Änderung an der grafischen Benutzeroberfläche wäre erforderlich.

Auf einem Tablet mit Windows-Betriebssystem sollte die Anwendung ohne weitere Anpassungen funktionieren. Wenn als Eingangssignal die rückseitige Webcam ausgewählt wird kann eine Live-Erkennung gestartet werden.

Bei einem solchen Einsatz als Embedded-System sind noch weitere Dinge zu beachten, wie die Art der Stromquelle (eigener Akku oder Anschluss an die Autobatterie), die Dimensionierung der Hardware und die Art der Ergebnisdarstellung (zum Beispiel via Head-up-Display oder Sprachausgabe).

Auch eine Spracheingabe zur Konfiguration des Programms wäre sinnvoll, da je nach Bedienung die Zulassung im Fahrzeug nicht erteilt wird.

In Hinblick auf das neu erlassene Urteil zur Verwertbarkeit von Dashcam-Aufnahmen, das ausdrücklich anlassbezogene Aufnahmen im Fahrzeug zulässt,⁶⁵ wäre auch eine anlassbezogene Speicherung eine mögliche Erweiterung. Durch die Kopplung mit Sensoren im Fahrzeug oder eine Auswertung des Kamerabilds zur Erkennung von Gefahren beziehungsweise Anlässen zur Speicherung könnte eine dauerhafte Aufnahme und Speicherung verhindert werden.

⁶⁵ (Pressestelle des Bundesgerichtshofs, 2018)

7 Vergleich

In diesem Kapitel wird die in dieser Arbeit entwickelte Implementierung mit den in der Marktanalyse in Abschnitt 2.1 genannten bereits veröffentlichten Systemen verglichen.

Die Erkennungsrate und Bearbeitungszeit unserer Implementierung war deutlich schlechter als bei vergleichbaren Forschungsprojekten. Mit den im Kapitel 6.1 gezeigten Optimierungsmöglichkeiten lässt sich dieser Wert aber noch verbessern.

Die Anzahl der unterscheidbaren Verkehrszeichen ist in unserer Implementierung ähnlich groß wie in den meisten bisherigen Forschungsprojekten. Zählt man die Klassen mit weniger als 30 Trainingsbildern dazu, so werden in unserer Implementierung deutlich mehr Klassen unterschieden.

Für unsere Implementierung wurden rund 16.000 Trainingsbilder verwendet. Die meisten dieser Bilder wurden manuell aufgenommen und entstammen keinem bereits verfügbaren Datensatz. Der Aufbau unseres Datensatzes war daher vergleichsweise aufwendiger als das reine Verwenden eines bereits verfügbaren Datensatzes.

Datensätze wie der GTSRB bieten rund 50.000 Bilder, wobei viele davon dasselbe Verkehrsschild aus nur leicht unterschiedlichen Winkeln oder Entfernungen beinhalten. Unser Datensatz enthält bisher nur deutlich unterschiedliche Bilder in zudem höherer Auflösung als beim GTSRB. Durch Anwendung von Techniken der digitalen Bildbearbeitung (Spiegelung, Rotation, Skalierung) lässt sich auch unser Datensatz auf die Größe des GTSRB bringen.⁶⁶

Die bei der Erkennung aufgetretenen Probleme waren sehr ähnlich wie in den zu Beginn untersuchten Systemen. Die meisten Probleme traten ebenfalls durch die verschiedenen Lichtverhältnisse, Farbkalibrationen und teilweise Verdeckungen auf. Auch die durch hohe Geschwindigkeiten entstehende Unschärfe und geringe Pixel-Größe von weit entfernten Verkehrszeichen führen bei allen Implementierungen zu Schwierigkeiten bei Erkennung und Klassifikation.

Dass die verglichenen Systeme oft bessere Ergebnisse, als die in dieser Arbeit implementierte Verkehrszeichenerkennung bieten, war zu erwarten, da in deren Entwicklung trotz all unserer Bemühungen vermutlich mehr Ressourcen investiert wurden.

⁶⁶ (Staravoiu, 2017)

8 Fazit

Die in der Vorbetrachtung durchgeführte Marktanalyse hat dabei geholfen, das Projektumfeld kennenzulernen und die Anforderungen an das Projekt festzulegen.

