Autonomes Fahren
Gruppe 2

Ibrahim Al Krad - 70481850

Mika Braunschweig - 70481898

Oliver Greiner-Petter - 70482614

Bennet Heidmann - 70476746

Artur Kück - 70476621

Philipp Miesner - 70476424

Calvin Mende - 70476423

Jakob Reichstein - 70476618

Roxana Reiß - 70476623

Yannick Schössow - 70482034

Inhaltsverzeichnis

Einleitung

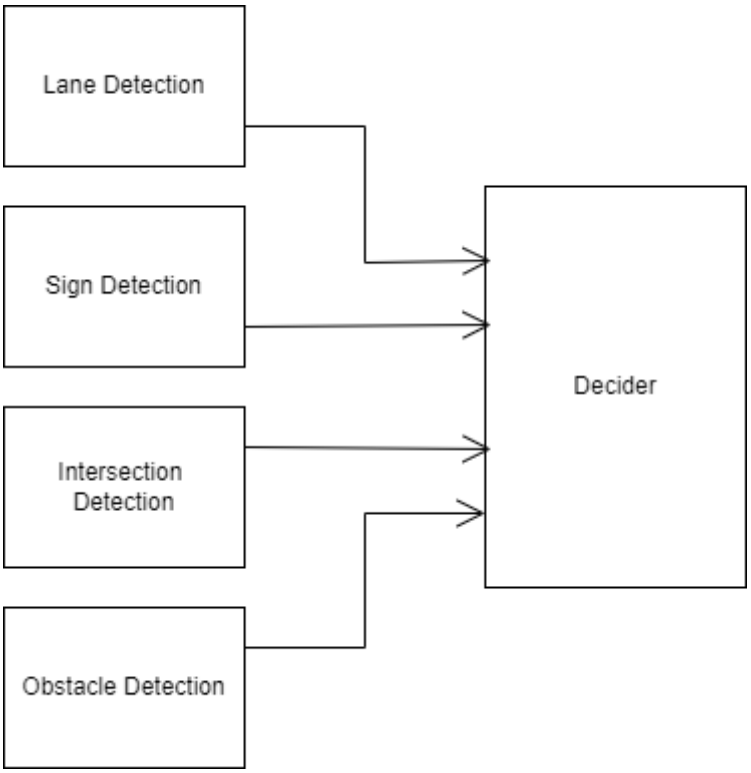

Abbildung 1: Systemarchitektur

Wir haben am Anfang eine Architektur für den Ablauf des autonomen Fahrens erstellt. Dabei werden die Daten der Spurerkennung, Schilderkennung, Kreuzungserkennung und der Hinderniserkennung an einen „Decider“ geschickt. Dieser entscheidet anhand der Daten, ob das Fahrzeug einer geraden Spur folgt, an einer Kreuzung steht oder ein Hindernis vor sich hat. Anhand dieser Entscheidung wird die nächsten Fahraufgaben ausgeführt. In einem „default-state“ folgt das Auto den Straßenmarkierungen. Kommt das Fahrzeug an eine Haltelinie, wird an dieser gehalten und geschaut, was für eine Kreuzung vorliegt. Dazu werden auch die Schilder an der Kreuzung einbezogen. Danach fährt das Fahrzeug durch die Kreuzung und landet wieder auf einer geraden Spur und folgt dieser. Befindet sich ein Hindernis vor dem Fahrzeug, hält dieses an und wartet, bis das Hindernis verschwunden ist.

Spurverfolgung

Beteiligte Personen:

- Roxana Reiß
- Calvin Mende
- Artur Kück

BEV

Als Erstes wurde in diesem Projekt eine „bird's eye view“ (BEV) programmiert. Dafür haben wir das Bild verzerrt und in Graustufen umformatiert. Es gab mehrere BEV-Ansätze, die unterschiedlich weit vor dem Auto ansetzen. Unsere finale Version beginnt an der Stoßstange des Autos und ist mit einem Schachbrett kalibriert, damit die später zu berechnenden Winkel und Radien realen Werten entsprechen. Die BEV zeigt ca. 2,4 x 2,4 Meter Fläche vor dem Auto an. Scharf ist das Bild jedoch nur in den ersten 70 cm, danach können Linien nur noch selten erkannt werden. Dies wird in der nachfolgenden Liniensuche berücksichtigt.

Abbildung 2: BEV zeigt großen Ausschnitt

Abbildung 3: BEV zeigt echte Winkel

Spurerkennung

Unser finaler Ansatz nutzt eine Linienerkennung, welche auf Kreisen basiert. Eine Linie wird erkannt, wenn man entlang eines Halbkreises vom Mittelpunkt der letzten erkannten Spur des vorherigen Halbkreises nach links und rechts sucht. Dabei wird ein bestimmter Betrachtungswinkel definiert, um einen Sprung in der Spurerkennung herauszufiltern. Beim innersten Kreis wird beim Mittelpunkt des Halbkreises nach rechts und links geschaut.

Nachdem ein Mittelpunkt der Spur erkannt wurde, geht die Suche auf dem nächsten Kreis identisch weiter. Die Bestimmung des Mittelpunkts der zu befahrende Spur erfolgt anhand der Entfernung von zwei als Fahrbahnmarkierung erkannten Linien voneinander. Ist die Entfernung zwischen 30 und 70 Pixel, so wird ein Punkt in der Mitte der beiden Punkte als Spurmitte erkannt. Ist die Entfernung zwischen den beiden Punkten über 70 Pixel, so wird angenommen, dass keine Mittelspur gesehen wurde, und die Spurmitte wird mit 3×rechte Koordinate+1×linke Koordinate geteilt durch 4 errechnet, damit wird der Mittelpunkt der rechten Spur anwiesiert.

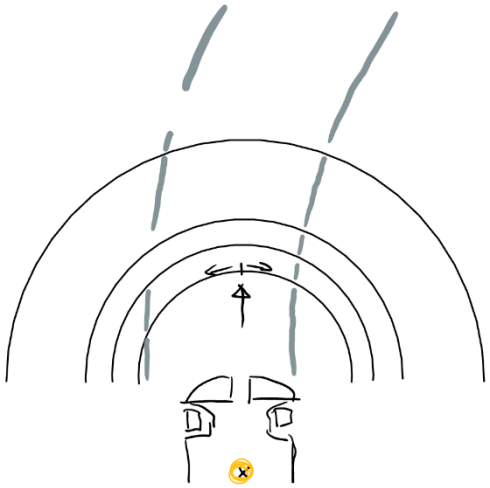

Abbildung 44: Spurerkennung

Eine Fahrbahnmarkierung wird erkannt, indem wir an unseren Halbkreisen den Farbverlauf der Pixel beobachten. Falls ein Wechsel von einem dunklen Bereich in einen hellen Bereich geschieht und innerhalb von 10 Pixeln wieder ein Austritt in einen dunklen Bereich erfolgt, wird der Mittelpunkt der Eintrittskoordinate und Austrittskoordinate entweder in die „right_point_list“ oder die „left_point_list“ des aktuellen Radius hinzugefügt. Anschließend kann die innerste legitime Koordinate genutzt werden, um die Spurmitte wie beschrieben zu ermitteln. Legitime Koordinaten sind Punkte, die keine direkten Punkte weiter im Inneren der Spur haben, aber dennoch die Entfernungsbedingungen erfüllen.

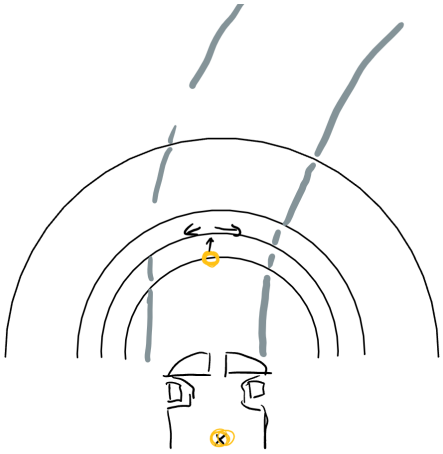

Abbildung 55: Spurerkennung

Mit dieser Methode werden für jeden Radius Punkte bestimmt, die die Fahrbahnmarkierungen kennzeichnen. Die Anordnung der Radien wurde erprobt. Für uns hat die anschließende Berechnung mit vielen kleinen Radien (50 bis 100 Pixel) und wenigen Radien, die weiter weg sind, am besten funktioniert. Es ist wichtig einige entfernte Punkte zu betrachten, damit vorallem gerade Linien mit wenig Schwanken innerhalb der Fahrbahn gefahren werden.

