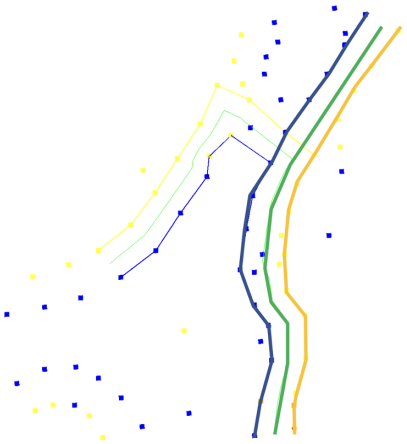

Bahnplanung mit Machine Learning für einen autonomen Formula Student Rennwagen

Abschlussbericht
FWP Formula Student Driverless

Alexander Feix

alexander.feix@tha.de
Matrikelnummer: 2127954
Studiengang: Informatik (IN5)

Technische Hochschule Augsburg
Fakultät für Informatik

Prüfer: Prof. Gundolf Kiefer

18. Januar 2024

1. Einleitung

Formula Student ist ein Wettbewerb indem Studierende aus verschiedenen Universitäten und Hochschulen aus der ganzen Welt jährlich neue Rennwagen bauen. Mit diesen Rennwagen treten die verschiedenen Teams jährlich auf verschiedenen Events gegeneinander an. Typischerweise repräsentiert ein Team eine Universität oder Hochschule und besteht aus einem interdisziplinären Team aus verschiedenen Studiengängen. Stark vertreten sind Studierende aus Studiengängen wie Maschinenbau oder Elektrotechnik, welche maßgeblich an der Konstruktion und Betrieb des Rennwagens beteiligt sind. Daneben gibt es noch weitere Studiengänge, wie beispielsweise Informatik. Das Rennauto kann sowohl mit Fahrer, als auch ohne Fahrer, also voll autonom fahren. Das voll autonome Fahren wird im Formula Student Bereich *Driverless* bezeichnet. Driverless umfasst mehrere Disziplinen, welche ohne Fahrer und ohne äußeren Eingriff bewältigt werden müssen. Für Informatik-Studenten bietet dieser Bereich eine spannende Möglichkeit die im Studium gelernte Theorie in der Praxis anzuwenden. Neben statischen Disziplinen, wo die technische Entwicklung der Saison am Fahrzeug gegenüber Fachexperten aus der Industrie verteidigt wird, gibt es auch dynamische Disziplinen. In den dynamischen Disziplinen wird der Rennwagen tatsächlich auf einer Strecke gefahren. Die spannendste Disziplin ist *AutoX („AutoCross“)*. Bei dieser Disziplin wird das Rennfahrzeug auf eine vollständig unbekannte Strecke gestellt und muss diese selbst für eine Runde durchfahren. Diese Strecke wird dann in der nächsten Disziplin *Trackdrive* für zehn Runden durchfahren. Eine Strecke ist dabei durch blaue Hütchen links und gelbe Hütchen rechts der Strecke begrenzt, sowie mehrere hundert Meter lang und mindestens drei Meter breit [5]. Um diese Strecke autonom zu Fahren, müssen zunächst die Hütchen erkannt werden, um eine Karte der Strecke aufzubauen. Die Hütchen werden mithilfe von Kameras und einem LiDAR Sensor erkannt. Mit dem LiDAR kann man die Entfernung zu einem Objekt schätzen und mit den Kameras zusätzlich die Farbe des Objekts erkennen. Im Formula Student Bereich sind diese Objekte natürlich in der Regel die blauen und gelben Hütchen. Nachdem die Hütchen erkannt wurden, muss das Fahrzeug lokalisiert werden. Die Lokalisierung funktioniert über einen *SLAM* Algorithmus. SLAM steht für *Simultaneous Localization And Mapping*. Dieser Algorithmus lokalisiert das Fahrzeug auf der Strecke und kann gleichzeitig eine Karte der Umgebung mit den erkannten Hütchen aufbauen und korrigieren. Nach der Lokalisierung und dem Aufbauen der Karte wird die schnellste Rennlinie für das Rennfahrzeug durch die Strecke gesucht. Die Bahnplanung ist der Kernfokus dieser Arbeit. Der Bahnplanungsalgorithmus hat die Aufgabe eine möglichst kurze und schnelle Strecke durch die Karte mit den Hütchen zu finden. In der Praxis sind in der ersten gefahrenen Runde einer unbekannten Strecke beim direkten Start nicht alle Hütchen der Strecke bekannt. Die Hütchen werden direkt nach

der Erkennung der Karte hinzugefügt und der Bahnplanungsalgorithmus berechnet eine neue Strecke.