In den theoretischen Grundlagen wurden einige Standards im Bereich der Verkehrszeichen und die Themen digitale Bildverarbeitung und künstliche Intelligenz einführend beschrieben. Ein Verständnis dieser Themen ist für viele aktuelle Themen der Software-Branche relevant.

Die Projektplanung hat sich im Nachhinein als sehr sinnvoll erwiesen, da es dank ihr bei der zeitlichen Planung und Umsetzung des praktischen Teils zu wenig Überraschungen kam.

Durch die praktische Implementierung wurde gezeigt, dass es auch mit relativ einfachen Mitteln möglich ist, eine rudimentäre Verkehrszeichenerkennung zu implementieren. Das verwendete Framework TensorFlow kann bereits nach einer sehr kurzen Zeit zur Bildklassifikation genutzt werden.

Die Schwierigkeit bei der Umsetzung besteht nun aber darin, auch gute Ergebnisse und eine hohe Erkennungsrate bei möglichst geringen Berechnungszeiten zu erhalten. Hier ist sowohl Fachwissen im Bereich der digitalen Bildverarbeitung als auch ein großer Satz an geeigneten Trainingsdaten erforderlich. Die Datenerhebung gestaltete sich als umfangreicher als ursprünglich geplant, da die Vielfalt der Verkehrsschilder in Deutschland sowie der Arbeitsaufwand, um diese mit Beispielbildern abzudecken, unterschätzt wurde.

Im Rahmen dieser unserer Arbeit wurde keine besonders hohe Erkennungsrate oder kurze Erkennungszeit erreicht. Durch weitere Optimierungen (siehe Abschnitt 6.1) sollten aber noch bessere Zeiten und Erkennungsraten erreicht werden können. Ein Vergleich mit den über mehrere Jahre hinweg entwickelten kommerziellen Systemen und akademischen Projekten ist daher nur bedingt möglich.

Mit den bisherigen Projekten (siehe Marktanalyse in Abschnitt 2.1) hat diese Implementation gemein, dass Fehler aufgrund sehr ähnlichen Ursachen entstehen. Die meisten Probleme traten auch in dieser Implementierung bei der Erkennung von Schildern im echten Straßenverkehr auf, die teilweise verdeckt sind, aufgrund der Geschwindigkeiten verschwommen sind, oder zu weit entfernt sind.

Ein wirklicher Mehrwert gegenüber anderen Systemen konnte aus technischer Sicht nicht erreicht werden. Aus Benutzersicht ist diese, gut dokumentierte und einfach zu installierende Implementierung aber durchaus interessant, da sie mit einem einsteigerfreundlichen GUI ausgestattet ist, die es auch nicht-Programmierern erlaubt, Videos mit Verkehrszeichen mit an ihrem Computer zu analysieren.

Die Forderungen hinsichtlich der Systemsicherheit und Verfügbarkeit wurden eingehalten. Auch die Integrierbarkeit ist dank der Skriptsprache Python gegeben. Die Ziele hinsichtlich Interaktivität und Usability wurden durch das übersichtliche und einfach zu bedienende GUI erreicht. Während der Entwicklung wurde die Skalierbarkeit der Software mit Videos verschiedener Längen getestet (Kleinstes: Einzelbild, Längstes: 3 Stunden). Die Erweiterbarkeit ist durch die Modularität des Programms und die einsteigerfreundliche Skriptsprache Python gegeben.

Die Forschung im Bereich der Verkehrszeichenerkennung ist eine nicht zu unterschätzende, für das autonome Fahren unerlässliche Voraussetzung. Die größten Herausforderungen bei der Implementierung einer sicheren und zuverlässigen Verkehrszeichenerkennung sind nach wie vor die Erkennungsrate und Bearbeitungsgeschwindigkeit.