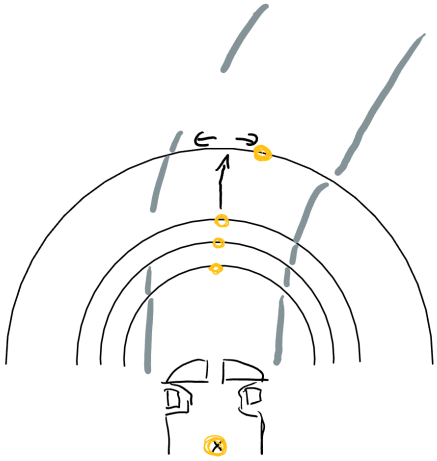

Abbildung 66: Spurerkennung

Nachdem wir nun alle Linienmarkierungen identifiziert haben, möchten wir einen Kreis durch unsere Punkte legen. Dafür vergleichen wir die durchschnittliche Entfernung aller Punkte zum aktuell zu erprobenden Fahrkreis und iterieren über 100 verschiedene Radien. Eine mögliche Verbesserung wäre, statt 100 verschiedener Radien jede Iteration nur 20 umliegende Radien (10 größer und 10 kleiner) als den zuletzt erkannten Radius zu betrachten. Unsere Laufzeit ist jedoch noch ausreichend gut, weshalb wir diese Laufzeitverbesserung nicht umgesetzt haben.

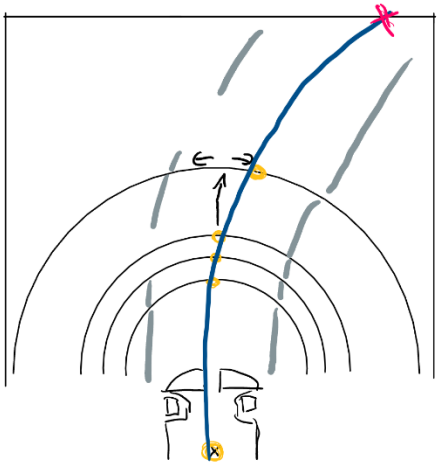

Abbildung 77: Spurerkennung - Lenkradiuserkennung

In den folgenden Bildern können Linienerkennungen unserer Bildverarbeitung gesehen werden. Wir haben unsere BEV schwarz-weiß konvertiert, jedoch ohne weitere Filter wie z.B. einem „Canny“-Algorithmus oder ähnlichen Verfahren.

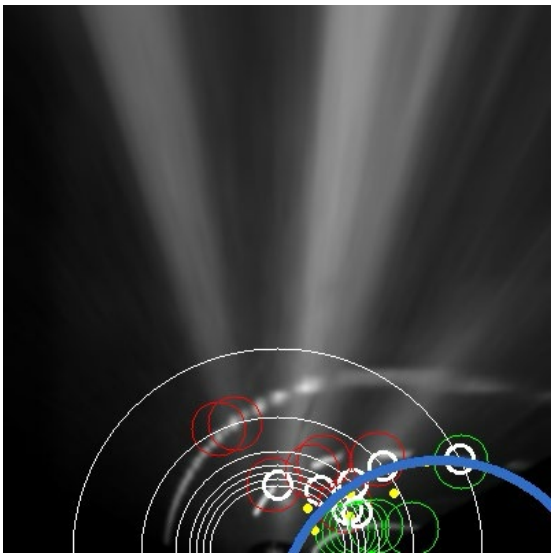

Abbildung 88: Bild der Linienerkennung, Rechtskurve

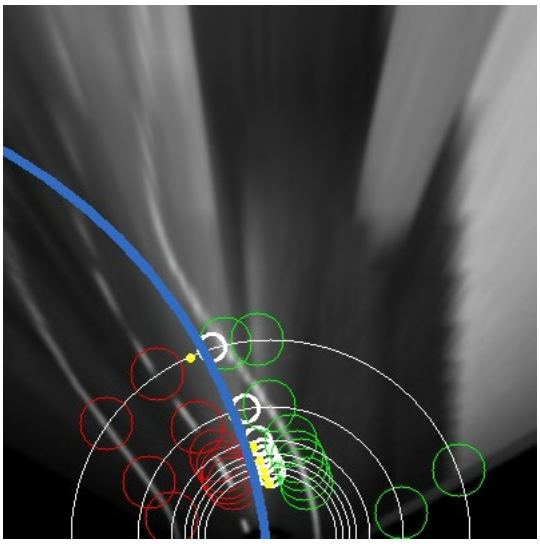

Abbildung 99: Bild der Linienerkennung. leichte Linkskrümmung

Die grünen Kreise sind alle Punkte, die der rechten Spur zugeordnet werden. Die roten Kreise zeigen die Punkte an, die der linken Spur zugeordnet wurden. Die weißen Kreise zeigen an, wo auf dem aktuellen Betrachtungshalbkreis die Suche nach Punkten begonnen hat, und der gelbe Punkt zeigt, wo letztendlich der anzufahrende Mittelpunkt der Spur erkannt wurde. Der dicke blaue Kreis zeigt die gewünschte Fahrbahn.

Wir haben besonders viele Halbkreise mit kleinem Radius, da dort die BEV die besten Bilder liefert und unser Erkennungsalgorithmus dort am besten funktioniert. Die entfernteren Halbkreise geben der Regression zusätzliche Sicherheit.

Nachdem der zu fahrender Radius bekannt ist, haben wir zunächst einen Ansatz versucht, bei dem wir mit dem Sinus der Höhe vom Mittelpunkt des Autos zu den Achsen und dem Radius den Lenkwinkel der Reifen definieren wollten. Dies hat jedoch nicht funktioniert, weshalb wir nun eine Methode nutzen, bei der wir die Entfernung des Schnittpunkts des Fahrkreises zum oberen mittleren Randpunkt im Bild berechnen und

abhängig davon lenken. Die Lenkung reagiert somit auf eine anzufahrende Position, da wir vorne und hinten denselben Lenkwinkel nutzen, wird die Ausrichtung des Autos damit auch berücksichtigt.

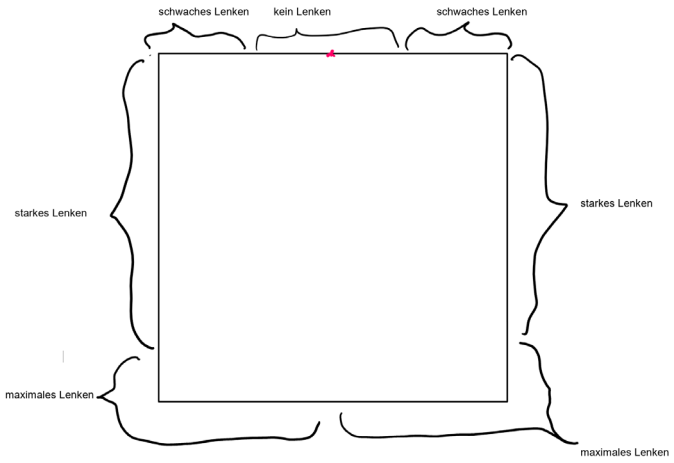

Abbildung 1010: Lenkverhalten abhängig vom Schnittpunkt des berechneten Lenkreises mit dem Rand

Die zu fahrende Geschwindigkeit ist abhängig vom gefahrenen Radius. Sie variiert zwischen 60 und 80 Geschwindigkeitseinheiten. Enge Kurven werden langsamer gefahren als gerade Strecken. Eine zu schnelle Geschwindigkeit ist aufgrund schwacher Lenkmotoren oder nicht ausreichend schneller Berechnung des Computers nicht empfehlenswert.

Umsetzung

Die Umsetzung der Birds-Eye-View (BEV) für die Spurenerkennung in diesem Code erfolgt durch eine Perspektivtransformation, die eine Vogelperspektive auf die Straßenbilder erzeugt. Zunächst werden die Quell- und Zielpunkte für die Transformation definiert und die entsprechende Transformationsmatrix mittels getPerspectiveTransform() berechnet. Während der Bildverarbeitung werden eingehende Kamerabilder in Graustufen umgewandelt und anschließend die Perspektivtransformation angewendet, um die BEV-Darstellung zu erhalten. Diese Transformation sorgt dafür, dass die Straßenmarkierungen in einer standardisierten Ansicht erscheinen, was die Erkennung und Analyse der Fahrspurmarkierungen vereinfacht.

Die Helper-Klasse in diesem Code implementiert die Berechnung des Straßenmittelpunkts und des Kurvenradius' aus einer Vogelperspektivenansicht des Straßenbildes. Sie verwendet OpenCV für die Bildverarbeitung und analysiert die Straßenmarkierungen in mehreren Radien, um den besten Mittelpunkt der Straße zu bestimmen. Die Methode calculate_radius() durchläuft mehrere feste Radien, sammelt relevante Punkte auf der Straße und bestimmt den Mittelpunkt, indem sie diese Punkte mittels Distanzberechnungen auswertet. Die resultierenden Mittelpunkte und Radien werden in der BEV-Darstellung visualisiert. Die Methode get_street_middle_from_points()

trennt die Punkte auf der linken und rechten Straßenseite und berechnet den Straßenmittelpunkt, während die Methode check_for_valid_point() sicherstellt, dass die Punkte auf der Straße liegen. Schließlich wird der resultierende Kurvenradius in einen Lenkwinkel umgerechnet, um Fahrbahnänderungen zu bestimmen.