In dieser Arbeit wird ein neuer Teil des Bahnplanungsalgorithmus vorgestellt, welcher mit modernen Machine Learning Methoden implementiert wurde. Diese Modifizierung des bereits existierenden Bahnplanungsalgorithmus hat einige schwerwiegende Probleme des alten Algorithmus effizient gelöst. Das Projekt wurde von Oktober 2022 bis Juli 2023 als Teil des Wahlpflichtfachs *Formula Student Driverless* der Technischen Hochschule Augsburg bei dem Rennteam *StarkStrom Augsburg* umgesetzt.

1.1 Aufbau der Arbeit

Zunächst werden im Kapitel 2 einige notwendige Grundlagen zum allgemeinen Verständnis der vorgestellten Konzepte vermittelt. Hier wird zunächst das Konzept von *Machine Learning* anhand eines einfachen Beispiels erklärt und danach die Aufgaben und Funktionen des Bahnplanungsalgorithmus erläutert. Im Kapitel 3 wird dann die eigene Tätigkeit der Saison und der Hauptteil dieser Arbeit vorgestellt. Zunächst wird der implementierte *Fréchet Distance Algorithmus* [1] zum Aufbau der Fahrbahn kurz vorgestellt, sowie der Aufbau des bisherigen Bahnplanungsalgorithmus als Referenz für das später implementierte *Neuronale Netz*. Danach wird die Entscheidungsgrundlage des bisherigen Algorithmus für die Auswahl eines Pfades dargestellt. Für die Implementierung des Neuronalen Netzes werden zunächst die Methoden und Prozesse zur Datensammlung erläutert, sowie die konkrete Implementierung. Im letzten Kapitel 4 werden die Ergebnisse des Neuronalen Netzes, sowie des bisherigen Bahnplanungsalgorithmus mit verschiedenen Metriken verglichen und anschließend evaluiert. Schlussendlich werden einige Verbesserungen oder alternative Vorschläge als weiterführende Arbeit für dieses Projekt vorgeschlagen.

2. Grundlagen

2.1 Machine Learning

Machine Learning is the science (and art) of programming computer so they can learn from data. So wird Machine Learning in dem Buch "*Hands On Machine Learning, Kapitel 1*" allgemein definiert. Viel besser kann man Machine Learning aber an einem Alltagsbeispiel erklären. Woher weiß ein Spam-Filter in einem Mailing-Programm, was Spam ist und welche Mails kein Spam sind? Spam-Filter sind gewöhnlich Machine Learning Programme, welche *gelernt* haben, wann eine neue Mail Spam oder kein Spam ist. Diese Programme werden auf den Grundlage eines Datensatzes entwickelt, welcher viele Mails enthält, wobei bei jeder einzelnen Mail die Information bekannt ist, ob diese Mail Spam Inhalte enthält oder nicht. Das Machine Learning Programm versucht nun auf der Grundlage dieser Daten vorherzusagen, ob eine neue Mail Spam ist. Meistens funktioniert dies so, dass das Programm nach einiger Zeit, selbst einige Merkmale in einer Mail (z.B. Autor, Links oder Medien-Inhalte) stärker oder einfacher gewichtet. So ist eine Mail beispielsweise sehr auffällig, wenn sehr viele Links zu Websiten enthalten sind.
Ein traditioneller Programmieransatz eines neuen Programms würde wie in Abbildung 2.1 aussehen. Das Problem bei dem traditionellen Ansatz ist, dass die Regeln für den Algo-

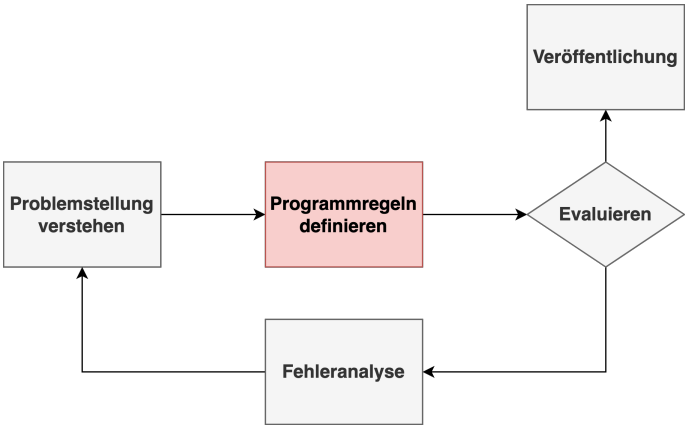

Abbildung 2.1: Traditioneller Programmieransatz [6, vgl. Abb. 1.1]

rithmus eines Programmes bekannt sein müssen. Es gibt aber Anwendungen, wie in dieser Arbeit vorgestellt, in der diese Programmregeln und Gewichte nicht bekannt sind oder

nur schwer herauszufinden sind. Daher wird ein Machine Learning Modell auf Trainingsdaten trainiert, um diese Regeln selbst herauszufinden und immer weiter zu optimieren. Der Machine Learning Ansatz verändert somit den klassischen Programmieransatz, wie in Abbildung 2.2 dargestellt. [6, Kapitel 1]