Literaturverzeichnis

- ADAC e.V. (27. März 2018). *Studie Intelligent Speed Adaptation*. Abgerufen am 23. Mai 2018 von ADAC:
https://www.adac.de/infotestrat/tests/assistentensysteme/intelligente_geschwindigkeitsassistenten_2018/default.aspx
- Audi AG. (11. September 17). *Audi Media Center*. Abgerufen am 16. Mai 2018 von Der neue Audi A8 – hochautomatisiertes Fahren auf Level 3: <https://www.audi-mediacenter.com/de/per-autopilot-richtung-zukunft-die-audi-vision-vom-autonomen-fahren-9305/der-neue-audi-a8-hochautomatisiertes-fahren-auf-level-3-9307>
- Auto Zeitung. (13. Oktober 2017). *Tesla-Autopilot im Vergleichstest*. Abgerufen am 16. Mai 2018 von Auto Zeitung: <https://www.autozeitung.de/autopilot-tesla-model-s-mercedes-e-klasse-audi-q7-bmw-7er-134282.html>
- autohaus24. (Oktober 2013). *Was Verkehrszeichenerkennung bedeutet*. Abgerufen am 23. Mai 2018 von autohaus24.de: <https://www.autohaus24.de/ratgeber/verkehrszeichenerkennung>
- Bahlmann, C., Zhu, Y., Ramesh, V., Pellkofer, M., & Koehler, T. (12. September 2005). *A system for traffic sign detection, tracking, and recognition using color, shape, and motion information*. Abgerufen am 23. Mai 2018 von IEEE Xplore:
<https://ieeexplore.ieee.org/abstract/document/1505111/>
- Barthl, B. (6. Mai 2013). *PDF – Verkehrszeichen nach StVO*. Abgerufen am 21. Oktober 2017 von Verkehrszeichen Online: <http://www.verkehrszeichen-online.org/verkehrsschilder/>
- Bernau, P. (12. 12 2017). *Elon Musk kommt zu spät*. Abgerufen am 16. Mai 2018 von Frankfurter Allgemeine: <http://www.faz.net/aktuell/wirtschaft/diginomics/elon-musk-nimmt-prognose-fuer-autonome-autos-zurueck-15337359.html>
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. New York, New York, USA: Springer Science+Business Media, LLC. Abgerufen am 29. Mai 2018 von <http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf>
- Bruder, R. (16. Mai 2018). *Reuter, Tu Clausthal*. Abgerufen am 16. Mai 2018 von Tu Clausthal: http://www2.in.tu-clausthal.de/~reuter/ausarbeitung/Ralf_Bruder_-_Digitale_Bildverarbeitung.pdf
- Bundesministerium für Verkehr und digitale Infrastruktur. (13. Dezember 2016). *Teststrecke auf der A9 erhält neue Schilder für das automatisierte Fahren*. Abgerufen am 25. Mai 2018 von BMVI: <https://www.bmvi.de/SharedDocs/DE/Pressemitteilungen/2016/199-dobrindt-neue-schilder-dta.html>
- Bundesregierung. (22. Mai 2017). *Allgemeine Verwaltungsvorschrift zur Straßenverkehrs-Ordnung (VwV-StVO)*, BAnz AT 29.05.2017 B8. Abgerufen am 21. Oktober 2017 von Verwaltungsvorschriften im Internet: http://www.verwaltungsvorschriften-im-internet.de/bsvwvbund_26012001_S3236420014.htm