Der Code der zwei Klassen Circle und SquareApproach ist für die geometrische Analyse und Berechnung von Schnittpunkten und Winkeln, die im Kontext der Straßenbildverarbeitung verwendet werden können. Die Circle-Klasse bietet Methoden zur Berechnung der nächsten Schnittpunkte zwischen einem Kreis und einer Linie sowie zur Ermittlung des nächstgelegenen Schnittpunkts zum Mittelpunkt des Bildes. Die Methode FindLineCircleIntersections berechnet die Schnittpunkte, während die Methode ClosestIntersection den nächstgelegenen Punkt ausgibt. Die SquareApproach-Klasse berechnet den Winkel zwischen einem gegebenen Punkt und einem Kreisradius, der durch die Bildränder begrenzt ist. Die Methode calc_angle ermittelt den Winkel basierend auf der Position des Punktes relativ zum Bildrand und zur Bildmitte. Diese Klassen werden verwendet, um vom ausgehenden Lenkwinkel/Radius des Kreises einen Schnittpunkt mit dem Bildrand zu ermitteln und diesen in einen Winkel umzurechnen. Der Lenkwinkel wird abschließend ans Auto weitergegeben.

Vorherige Umsetzungen

Die Spurerkennung wurde in einer vorherigen Iteration mit gleichem Verfahren auf 3 bis 5 horizontalen Schnittebenen genutzt. Dieser Ansatz hatte uns schon nach einigen Wochen der Entwicklung erlaubt gerade Strecken gut zu befahren. Dabei hatten wir jedoch das Lenkverhalten lediglich von einer relativen Koordinate zur gesamten Bildbreite errechnet. Dieses Verfahren mit der Schnittpunkterkennung wird in unserem Vorgehen mit Halbkreisen immernoch verwendet, die Umrechnung in einen Lenkwinkel funktioniert anders.

Ein weiteres Vorgehen, war die Nutzung von Histogrammen, diese haben wir jedoch nicht mehr weitergenutzt. Die Idee wurde jedoch in der Kreuzungserkennung ähnlich umgesetzt und perfektioniert. Für unsere Spurerkennung war dieses Vorgehen leider auch nicht optimal, da Kurven in einem Histogramm nicht von diagonalen unterschieden werden können und so das Auto entweder zu spät auf Kurven lenkte oder auf geraden ein starkes Pendeln verzeichnete.

Bewertung

Die Linienerkennung ist mit diesem Ansatz im Vergleich zu allen anderen Ansätzen am akkuratesten. Einige Features, die wir in anderen Ständen hatten, konnten wir in dieser Version leider nicht umsetzen, zum Beispiel eine Erkennung, auf welcher Spur sich das Auto befindet oder ob sich das Auto überhaupt noch auf einer Spur befindet. In einer früheren Iteration war die Differenzierung der States „on street“ und „off street“ sehr hilfreich, da das Auto regelmäßig von der Straße abgekommen ist. In unserer aktuellen

Version wurde sich gegen die Implementierung dieses Features entschieden, da das Fahrzeug bei guten Lichtbedingungen und ohne Reflektionen im Boden die Straße zu über 95% der Zeit nicht verlässt.

Das Fahrzeug kann Kurven sehr gleichmäßig fahren und bricht dort nicht aus. Auf geraden Strecken pendelt das Fahrzeug noch leicht. Wir haben dafür versucht, eine Methode zum leicht parallelen Fahren innerhalb einer Spur wiederzuverwenden. Diese hat in einer früheren Iteration abhängig vom Winkel der Straße zum Auto bewertet, ob parallel gelenkt werden kann oder ob eine Kurve gelenkt werden muss. Dies lässt sich leider in dieser Form nicht bei unseren Kreisregressionen implementieren. Man müsste dafür bewerten, ob die Spur eine Gerade ist und ob diese mit Parallellenkung noch befahrbar ist. Das Problem bei unseren Kreisregressionen ist dabei, dass eine diagonale, gerade Spur als Kurve mit kleinerem Radius erkannt wird. Das Zentrum unseres Fahrkreises liegt nämlich immer auf einer Höhe, orthogonal zum Auto. Eine Verfeinerung wäre es, wenn man für jeden möglichen Lenkkreis auf der orthogonalen y-Höhe noch weitere mögliche Lenkkreise ober- und unterhalb des Zentrums des aktuell betrachteten Kreises ausprobiert. Man würde damit Rechenzeit für gleichmäßigeres Fahren auf geraden Strecken eintauschen.

Insgesamt ist das Ergebnis zufriedenstellend. Unser Code ist durch die Auslagerung in insgesamt drei Klassen übersichtlich, wartbar und erweiterbar. Das Auto fährt die Spur mit diesem Code zufriedenstellend und bricht nur sehr selten aus der Spur aus. Meistens liegt es dann an einer schlechten Position beim Verlassen einer Kreuzung oder schlechten Lichtverhältnissen.

Verkehrsschilderkennung

Beteiligte Personen:

- Ibrahim Al Krad
- Oliver Greiner-Petter
- Yannick Schössow

Ansatz

Verkehrsschilder sind ein essenzieller Bestandteil des Straßenverkehrs, ohne welche komplettes Chaos ausbrechen würde. Sie geben vor, wie an welchen Stellen gefahren werden soll und entfernen den Interpretationsspielraum für eine ordentliche Verkehrssituation.

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 die 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.

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.

Umsetzung

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:

