Schildererkennung und Hinderniserkennung

Ibrahim Al Krad, Oliver Greiner-Petter, Yannick Schössow

Ansatz Schilderkennung

Verkehrsschilder sind ein essenzieller Bestandteil des Straßenverkehrs, ohne die komplettes Chaos ausbrechen würde. Sie geben vor, wie an welchen Stellen gefahren werden soll und entfernen den Interpretationsspielraum für eine ordentliche Vehrkerssituation. Verkehrsschilder haben bewusst distinkte Formen und Farben, die sie von der Umgebung im Straßenverkehr abheben, um sie so sichtbar wie möglich für Verkehrsteilnehmer zu machen.

Diese Eigenschaft nutzen wir, um die Schilder mithilfe der Kamera und künstlicher Intelligenz zu erkennen und geben die Informationen über erkannte Schilder an den Decider weiter, um entsprechend zu reagieren. Bei der Umsetzung spielten Machine Learning, OpenCV, und dessen Cascade Classifiers wichtige Rollen. Im Folgenden erklären wir die erwähnten Technologien und ihre Einsatzgebiete.

Machine Learning

Machine Learning ist ein Teilbereich der Künstlichen Intelligenz, der darauf abzielt, neuronale Netze zu produzieren, welche in der Lage sind, aus Daten (z.B. Bilder) gewisse Muster zu erkennen. Vorteilhaft dabei ist, dass die Entwicklung des neuronalen Netzes ohne menschliche Programmierarbeit stattfindet. Diese Eigenschaft macht den Einsatz von KI und ML auch für Nicht-Informatiker sinnvoll. Es basiert auf Algorithmen, die Muster und Zusammenhänge in großen Datenmengen erkennen und nutzen, um Vorhersagen oder Entscheidungen zu treffen. Der Prozess des maschinellen Lernens umfasst in der Regel das Sammeln und Vorverarbeiten von Daten, das Trainieren eines Modells mit diesen Daten, die Evaluierung der Modellleistung und die anschließende Anwendung des Modells auf neue, unbekannte Daten. Typische Anwendungen sind Bilderkennung, Sprachverarbeitung und Empfehlungsdienste, die alle durch kontinuierliche Anpassung und Optimierung der Algorithmen verbessert werden.

Feature Extraction

Ein entscheidender Faktor im maschinellen Lernen, der sich auf die Auswahl und Transformation relevanter Merkmale aus Rohdaten konzentriert, ist das Feature Extraction. Diese Merkmale sind wichtig, da sie die Grundlage für das Training von Modellen bilden und deren Leistung erheblich beeinflussen können. Der Prozess umfasst das Identifizieren von Attributen, die für die Aufgabe relevant sind, und das Reduzieren von Datenredundanz, um die Verarbeitungseffizienz zu erhöhen. Anders als beim Deep Learning wird beim Machine Learning das Feature Extracting von einem Menschen gemacht. Das kann

Feature Extraction kann auf verschiedene Weise durchgeführt werden, abhängig von der Art der Daten. Bei numerischen Daten können statistische Merkmale wie Mittelwert, Standardabweichung oder Korrelationskoeffizienten verwendet werden. Bei Bilddaten werden häufig Techniken wie Kantenerkennung oder Farbhistogramme angewendet. Für Textdaten können Wortfrequenzen oder semantische Merkmale extrahiert werden. Ziel ist es, die Komplexität der Daten zu reduzieren und gleichzeitig die relevanten Informationen beizubehalten, um die Leistungsfähigkeit der nachfolgenden maschinellen Lernalgorithmen zu maximieren.