- Buric, C. (2. Dezember 2010). *Verkehrszeichenerkennung im Vergleich: Mit Navi und Kamera auf Schilderjagd - ADAC fordert weitere Verbesserung der Technik*. Abgerufen am 23. Mai 2018 von Presseportal: <https://www.presseportal.de/pm/7849/1727847>
- de la Escalera, A., Armingol, J., & Mata, M. (2002). *Traffic sign recognition and analysis for intelligent vehicles*. Abgerufen am 22. Mai 2018 von Pennsylvania State University: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.542.4195&rep=rep1&type=pdf>
- Die Vertragsparteien. (8. November 1968). *Übereinkommen über Strassenverkehrszeichen*. Abgerufen am 24. Februar 2018 von Das Portal der Schweizer Regierung: <https://www.admin.ch/opc/de/classified-compilation/19680245/index.html>
- DIN EN ISO 7010. (24. Februar 2018). *Farbtabelle*. Abgerufen am 24. Februar 2018 von DIN EN ISO 7010: <http://www.iso7010.de/farbtabelle/>
- dpa, Reuters. (20. März 2018). *Frau stirbt nach Unfall mit selbstfahrendem Auto von Uber*. Abgerufen am 16. Mai 2018 von Der Tagesspiegel: <https://www.tagesspiegel.de/weltspiegel/uber-in-den-usa-frau-stirbt-nach-unfall-mit-selbstfahrendem-auto-von-uber/21089290.html>
- Ford Motor Company. (10. Februar 2017). *Ford Invests In Argo AI, A New Artificial Intelligence Company, In Drive For Autonomous Vehicle Leadership*. Abgerufen am 22. Mai 2018 von Ford Media Center: <https://media.ford.com/content/fordmedia/fna/us/en/news/2017/02/10/ford-invests-in-argo-ai-new-artificial-intelligence-company.html>
- Gallinge, B. (6. August 2015). *MyDriveAssist im Test*. Abgerufen am 23. Mai 2018 von connect: <https://www.connect.de/testbericht/mydriveassist-test-fahrassistentz-app-android-ios-3185291.html>
- Gehrig, S. (17. 2 2018). *Unterlagen zur Vorlesung Digitale Bildverarbeitung*. Abgerufen am 25. Mai 2018 von Stefan Gehrig's Homepage: <http://www.lehre.dhbw-stuttgart.de/~sghegrig/>
- Hangün, B., & Eyecioğlu, Ö. (15. Mai 2017). *Performance Comparison Between OpenCV Built in CPU and GPU Functions on Image Processing Operations*. Abgerufen am 16. Mai 2018 von DergiPark: <http://dergipark.gov.tr/download/article-file/320019>
- Heilck, J. (2016). *Unterlagen zur Vorlesung Projektmanagement. Projektmanagement 2016*. Stuttgart, Baden-Württemberg, Deutschland: DHBW Stuttgart.
- Kehl, M., Enzweiler, M., Froehlich, B., Franke, U., & Heiden, W. (2. November 2015). *Vision-Based Road Sign Detection*. Abgerufen am 23. Mai 2018 von IEEE Xplore: <https://ieeexplore.ieee.org/document/7313181/>
- King, D. E. (22. Januar 2018). *Dlib C++ Library*. Abgerufen am 7. Februar 2018 von Dlib C++ Library: <http://dlib.net/>
- Korsch, U. (22. Juni 2017). *VzKat 2017 (amtliche Version)*. Abgerufen am 21. Februar 2018 von VzKat: <http://www.vzkat.de/2017/VzKat.htm>

- Liang, M., & Hu, X. (2015). Recurrent Convolutional Neural Network for Object Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (S. 3367-3375). Abgerufen am 30. Mai 2018 von Computer Vision Foundation open access: http://openaccess.thecvf.com/content_cvpr_2015/papers/Liang_Recurrent_Convolutional_Neural_2015_CVPR_paper.pdf
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (8. Dezember 2015). *SSD: Single Shot MultiBox Detector*. Abgerufen am 28. Mai 2018 von arXiv.org: <https://arxiv.org/abs/1512.02325>
- Melde, A. (2018). *Containerisierung von verschiedenen auf Machine Learning basierenden Video- und Bildanalyse-Programmen mithilfe von Docker*. Projektarbeit, Duale Hochschule Baden-Württemberg Stuttgart, Informatik, Karlsruhe.
- Mortsiefer, H. (1. Juli 2016). *Tödlicher Unfall eines Tesla ist ein Rückschlag für Autoindustrie*. Abgerufen am 22. Mai 2018 von Der Tagesspiegel: <https://www.tagesspiegel.de/wirtschaft/autonome-fahrzeuge-toedlicher-unfall-eines-tesla-ist-ein-rueckschlag-fuer-autoindustrie/13815792.html>
- Noé, B. (Januar 2018). *Spracherkennung, Voice Apps und Sprechererkennung*. Abgerufen am 26. Mai 2018 von Unterlagen zur Vorlesung Digitale Sprachverarbeitung: <http://www.lehre.dhbw-stuttgart.de/~noe/speechproc.html>
- NVIDIA Corporation. (25. Mai 2018). *GPU-Accelerated TensorFlow*. Abgerufen am 25. Mai 2018 von NVIDIA Data Center: <https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/tensorflow/>
- OpenALPR Technology, Inc. (13. Mai 2017). *openalpr: Automatic License Plate Recognition Library*. Abgerufen am 4. November 2017 von GitHub: <https://github.com/openalpr/openalpr>
- OpenCV team. (7. Februar 2018). *About*. Abgerufen am 7. Februar 2018 von OpenCV: <https://opencv.org/about.html>
- Pressestelle des Bundesgerichtshofs. (15. Mai 2018). *Verwertbarkeit von Dashcam-Aufnahmen als Beweismittel im Unfallhaftpflichtprozess*. Abgerufen am 26. Mai 2018 von Der Bundesgerichtshof: http://juris.bundesgerichtshof.de/cgi-bin/rechtsprechung/document.py?Gericht=bgh&Art=pm&pm_nummer=0088/18
- PwC - PricewaterhouseCoopers IL. (2017). *Sizing the prize: PwC's Global Artificial Intelligence Study*. Abgerufen am 22. Mai 2018 von PwC Global: <https://www.pwc.com/gx/en/issues/analytics/assets/pwc-ai-analysis-sizing-the-prize-report.pdf>
- Rashid, T. (2017). *Neuronale Netze selbst programmieren*. (F. Langenau, Übers.) Heidelberg, Baden-Württemberg, Deutschland: O'Reilly.
- Raval, S. (9. September 2016). *Build a TensorFlow Image Classifier in 5 Min*. Abgerufen am 21. Oktober 2017 von YouTube: <https://www.youtube.com/watch?v=QfNvhPx5Px8>