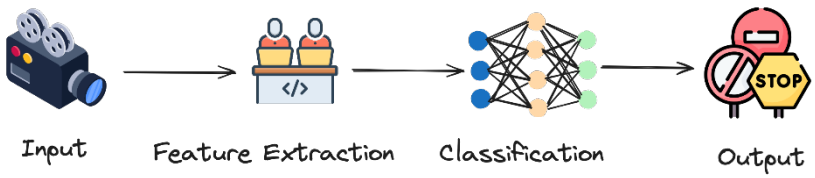

Abbildung 1111: Konzept von Machine Learning

```
|  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 1212: 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.

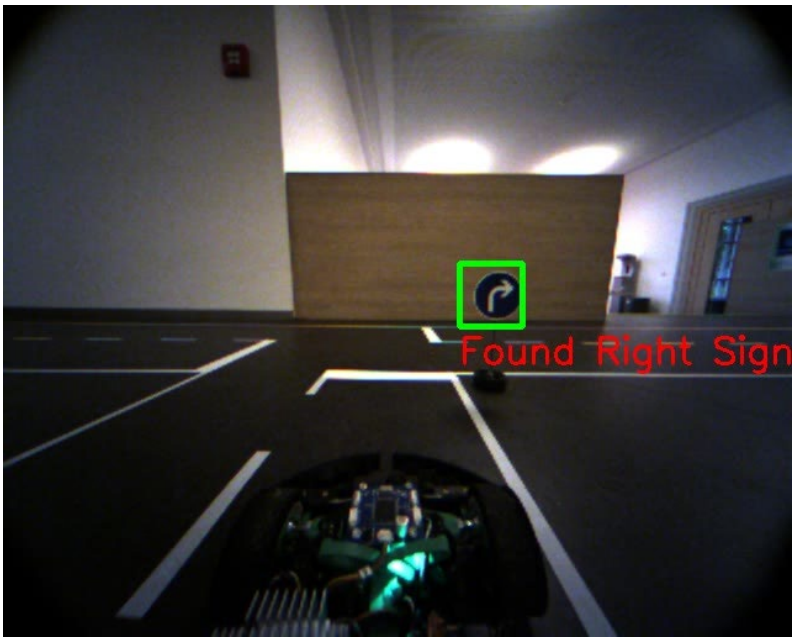

Abbildung 1313: Machine Learning in Anwendung

Probleme

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 standardisiert, 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 selber 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 immer noch nicht wissen, welche Hyperparameter ideal sind (wie viele positiv bzw. negativ images, wieviel padding beim annotaten, wie viele stages, etc.).

Hierfür haben wir per Trial-and-Error unsere Ergebnisse immer weiter verbessert. Ideal für uns waren z.B. >350 positiv images und >200 negativ images und ca. 20 stages training, unter vielen weiteren Parametern.

Bewertung

Insgesamt lässt sich sagen, dass das Ergebnis zufriedenstellend war. Wir erkennen alle geforderten Verkehrsschilder sehr gut und haben selten bis gar keine “false positives”. Diese filtern wir zusätzlich heraus, indem wir voraussetzen, dass ein Verkehrsschild mindestens 2 Frames durch OpenCV erkannt wurde, bevor wir es auf dem IPC veröffentlichen.

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

Hinderniserkennung

Beteiligte Personen:

- Ibrahim Al Krad
- Oliver Greiner-Petter
- Yannick Schössow

Ansatz

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.

Umsetzung

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.

Probleme

Zwischendurch gab es Probleme mit dem Erkennen von “false positives” in dunklen Bereichen. Dieses Problem haben wir behoben, indem wir einen Mindestwert für alle Farbkanäle hinzufügten.

Zusätzlich haben wir auch Probleme mit dem Sichtfeld gehabt. Wir haben dies zuerst etwas breit eingestellt, und somit zu oft ein Hindernis erkannt. Mittels Trial-and-Error haben wir jedoch eine gute Größe der Boundary Box zum Erkennen gefunden.

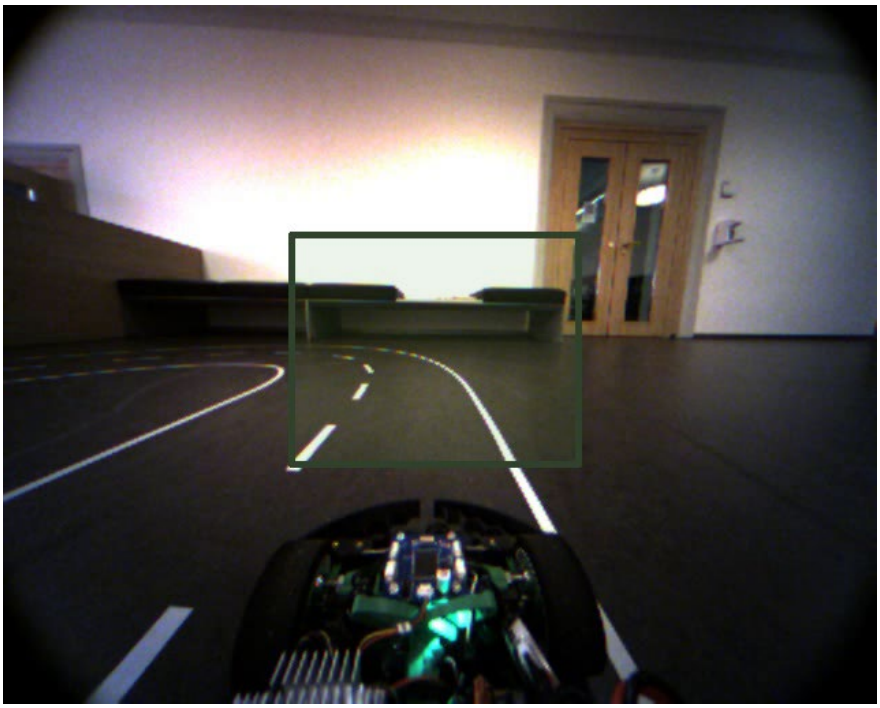

Abbildung 1414: False Positive - Boundary Box zum Erkennen von Hindernissen

Bewertung

Letztendlich funktioniert unsere Hinderniserkennung, gemessen an der Komplexität, sehr gut. Hindernisse neben der Spur oder auf der anderen Fahrbahnseite werden fast immer ignoriert. Ab und an kommt es zu “false positives”. Darunter fällt z.B., dass wir die Krümmung der Kurven nicht einberechnen und so Hindernisse am Straßenrand oder im Gegenverkehr missinterpretieren könnten.

Kreuzungserkennung

Beteiligte Personen:

- Mika Braunschweig

Ansatz

Für die Erkennung einer Haltelinie wird als erstes eine farblose Birds-Eye-View erstellt, die mit verschiedenen Effekten bearbeitet wird. Dann wird diese über mehrere Schritte ausgewertet, um Kreuzungen zu erkennen und dabei relativ robust gegen andere falsche positive Erkennungen zu sein.

Der Ansatz mit der Haltelinie wurde gewählt, da diese bei der Strecke immer vorhanden ist zu erwarten ist, dass eine Erkennung mit diesem Vorgehen ermöglicht wird.

Umsetzung

Birds-Eye-View

Die Konvertierung in die farblose Birds-Eye-View (folglich mit BEV abgekürzt) wurde von der Gruppe für die Spurerkennung erstellt. Hier wurde dann eine Transformation ausgewählt, die einen großen Teil der Strecke zeigt.

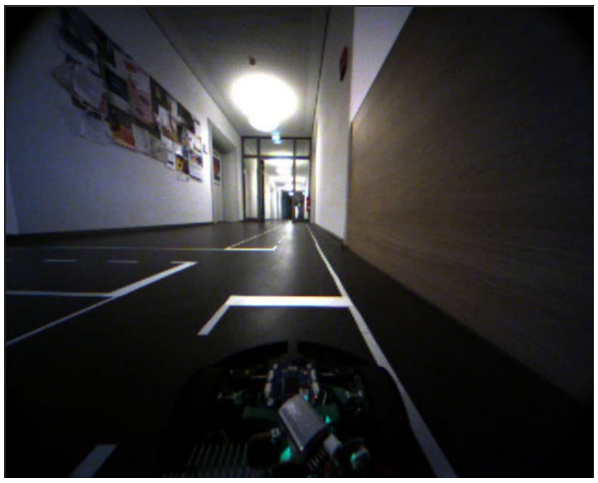

Abbildung 1515: Ausgangsbild

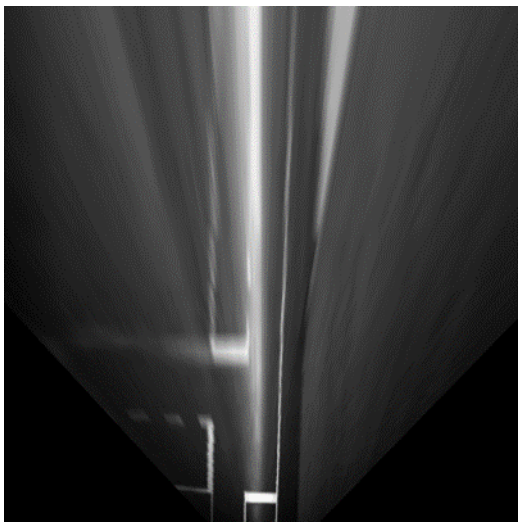

Abbildung 1616: Ausgangs-BEV

Die Bearbeitung des BEV-Bildes enthält die folgenden Schritte.

Als erstes wird das Bild mit gausschem Weichzeichnen (in OpenCV “GaussianBlur”) unschärfer gemacht. Der Grund hierfür ist, dass dadurch benachbarte Pixel keine zu harten Kanten haben.

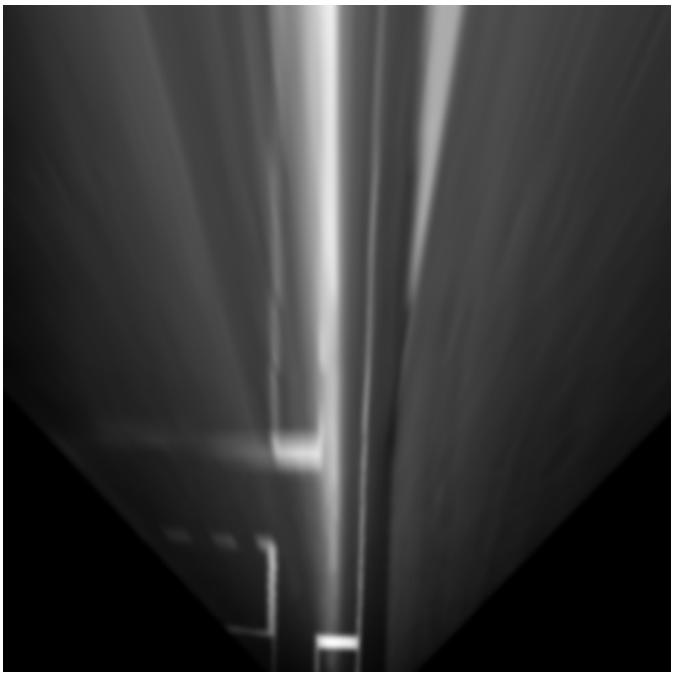

Abbildung 1717: BEV mit Weichzeichner

Als nächstes werden über den Canny Algorithmus die Kanten erkannt, damit die Linien später zu echten Werten konvertiert werden können:

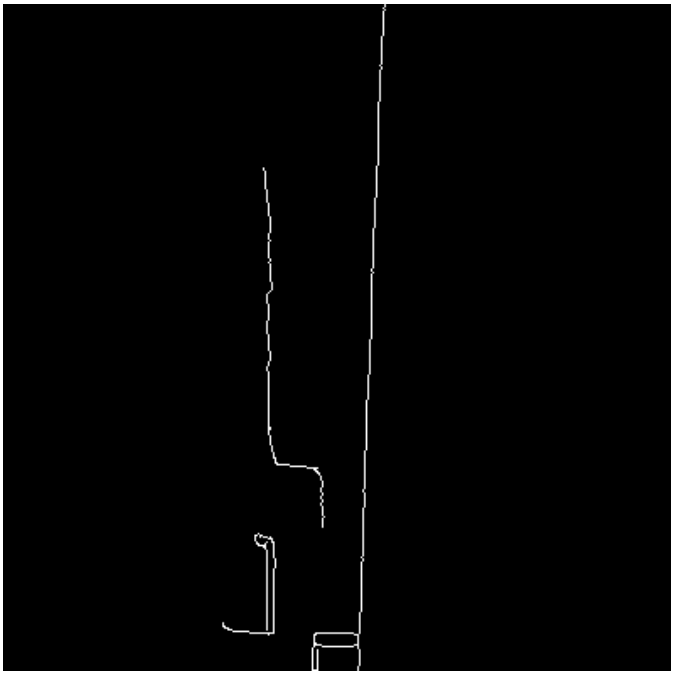

Abbildung 1818: BEV mit Canny

Zuletzt wird dieses Ergebnis nochmal über den GaussianBlur-Algorithmus unscharf gemacht. Der Grund hierfür ist, dass die Linien später noch über einen Algorithmus in echte Linien aus mehreren Punkten konvertiert werden und es hierdurch zu besseren Ergebnissen dabei kommt:

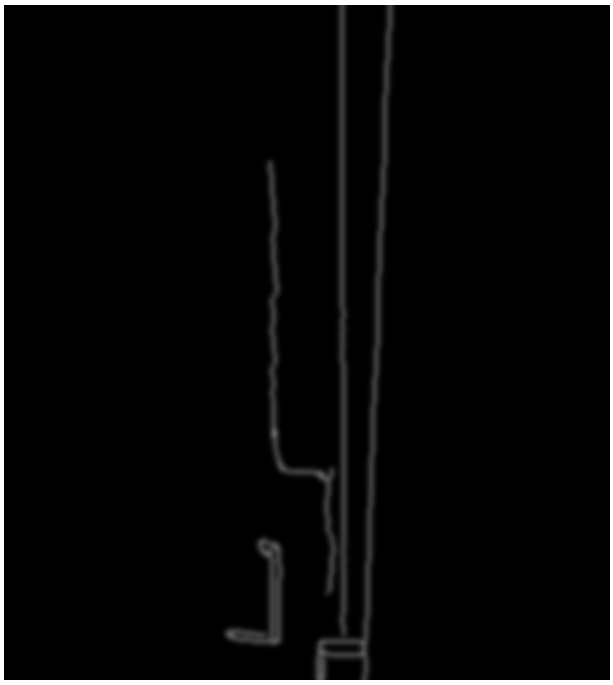

Abbildung 1919: BEV nach erneutem Weichzeichnen

Histogramme

Um das Bild zu analysieren, wurde eine Histogramm-Klasse (die eher an ein spezielles Säulendiagramm erinnert) erstellt, die ein paar Hilfsfunktionen bereitstellt, wie den höchsten Punkt zu finden oder eine Liste der höchsten Punkte (wobei hier pro Gruppe an benachbarten Balken, die die Bedingung erfüllen, nur der mittlere zurückgegeben wird).

Was hierbei noch erwähnenswert ist, ist das es eine „blur" Methode gibt. Der Sinn dieser Methode ist, dass die Werte als Durchschnitt der umliegenden Werte genommen wird, um harte Spitzen mit Lücken dazwischen zu vermeiden. Ein Beispiel gibt folgendes Diagramm, wo Blau die Originaldaten und Orange die verwischten Daten sind. Die Idee ist hier ähnlich zum „Gausschen"-Weichzeichnen, da sich der neue Wert aus dem Durchschnitt der umliegenden Punkte zusammensetzt.