Vorgehen

Als Erstes werden Trainingsdaten gesammelt. Dafür wird ein Video aufgenommen, wo das Schild, das erkannt werden muss, aus verschiedenen Winkeln in verschiedenen Orten sichtbar ist und eins, wo das Schild nicht auftaucht. Aus den beiden Videos werden dann mithilfe von FFmpeg die Frames extrahiert. Die Frames des Videos mit dem zu erkennenden Schild werden dann manuell annotiert und die Schilder (bzw. Merkmale) markiert. Diese Klassifikation dient dazu, sogenannte 'Positive Examples' -also Bilder, wo das Schild auftaucht- als auch 'Negative Examples' zu klassifizieren. Mit diesen gesammelten Trainingsdaten können wir ein KI-Modell trainieren, welches dann das gewünschte Objekt in Bildern erkennt. Dabei ist es wichtig auf die Hyperparameter, wie Anzahl der Stages, Anzahl der positive und negative Examples etc. zu achten. Sobald ein KI-Modell erzeugt wurde, geht es in die Testphase über, wo ggf. die Hyperparameter angepasst werden müssen oder weitere Stages trainiert werden müssen. In Abbildung 1 ist das Konzept des maschinellen Lernens visuell dargestellt. In Abbildung 2 sieht man die Konsolenausgabe während des Trainings:

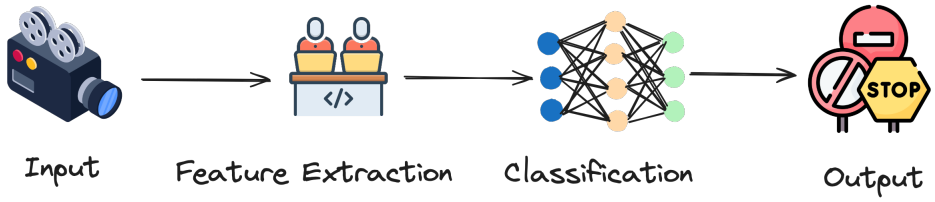

Abbildung 1: Konzept von Machine Learning

```
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|0.0923077|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 0 minutes 20 seconds.

===== TRAINING 5-stage =====
<BEGIN
POS count : consumed   130 : 130
NEG count : acceptanceRatio    65 : 5.28954e-05
Precalculation time: 1.223
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1| 0.123077|
+----+---------+---------+
|   3|        1|        0|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 0 minutes 33 seconds.

===== TRAINING 6-stage =====
<BEGIN
POS count : consumed   130 : 130
NEG current samples: 3
```

Abbildung 2: KI Modell trainieren

Wenn ein Schild erkannt wird, wird es im Kamerafeed entsprechend markiert, in der Debug-Konsole ausgegeben und an den Decider weitergeleitet, der darüber entscheidet, welches Verhalten als Reaktion ausgeführt werden muss.

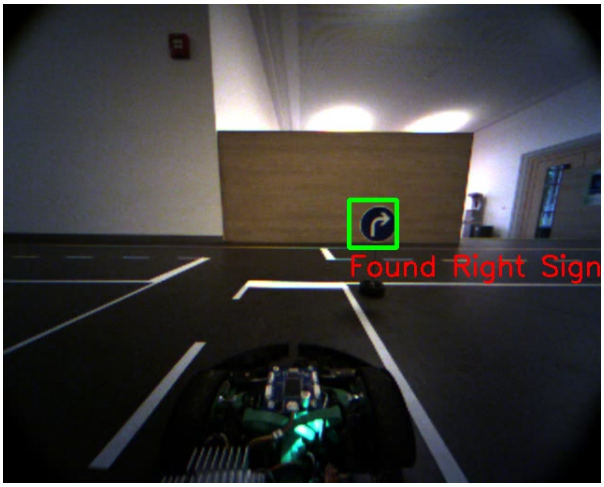

Abbildung 3: Machine Learning in Anwendung

Herausforderungen bei der Schilderkennung

Es war aufgrund von Speicherplatzproblemen nicht möglich, TensorFlow auf dem Auto zu installieren, unser erster Ansatz war somit nicht umsetzbar und wir mussten einen Neuen suchen. OpenCV Haar Cascades schienen hier vielversprechend, da diese schon auf dem Auto installiert waren. Haar Cascades sind heutzutage jedoch sehr unbeliebt und es gibt keine vorgefertigten Modelle für unseren Anwendungsfall, also mussten wir eigene Modelle trainieren.