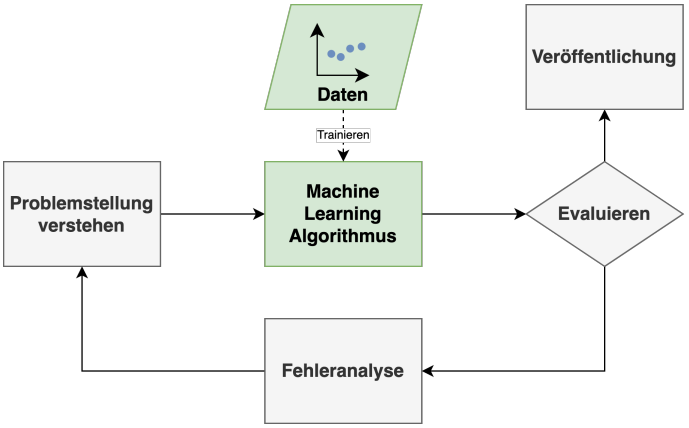

Abbildung 2.2: Machine Learning Ansatz [6, vgl. Abb. 1.2]

2.2 Bahnplanung

Damit das Rennfahrzeug durch die Strecke navigieren kann, wird eine Ideallinie benötigt. Der Bahnplanungsalgorithmus hat somit die Aufgabe eine möglichst korrekte, kurze und dynamische Ideallinie zu planen. Diese Ideallinie wird auch *Trajektorie* genannt, da diese Linie im optimalsten Fall die am schnellsten befahrbare Strecke für das Rennfahrzeug ist. Der Bahnplanungsalgorithmus bekommt als Eingabe eine Karte mit den Hütchenpositionen, wobei die Hütchen optional noch Farbinformationen enthalten können. Zusätzlich wird die aktuelle Fahrzeugposition als Eingabe verwendet. Eine Fahrzeugposition enthält die Fahrzeugposition auf der Karte, sowie die aktuelle Fahrtrichtung. Nach dem kompletten Durchfahren der ersten Runde, wird ein Signal ausgegeben, dass die Trajektorie gefunden wurde und der Bahnplanungsalgorithmus keine neue Bahn mehr planen muss. Außerdem werden alle Informationen über die gefundene Strecke ausgegeben, wie beispielsweise die Trajektorie, die Breiten der Strecke oder die Hütchen-Verbindungen der linken und rechten Streckenbegrenzung. Diese Streckenbegrenzungen werden im folgenden *Boundaries* bezeichnet. Eine Strecke besteht aus zwei Boundaries, der linken und der rechten Streckenbegrenzung. Eine Boundary enthält Informationen wie deren Pfadlänge, Anzahl der enthaltenen Hütchen oder die Winkel zwischen zwei verbundenen Hütchen.
Der grundlegende Bahnplanungsalgorithmus bei StarkStrom Augsburg existiert bereits seit einigen Jahren. Zuletzt wurde der komplizierte Algorithmus durch eine einfachere Implementierung ersetzt, welche in im weiteren algorithmisch dargestellt ist [1]. Der Bahnplanungsalgorithmus wurde immer weiter entwickelt, dokumentiert und optimiert. Der Algorithmus plant die Bahn unter Berücksichtigung einiger Parameter, welche einfach in einer Config-Datei angepasst werden können. Außerdem gibt der Algorithmus nur regelkonforme Fahrbahnen aus.
Das Bahnplanungsalgorithmus iteriert mit einer Rate von 20 Hz. Somit wird die Karte 20 mal pro Sekunde aktualisiert und der Algorithmus darf pro Iteration maximal 50 ms benötigen.

Da der Bahnplanungsalgorithmus in einigen Fahrsituationen aufgrund dessen Implementierung falsche Bahnen erzeugt hat, wurde der Algorithmus mithilfe von Machine Learning Methoden modifiziert und teilweise verbessert. Der modifizierte Algorithmus behebt einen konzeptionellen Fehler, welcher unter Umständen zur Disqualifikation einer Disziplin auf einem Event geführt hätte.