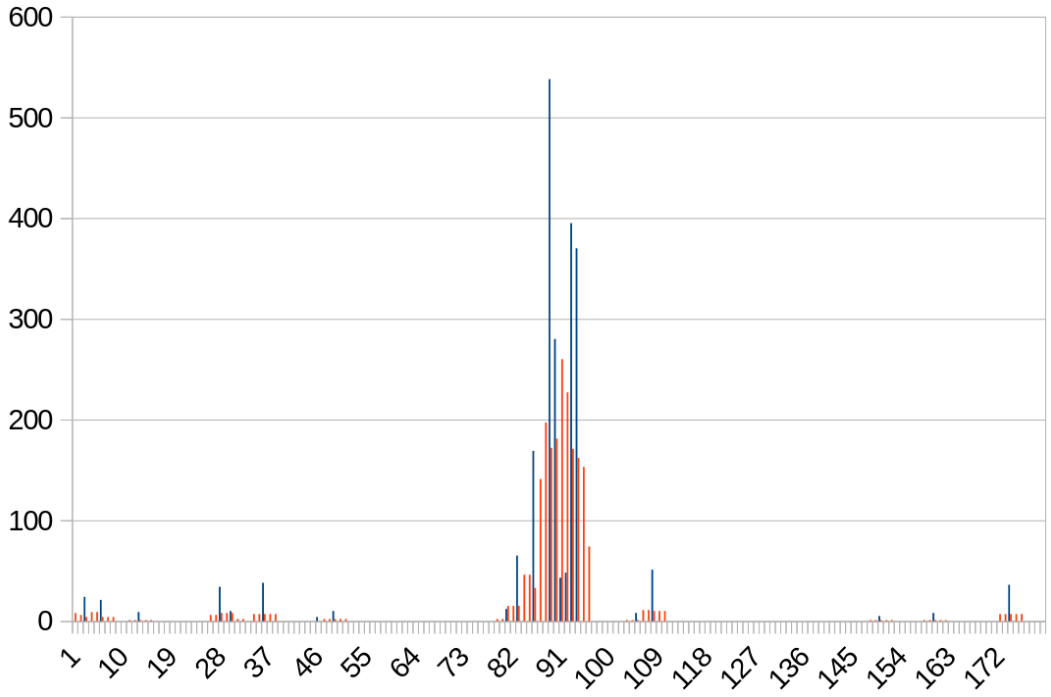

Abbildung 2020: Histogramm der erkannten Linien

Der Effekt ist gerade hier gut zu sehen, wo die beiden Datenpunkte nun keine Lücke mehr dazwischen haben und verbunden sind.

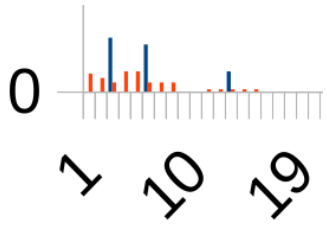

Abbildung 2121: Histogramm bei erkannter Haltelinie

Konvertierung in Linien

Um die aus dem Canny generierten Linien mit den Histogrammen analysieren zu können wird die OpenCV Funktion “findContours” genutzt. Hierdurch wird eine Menge an Linien generiert. Und jede dieser Linien besteht aus mehreren Datenpunkten. Um die Datenmenge zu reduzieren und folglich die Rechenzeit zu verringern, wird die “approxPolyDP” Funktion genutzt. Außerdem werden für die Erkennung kurze Linien entfernt.

Filterung nach Kurven und unplausiblen Datenpunkten

Durch die Bodenreflektion ist es möglich, dass falsche Linien auf dem BEV-Bild erkannt werden und auch in Kurven könnte standardmäßig die normale Straßenlinie an einer Stelle als Haltelinie erkannt werden. Als Schutz dagegen wird mit der Histogramm-Klasse ein Diagramm erstellt, wobei die X-Werte die Winkel sind und die Y-Werte die Länge der Linie. Dieses wird dann verwischt und wenn es mehr als 2 Spitzengruppen gibt, die

mindestens 80% der höchsten Spitzen gibt wird angenommen, dass es zu viele Fehlerdaten gibt, und es wird keine Kreuzung erkannt.

Es folgt nun ein Beispiel, wo das Bild als gut genug erkannt wird und weiterverarbeitet wird, da es nicht zu viele gleichmäßige Störungen gibt.

Abbildung 2222: BEV-Aufnahme

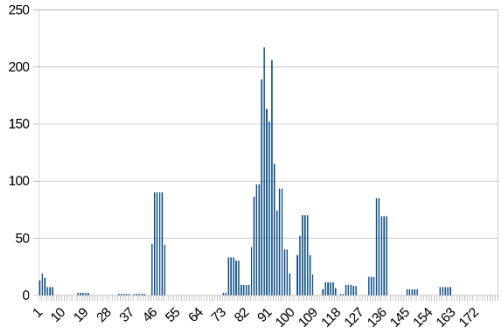

Abbildung 2323: Histogramm zur BEV

Diese Daten könnten theoretisch auch genutzt werden, um herauszufinden wie das Auto im Verhältnis zur Strecke steht, da (wenn es nicht zu viele störende Daten gibt!) die höchste Spitze an dem Winkel sein sollte, wie gerade die Straße im Verhältnis zum Auto ist, um zu sehen in welchem Winkel horizontale Linien sein sollten. Aus zeitlichen Gründen wurde dies aber nicht implementiert.

Histogramm aus Horizontalen Linien

Als nächstes wird ein Histogramm erstellt, indem nur die Linien, die vertikal sind (die Bedingung ist, dass der Winkel 90° mit erlaubter Ungenauigkeit ist), enthalten sind. Der Index ist hierbei die vertikale Position im Bild und der Wert ist Länge der Linie. Nach dem Verwischen wird dann geschaut, wo der höchste Punkt ist, der einen Wert von über 6 hat.

Abbildung 2424: BEV-Aufnahme

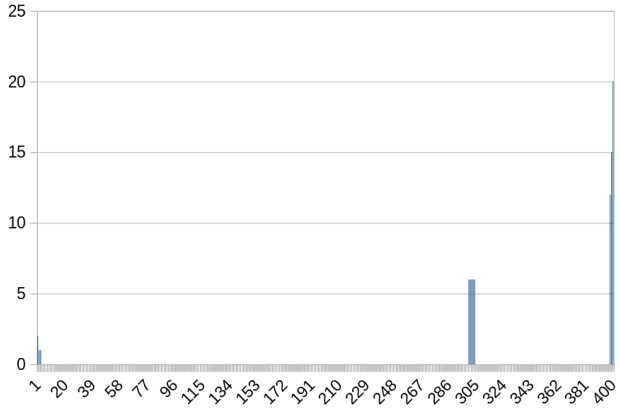

Abbildung 2525: Histogramm der horizontalen Linien

Klassifizierung der Kreuzung

Ausgehend von dieser Höhe wird ab nun auf dem Bild gearbeitet. Zum einen wird an zwei Stellen geschaut, wo die obere und wo die untere Linie ist.

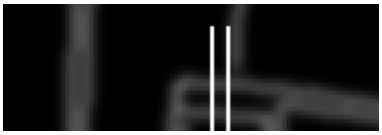

Abbildung 2626: Markierung von Ausfahrten der Kreuzung

Dies ist für eine genauere Distanzbestimmung Wichtig und um später den Kreuzungstypen zuverlässiger zu bestimmen.

Wenn an einer Stelle genau zwei Linien erkannt werden (wieder über die Histogramm-Klasse, wobei x die Position ist und y die Helligkeit; es wird nicht verwischt, da es durch den Gauss-Algorithmus schon in Bildform passiert ist).

Falls es genau zwei Linien gibt, so wird über die bekannte Größe der Kreuzung auf dem gleichen Weg getestet, in welche Richtungen abgebogen werden können. (hierbei werden teilweise Annahmen gemacht, falls nicht genau 2 Möglichkeiten erkannt werden, da der Decider dies in einer alten Version erwartet hatte).

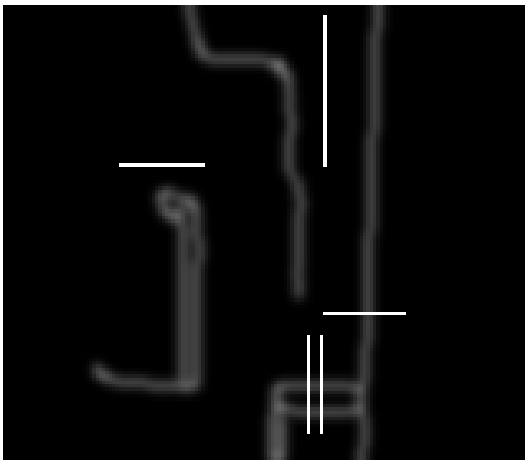

Abbildung 2727: vollständig erkannte Kreuzung

Letzte Schritte

Wenn eine Kreuzung erkannt wurde, muss noch die Bedingung erfüllt sein, dass in den letzten acht Bildern mindestens dreimal eine Kreuzung erkannt wurde.

Wenn auch dies wahr ist so wird die Höhe der tieferen Linie in Zentimeter umgerechnet und sowohl diese Zahl als auch die Abbiegemöglichkeiten über den IPC-Hub gepublisht.

Bewertung

Bildverarbeitung

Die verarbeiteten Bilder selbst konnten sehr gut verarbeitet werden und die Nutzung von Gauss und Canny in Kombination hat sich als relativ zuverlässig rausgestellt, wobei durch den Canny aber teilweise Linien verloren, gegangen sind oder durch die Kombination mit dem Gauss zusammengeführt worden sind. Auch hat die Nutzung des Canny-Algorithmus das Problem, dass manchmal im Hintergrund verschiedenes als Kreuzung erkannt werden kann.

Verbessert werden könnte auch noch, dass die Thresholds für den Canny-Algorithmus dynamisch von der Helligkeit abhängen. Das größte Problem für die Teststrecke war jedoch die bereits erwähnten Reflektionen, da diese teilweise auch in der Fahrspur sind oder dafür sorgen können, dass Haltelinien nicht mehr durchgängig sind.

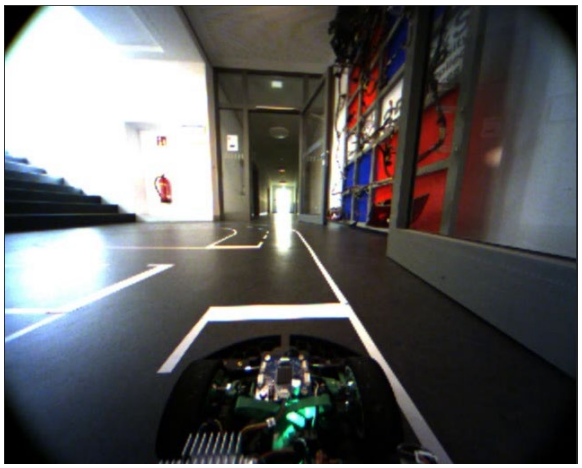