- Raval, S. (10. September 2016). *tensorflow_image_classifier/src/label_image.py*. Abgerufen am 21. Oktober 2017 von GitHub:
https://github.com/llSourcell/tensorflow_image_classifier/blob/master/src/label_image.py
- Sablatnig, R., Zambanini, S., & Licandro, R. (2014). Einführung in Computer Vision. In *Einführung in Visual Computing* (S. 5 - 8). Wien: TU Wien. Abgerufen am 7. Februar 2018 von
<https://www.cg.tuwien.ac.at/courses/EinfVisComp/Skriptum/SS14/EVC-02%20Einf%C3%BChrung%20in%20Computer%20Vision.pdf>
- Seitz, P., Lang, G. K., Gilliardt, B., & Pandazis, J. C. (1991). The Robust Recognition of Traffic Signs from a Moving Car. In B. Radig, *Mustererkennung 1991* (S. 287-294). Zürich: Paul Scherrer Institut Zürich. Abgerufen am 22. Februar 2018 von https://link.springer.com/chapter/10.1007/978-3-662-08896-8_37
- Sermanet, P., & LeCunn, Y. (5. August 2011). *IEEE Xplore*. Abgerufen am 25. Mai 2018 von Traffic sign recognition with multi-scale Convolutional Networks:
<https://ieeexplore.ieee.org/document/6033589/>
- Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2011). The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *Proceedings of the IEEE International Joint Conference on Neural Networks* (S. 1453 - 1460). San Jose, California, USA. Abgerufen am 21. Oktober 2017 von <http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset>
- Staravoitau, A. (15. Januar 2017). *Traffic signs classification with a convolutional network*. Abgerufen am 28. Mai 2018 von Alex Staravoitau's Blog: <https://navoshta.com/traffic-signs-classification/>
- Sung, J.-c. (18. Februar 2017). *SSD in TensorFlow: Traffic Sign Detection and Classification*. Abgerufen am 28. Mai 2018 von GitHub:
https://github.com/georgesung/ssd_tensorflow_traffic_sign_detection
- The TensorFlow Authors. (30. März 2018). *Introduction to TensorFlow Lite*. Abgerufen am 25. Mai 2018 von TensorFlow: <https://www.tensorflow.org/mobile/tflite/>
- The TensorFlow Authors. (15. Februar 2018). *retrain.py*. Abgerufen am 21. Februar 2018 von GitHub:
https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/image_retraining/retrain.py
- Tiedemann, M. (6. Oktober 2017). *Fünf Anwendungsfälle für Machine Learning in der Industrie 4.0*. Abgerufen am 7. Februar 2018 von Alexander Thamm Data Science Services:
<https://www.alexanderthamm.com/artikel/anwendungsfaelle-machine-learning-in-der-industrie-4-0/>
- What is Torch?* (25. Mai 2018). (R. Collobert, C. Farabet, K. Kavukcuoglu, & S. Chintala, Produzenten) Abgerufen am 25. Mai 2018 von Torch: <http://torch.ch/>
- Wikipedia-Autoren. (31. März 2018). *HSV-Farbraum*. Abgerufen am 31. März 2018 von Wikipedia, Die freie Enzyklopädie: <https://de.wikipedia.org/w/index.php?title=HSV-Farbraum&oldid=174297421>