Zunächst, hat das Training Schwierigkeiten mit sich gebracht, denn wir wussten nicht, welche Hyperparameter ideal sind und auch nicht, wie gute Input-Daten aussehen.

Unser erster Versuch war es mit dem GTSRB Dataset[1] zu trainieren. Jedoch hat dieses Vorgehen nicht in unserem Environment funktioniert, da das Kamerabild des Autos sich von dem des Trainingssets unterscheidet. Außerdem sind unsere Verkehrszeichen nicht standartisiert, wodurch es zu Problemen bei der Erkennung kommt; ein Beispiel hierfür ist das Parkschild, welches in Realität oft anders aussieht.

Letztendlich haben wir uns dazu entschieden die Trainingsdaten serlber zu erstellen, indem wir auf dem Auto den Video Recorder nutzen und die Ausgabe als Daten nutzen. Die Ausgabe wird von uns in einzelne Bilder per FFMpeg aufgeteilt und händisch über OpenCV CLI Tools annotiert. Nun ist das Problem, dass wir immernoch nicht wissen, welche Hyperparameter ideal sind (wieviele positiv bzw. negativ images, wieviel padding beim annotaten, wieviele stages, etc.). Hierfür haben wir per Trial-and-Error unsere Ergebnisse immmer weiter verbessert. Ideal für uns waren z.B. >350 positiv images und >200 negativ images und ca. 20 stages training, unter vielen weitern Parametern.

[1] https://benchmark.ini.rub.de/

Ansatz Hinderniserkennung

Zunächst erscheint die Idee, Hindernisse als Schilder zu erkennen am einfachsten und gleichzeitig am effizientesten, da der aktuelle Stand der Schilderkennung vielversprechend ist. Allerdings werden dann Hindernisse, welche sich nicht auf der Fahrbahn befinden, aber im Kamerafeld auftauchen, ebenfalls erkannt.

Da davon ausgegangen wird, dass die Hindernisse eine grüne Oberfläche haben, war das ideal für eine klassische Farberkennung. Aus dem gesamten Bild werden die Pixel aus der Mitte (zwischen den Spuren) analysiert. Beinhalten diese einen bestimmten vordefinierten Mindestwert von Grün, so wird von einem Hindernis auf der Spur ausgegangen.

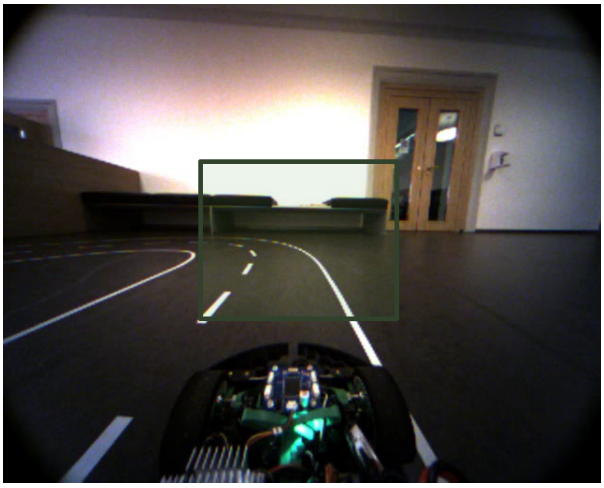

Abbildung 4: Boundary Box zum erkennen von Hindernissen

Herausforderungen bei der Hinderniserkennung

Probleme:

- Es erwies sich als schwierig, ideale Werte für den Ausschnitt zu finden, nicht zu klein oder zu groß.
- Außerdem war es schwierig einen passenden Threshold für die RGB-Werte zu finden.
- Diese Pobleme haben wir mittels Trial-and-Error gelöst.