Abbildung 2828: Bildaufnahme mit Störlicht

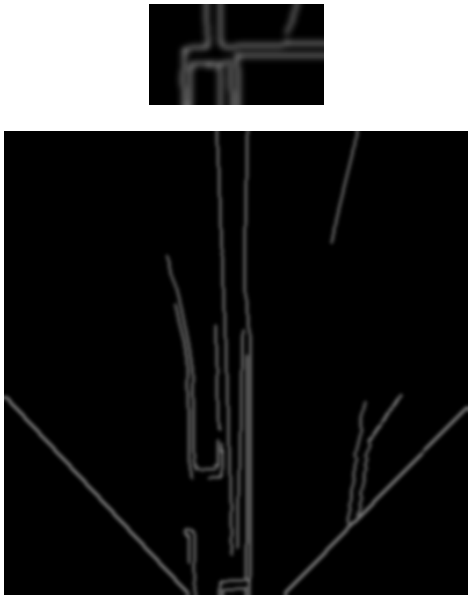

Abbildung 2929: BEV mit Filteralgorithmen zur Lichtentfernung

Ansonsten sollte die BEV noch so angepasst werden, dass die Höhe nicht unnötig hoch ist und nur so hoch wie nötig, da bei steigender Distanz die initiale Erkennung häufiger Fehler aufweist und gerade die Kreuzungsklassifikation hier die Kreuzung häufig falsch erkennt.

Initiale Erkennung

Der Ansatz für die Entfernung und initiale Erkennung funktioniert relativ zuverlässig. Das größte Problem ist, dass durch Licht häufig Bodenreflektionen entstehen, die die Erkennung deutlich erschweren und dadurch die hohe Anzahl der zusätzlichen Bedingungen existiert. Hierdurch kann es leider vorkommen, dass eine Kreuzung nicht erkannt wird oder das etwas fälschlicherweise als Kreuzung erkannt wird.

Der Test, wie gut das Bild ist anhand von der Anzahl der Spitzen über 80% der höchsten Spitze ist möglicherweise auch nicht sehr produktiv, da hierfür alle Linien zur Kenntnis

genommen werden. Auch könnte es verbessert werden, dass wenn die komplette Spur erkannt wird, nur auf der Spur gesucht werden soll.

Der Test auf die Mindestlänge der Haltelinie ist sinnvoll, da hierdurch nur an einer Stelle getestet werden muss und nicht an jeder, wo eine horizontale Linie erkannt wird.

Der finale Test über die Doppellinie ist sinnvoll, wobei es hier aber noch deutlich besser wäre, wenn noch zusätzlich getestet wird, dass beide Linien horizontal sind, um falsche positive Ergebnisse durch beispielsweise Lichteinwirkungen zu vermeiden.

Kreuzungstyperkennung

Die Erkennung vom Kreuzungstypen ist leider sehr rudimentär und ist nur auf diese Kreuzungen mit fester Länge ausgelegt. Sie könnte außerdem noch verbessert werden, indem an mehreren Stellen getestet wird, ob es einen Weg gibt und dann geschaut wird, an wie vielen Stellen es einen Weg gibt. Da der Code im „Decider“ in einer alten Version so aussah, als würden nur gültige Kreuzungstypen erwartet werden, werden teilweise auch die Fahrmöglichkeiten nur basierend auf Vermutungen gesetzt. Dies wäre in einem echten autonomen Auto aufgrund der Sicherheit nicht denkbar.

Andere Ansatzideen

Ein anderer Ansatz, der möglicherweise besser wäre, ist nur auf der BEV ohne Bildverarbeitung zu arbeiten und über die Histogramm-Klasse nur auf einer geraden Linie vor dem Auto die Haltelinie zu suchen, wobei x die Vertikale Position ist und y die Summe der Pixelwerte.

Diese Idee ist leider sehr spät gekommen und es konnten leider allgemein auch nicht so viele Ansätze getestet werden, da zwei von den zuständigen Personen für die Kreuzungserkennung irgendwann verschwunden sind. Ihr Beitrag zur Erkennung bestand hauptsächlich aus dem Aufnehmen von ein paar Videos und deswegen wurden Sie in der Liste der mitarbeitenden Personen nicht aufgeführt.

Die Gruppe der Verkehrsschilderkennung hatte auch eine weitere Idee für eine Erweiterung des ursprünglichen Ansatzes. Ein Modell ähnlich der Verkehrsschilderkennung könnte eingebunden werden, um die Genauigkeit bei der Bestimmung des Kreuzungstypen zu erhöhen. Dabei sollten die Kreuzungen wie die Verkehrsschilder mittels eines KI-Modells erkannt werden und den erkannten Typen über eine Schnittstelle an den Decider übermittelt werden. Dieser Ansatz wurde jedoch nicht weiterverfolgt, da im Gegensatz zu den Verkehrsschildern die Erkennung doch nicht so präzise wie erhofft war und für Optimierungsarbeiten keine Zeit mehr übrig war.

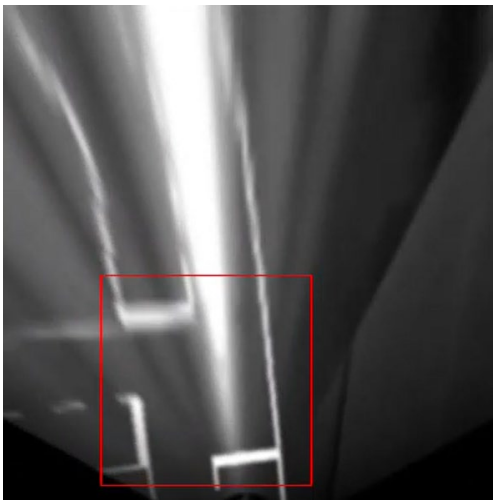

Abbildung 3030: Erkennen der Kreuzung mit OpenCV

Verhalten an Kreuzungen

Beteiligte Personen:

- Philipp Miesner
- Bennet Heidmann
- Jakob Reichstein

Ansatz

Das Verhalten an Kreuzungen wird in einem eigenen Prozess namens Decider implementiert. Wir haben den Prozess so genannt, da dieser nicht nur an Kreuzungen, sondern auch generell entscheidet, ob das Auto fährt, stoppt oder abbiegt. Außerdem ruft er auch die entsprechenden Fahr-Methoden auf. Der Decider ist somit sowohl Teil der Planung als auch der Ausführung.

Um diese Aufgabe zu erfüllen, nutzt der Decider die Informationen, die er über den IPC-Hub von anderen Prozessen erhält:

- Von der Spurverfolgung empfängt der Decider die dort berechneten Geschwindigkeits- und Lenkungswerte.
- Die Verkehrsschilderkennung informiert den Decider, welche Schilder sich vor dem Auto und in welchem Abstand zu diesem befinden.
- Durch die Kreuzungserkennung erhält der Decider Nachrichten, wenn eine Kreuzung identifiziert wird. Diese enthalten den Kreuzungstyp und die Entfernung.
- Die Hinderniserkennung sendet Daten an den Decider, wenn sich ein Objekt im Bereich vor dem Auto befindet.

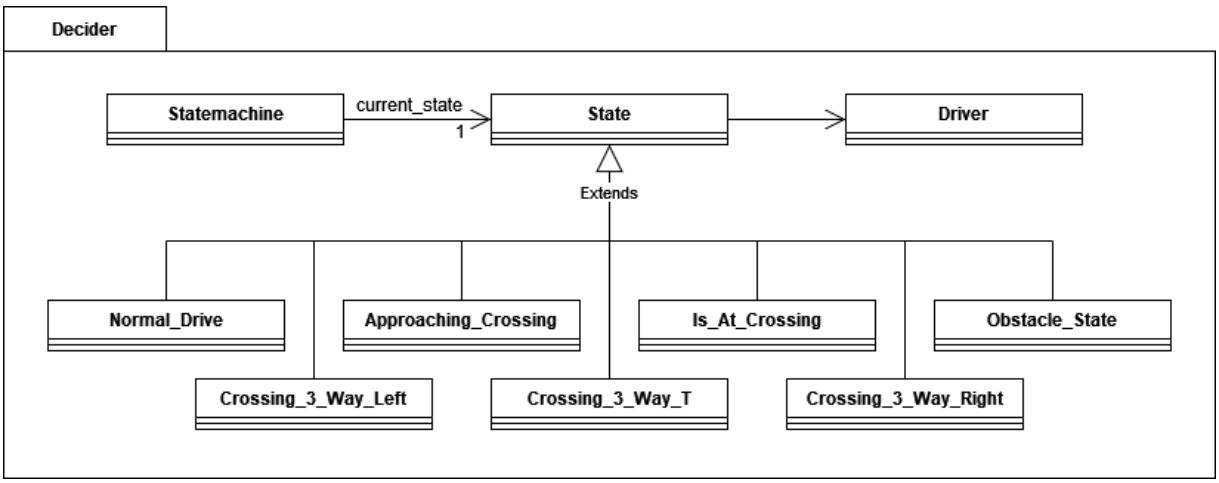

Abbildung 3131: Architektur des Deciders

Das Verhalten des Autos, welches der Decider implementiert, wird mithilfe einer Statemachine umgesetzt. Die Statemachine führt das Verhalten eines States aus, welcher jeweils eine bestimmte Fahrsituation repräsentiert. Die States ändern den