- Wikipedia-Autoren. (24. Februar 2018). *Verkehrszeichen (Deutschland)*. Abgerufen am 24. Februar 2018 von Wikipedia, Die freie Enzyklopädie: Wikipedia-Autoren
- Wilkens, A. (17. Mai 2018). *Auffahrunfall mit Tesla im "Autopilot"-Modus – US-Behörde ermittelt*. Abgerufen am 26. Mai 2018 von heise online:
<https://www.heise.de/newsticker/meldung/Auffahrunfall-mit-Tesla-im-Autopilot-Modus-US-Behoerde-ermittelt-4050902.html>
- Williams, A. (23. März 2016). *Crosswalk photo by Adrian Williams (@nairdasemaj) on Unsplash*. Abgerufen am 20. Oktober 2017 von Unsplash:
<https://unsplash.com/photos/AU07BMLW1NA>
- Wilms, C. (12. Dezember 2013). *CUDA und Python*. Abgerufen am 16. Mai 2018 von Uni Hamburg:
https://kogs-www.informatik.uni-hamburg.de/~seppke/content/teaching/wise1314/20131212_wilms-pycuda.pdf
- Wolski, W. (2016). *PONS Kompaktwörterbuch : Deutsch als Fremdsprache*. Stuttgart: PONS. Abgerufen am 23. Mai 2018 von
<https://books.google.de/books?id=LjnICgAAQBAJ&pg=PA524&lpg=PA524>
- Zanetti, F. (2016). *Convolutional Networks for Traffic Sign Classification*. Master's Thesis, Chalmers University of Technology, Communication Engineering, Göteborg, Schweden. Abgerufen am 19. Februar 2018 von
<http://publications.lib.chalmers.se/records/fulltext/245747/245747.pdf>
- Zhang, J., Huang, M., Jin, X., & Li, X. (16. November 2017). A Real-Time Chinese Traffic Sign Detection Algorithm Based on Modified YOLOv2. *Algorithms*, 10(4:127). doi:10.3390/a10040127

Abbildungen ohne Quellenangabe wurden im Rahmen dieser Arbeit als Fotos, Screenshots, Tabellen oder mithilfe des Programms Struktogrammer von Hans-Ulrich Steck selbst erstellt.

Deckblatt: (Williams, 2016)

Zum Schreiben dieser Arbeit wurde Microsoft Office Word verwendet.
Die Literaturangaben entsprechen dem APA Sixth Edition Format.

Anhang

Anhangsverzeichnis

| | |
|---|----|
| Anhang 1: OpenALPR Kennzeichenerkennung Beispielrückgabe..... | 67 |
|---|----|

Anhang 1: OpenALPR Kennzeichenerkennung Beispielrückgabe

```
{
  "version"          : 2,
  "data_type"        : "alpr_results",
  "epoch_time"       : 1509800088303,
  "img_width"        : 640,
  "img_height"       : 360,
  "processing_time_ms" : 72.983803,
  "regions_of_interest" : [
    {"x":0,"y":0,"width":640,"height":360}
  ],
  "results":[{
    "plate"          : "MCM5493",
    "confidence"     : 93.474541,
    "matches_template" : 0,
    "plate_index"    : 0,
    "region"         : "",
    "region_confidence" : 0,
    "processing_time_ms" : 23.922096,
    "requested_topn"  : 1,
    "coordinates"    : [
      {"x":136,"y":227},
      {"x":249,"y":227},
      {"x":249,"y":255},
      {"x":136,"y":255}
    ],
    "candidates":[{
      "plate"          : "MCM5493",
      "confidence"     : 93.474541,
      "matches_template" : 0
    }]
  }]
}
```