momentan von der Statemachine ausgewählten State basierend auf Informationen, die über den IPC-Hub empfangen werden. Um konkrete Fahr-Methoden auszuführen, benutzen die States den Driver.

Umsetzung

Für das Verhalten an Kreuzungen wird eine Statemachine genutzt, die zwischen den verschiedenen States wechselt und diese ausführt. Die Statemachine stellt dafür bestimmte Methoden bereit:

- **change_state:** Diese Methode bekommt einen State als Parameter übergeben und wechselt in diesen.
- **run:** Diese Methode bekommt optionale Daten als Parameter übergeben und führt den aktuellen State aus.

Die verschiedenen States sind unter anderem das normale Fahren, das Annähern an eine Kreuzung, das Stehen an der Kreuzung, die verschiedenen Kreuzungstypen und der Zustand, wenn ein Hindernis erkannt wurde. Sie sind als Singletons implementiert, damit die Verbindungen zum IPC-Hub jeweils nur einmal initialisiert werden müssen.

- **Normal_Drive:** Dieser State wird aufgerufen, wenn sich das Fahrzeug auf einer geraden Spur befindet und normal fahren soll. Dieser State wird verlassen, sobald eine Kreuzung oder ein Hindernis vor dem Auto identifiziert wurde.
- **Approaching_Crossing:** Dieser State wird aufgerufen, wenn das Fahrzeug eine Kreuzung erkennt und sich der Haltelinie nähert. Wenn sich das Auto an der Kreuzung befindet, wird in den State Is_At_Crossing gewechselt.
- **Is_At_Crossing:** In diesem State kommt das Fahrzeug an der Haltelinie zum Stehen. Daraufhin wird geschaut, welcher Kreuzungstyp erkannt wurde, um als nächstes in diesen State zu wechseln. Bei der Kreuzungsart kann es sich um eine Linksabbiege-, Rechtsabbiege- oder T-Kreuzung handeln.
- **Crossing_3_Way_Right:** Bei dieser Kreuzungsart kann das Fahrzeug geradeaus fahren oder rechts abbiegen. Als erstes wird überprüft, ob Schilder erkannt wurden. Durch die Schilder wird entschieden, wohin das Fahrzeug fährt. Wenn ein Stoppschild identifiziert wurde, wird noch für 2 Sekunden gewartet, bis geradeaus weitergefahren wird. Bei einem Vorfahrtsschild wird ebenfalls geradeaus weitergefahren. Wenn ein Rechtsabbieger-Schild vorhanden ist, biegt das Auto nach rechts ab. Nachdem die Kreuzung durchfahren wurde, wird wieder in den State Normal_Drive gewechselt.
- **Crossing_3_Way_Left:** Bei dieser Kreuzungsart kann das Fahrzeug geradeaus fahren oder links abbiegen. Auch hier werden wieder die Schilder ausgelesen, um den Fahrtweg des Fahrzeugs zu bestimmen und anschließend in den Normal_Drive State zurückgewechselt.

- **Crossing_3_Way_T:** Hier kann das Fahrzeug entweder links oder rechts abbiegen. Sofern ein Linksabbiege-Schild vorhanden ist, wird links abgebogen, sonst immer rechts.
- **Obstacle_State:** In diesen State wird gewechselt, wenn das Fahrzeug ein Hindernis vor sich auf der Fahrbahn erkennt. Daraufhin bleibt das Fahrzeug stehen und wartet so lange, bis das Hindernis verschwunden ist. Danach wird wieder in den Normal_Drive State gewechselt.

Alle States haben überwiegend die gleichen Methoden und unterscheiden sich nur in der Fahraufgabe, welche ausgeführt werden soll:

- **get_instance:** Eine Singleton-Instanz des States wird erzeugt und zurückgegeben.
- **initialize:** Die Methode startet die Kommunikation mit dem IPC-Hub einmal zu Fahrbeginn. Es werden für den State jeweils relevante Message-Typen abonniert (z. B. die Werte der Lane-Detection im Normal_Drive State).
- **on_entry:** Beim Wechsel des States wird diese Methode zu Beginn aufgerufen, welche wiederum die initialize-Methode ausführt. Es wird außerdem eine Nachricht an den IPC-Hub geschickt, dass dieser State nicht mehr taub-geschaltet werden soll (also Nachrichten der abonnierten Themen an ihn weitergeleitet werden). Daraufhin wird in die run Methode übergegangen.
- **run:** In dieser Methode wird der eigentliche Inhalt des States ausgeführt. Es werden die Daten vom IPC-Hub ausgelesen und basierend darauf das bestimmte Verhalten des States umgesetzt. Aus dieser Methode wird in den nächsten State gewechselt.
- **on_exit:** Diese Methode wird beim Verlassen des States aufgerufen. Es wird eine Nachricht an den IPC-Hub gesendet, dass der State taub-geschaltet werden soll, damit Nachrichten nicht mehr an ihn weitergeleitet werden. Wenn der State nämlich nicht mehr aktiv ist, kann er auch keine Nachrichten mehr auslesen, was sonst zu Fehlern bei dem IPC-Hub führen würde.

Der Driver fungiert als Fahr-Service für die States, indem er ihnen verschiedene statische Fahr-Methoden zur Verfügung stellt:

- **initialize**: Diese Methode muss einmal am Fahrstart aufgerufen werden, um die Kommunikation mit dem IPC-Hub zu starten.
- **drive_forward**: Diese Methode wird genutzt, um normal vorwärtszufahren. Dazu sendet sie Start_Driving_Tasks an den IPC-Hub und nutzt die Geschwindigkeits- und Lenkungswerte der Spurverfolgung.
- **drive** & **drive_both_steering_values**: Diese Methoden ermöglichen es dem Aufrufer selbst die Geschwindigkeit sowie den vorderen (bzw. auch den hinteren) Lenkungswert festzulegen.
- **turn_right** & **turn_left**: Diese Methoden werden genutzt, um an Kreuzungen rechts bzw. links abzubiegen. Da dem Decider genauere Umgebungsinformationen

fehlen, wurden diese Methoden an die Teststrecke angepasst, indem für bestimmte Zeiten mit einer entsprechenden Geschwindigkeit und Richtung gefahren wird.

- **stop**: Diese Methode erlaubt es den States, das Auto anzuhalten. Optional kann auch eine Zeit in Sekunden angegeben werden, wie lange das Auto stehen soll.
- **wait:** Hierbei handelt es sich um eine Hilfsmethode, um eine bestimmte Zeit abzuwarten, z. B. zwischen dem Senden zweier Fahr-Befehle.

Bewertung

Das Konzept der Statemachine hielten wir für eine gute Idee zur Umsetzung des Controllers. Es gibt eine klare Aufteilung des Verhaltens, was die Statemachine modular und anpassbar macht. Für unseren Anwendungsfall werden ferner nicht besonders viele Zustände benötigt (siehe oben). In der Realität sieht der Aufbau einer Statemachine für ein autonomes Fahrzeug wahrscheinlich anders aus. Es gibt viele Sonderfälle im echten Straßenverkehr, wodurch die Anzahl an Zuständen immens wachsen würde.

Eine weitere Hürde bei diesem Projekt ist die Testbarkeit unserer Statemachine. Um den Ablauf vollständig testen zu können, werden die gesendeten Informationen aller anderen Subgruppen benötigt. Erst wenn die Teilstände in einem guten Zustand sind, lässt sich die Statemachine final testen. Dies stellte uns lange Zeit vor ein Problem, da sich die anderen Gruppen, wie auch wir, noch in der Entwicklungsphase befanden.

Das Abbiegen des Autos ist mit unserer Implementierung „hardcodiert“. Die anderen Gruppen kümmerten sich um Linienverfolgung, Kreuzungserkennung und Schildererkennung. Unser Teilprojekt stellte hingegen den Ablauf des Autos dar. Dabei fühlten wir uns für das Abbiegen verantwortlich. Da wir uns aber nicht intensiv mit den Themen der anderen Gruppen beschäftigt hatten, war das „hardcodieren“ des Abbiegens die für uns einzige Lösung. Eine HD-Map, in der alle Daten der anderen Prozesse fusioniert werden, wäre hier wahrscheinlich eine gute Option gewesen, um die Umgebungsinformationen den verschiedenen States sowie dem Driver in einer strukturierten Form zur Verfügung zu stellen.

In Zukunft sollten generell Schnittstellen für den Informationsaustauch besser abgesprochen werden. Hier wurde das Festlegen klarer Schnittstellen anfangs dadurch erschwert, dass die einzelnen Teams nur wenig Erfahrung mit den jeweiligen Aufgaben bzw. den dort einsetzbaren Technologien hatten. Dadurch war es schwierig abzuschätzen, welche Informationen und in welcher konkreten Form bereitgestellt werden konnten.

Um unser Projekt zu vervollständigen, könnte das Verhalten bei Hindernissen auf der Strecke noch erweitert werden, da momentan bei Hindernissen nur angehalten wird. Hier wäre es sinnvoll, wenn das Auto (sofern es dabei auf der Strecke bleiben kann) die

Hindernisse umfährt. Dies haben wir allerdings aufgrund der Zeit nicht mehr implementieren können.

Zusammenfassend war die Wahl der Statemachine als Controller für das autonome Fahrzeug grundsätzlich sinnvoll. Die Statemachine lässt sich beliebig erweitern. Lediglich die Bündelung von und der Zugriff auf Umgebungsinformationen wäre mit Implementierung einer HD-Map effektiver gewesen, z.B. für das Abbiegen. Trotz allem, erledigt der Controller die geforderte Aufgabe, Entscheidungen korrekt und situationsabhängig zu treffen.

Retrospektive - Gruppenübergreifend

Das gesamte Projekt wurde hauptsächlich in den kleineren Gruppen bearbeitet. Es ist zwar schade, dass eine Gruppe nach einigen Wochen mit weniger Teilnehmern besetzt war, dafür haben andere Teammitglieder schwächer besetzten Gruppen, falls Fragen aufkamen, immer gerne geholfen. Es gab ein wöchentliches Teammeeting bei welchem Fragen gestellt werden konnten und Probleme Gruppenübergreifend thematisiert wurden. Leider war der Inhalt der ersten vier bis fünf Meetings, dass wir uns mit unseren Aufgaben zunächst vertraut machen müssen und kaum Fortschritt erzielen konnten.

Die meiste teamübergreifende Arbeit geschah in den letzten 2 Wochen. In dieser Zeit haben wir sowohl Integrationstests als auch intensives Bugfixing mit Hilfe von Kollegen mehrerer unterschiedlicher Teams. Das Ergebnis dieser Arbeit hat zu einem zufriedenstellenden Ergebnis geführt, zu welchem jede genannte Person einen angemessenen Beitrag geleistet hat.

Die Arbeit im Team hat jedem von uns Einblicke in mögliche zukünftige Arbeit geben, sowie Vorstellungen und Wünsche diesbezüglich festigen oder hinterfragen lassen. Abschließend möchten wir einen Dank an die Leiter des Kurses aussprechen, welche uns bei vielen Problemen und Ansätzen in der Codeentwicklung immer zur Seite standen.