3. Machine Learning in der Bahnplanung

Der Bahnplanungsalgorithmus hat prinzipiell verschiedene Aufgaben. Zunächst müssen die linke und rechte Boundary mit ihren Hütchen jeweils verbunden werden. Nachdem alle Hütchen einer Boundary zugeordnet und somit verbunden wurden, wird die Mittellinie gesucht. Die Mittellinie wird gefunden, indem man für jedes Hütchen der linken Boundary das am kürzesten entfernte Hütchen der rechten Boundary findet [1]. Von diesen beiden Punkten wird der Mittelpunkt einer Liste *middle_points* hinzugefügt, welche im weiteren zur Ideallinie optimiert wird. Ein Ausschnitt des Bahnplanungsalgorithmus zum Finden des am kürzesten entfernten Punkts der gegenüberliegenden Boundary ist in Algorithmus 1 beschrieben.

Algorithm 1 Implementation des Fréchet Distance Algorithmus

for $left\_point$ in $left\_points$ **do**
  $closest\_distance \leftarrow infinity$
  $closest\_point \leftarrow -1$
  for $right\_segment$ in $right\_segments$ **do**
    $\vec{l} \leftarrow q_2 - q_1$
    $vector\_length \leftarrow \vec{l}[0]^2 + \vec{l}[1]^2$
    if $vector\_length$ is near 0 **then**
      $distance \leftarrow ||\vec{l} - p_1||$
      if $distance < closest\_distance$ **then**
        $closest\_point \leftarrow q_1$
      end if
      continue
    end if
    $\vec{p} \leftarrow p_1 - q_1$
    $t \leftarrow max(0., min(1., \frac{\vec{p} \cdot \vec{l}}{vector\_length}))$
    $projection \leftarrow q_1 + t \times q_2 - q_1$
    $distance \leftarrow ||p_1 - projection||$
    if $distance < closest\_distance$ **then**
      $closest\_point \leftarrow projection$
    end if
  end for
  append $left\_point, closest\_point$ **to** $middle\_points$
end for

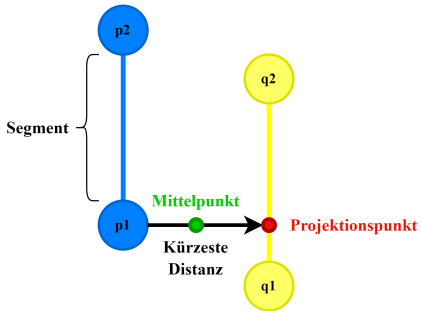

Abbildung 3.1: Der Projektionspunkt liegt auf der gegenüberliegenden Strecke

Dabei werden die Punkte der gegenüberliegenden Boundary in Segmente aufgeteilt. Ein Segment beschreibt zwei benachbarte Punkte. Der Algorithmus sucht einen Punkt, welcher auf der Verbindungsstrecke dieser beiden Punkte liegt. Der gefundene Punkt kann entweder auf der Strecke oder direkt auf einem der beiden Punkte liegen [1]. Der Mittelpunkt der orthogonalen Verbindung des gefundenen Punktes zur gegenüberliegenden Boundary wird der Mittellinie hinzugefügt. Nachdem weitere Punkte gefunden wurden, werden alle einem globalen Pfad hinzugefügt, der später zur Trajektorie optimiert wird.

Die Bahnplanung ist so konzeptioniert, dass sie in Echtzeit während das Fahrzeug auf der Strecke fährt, verschiedene mögliche Fahrbahnen zu verschiedenen Zeitpunkten berechnet. Die verschiedenen Fahrbahnen zum gleichen Zeitpunkt lassen sich unter anderem wegen einigen *False-Positives* in der Karte erklären. Das bedeutet, dass einige Punkte der Karte als Hütchen hinzugefügt werden, welche aber in der Realität gar keine Hütchen sind. Diese Hütchen wurden fälschlicherweise von der Objekterkennung als Hütchen markiert. Ein Beispiel für mehrere mögliche Pfade für eine Fahrzeugposition ist in Abbildung 3 dargestellt. Zusätzlich ist zu keinem Zeitpunkt ein konkreter Zielpunkt für das Fahrzeug definiert, dem das Fahrzeug folgen könnte. Daher baut der Bahnplanungsalgorithmus auf einer Such-Heuristik auf, welche in [7] erstmals vorgestellt wurde. Dabei werden verschiedene mögliche Fahrbahnen in Betracht gezogen und durch ein *Ranking-Problem* gelöst.

Eine Teilaufgabe der Bahnplanung besteht also darin, aus einer Auswahl von geometrisch möglichen Fahrbahnen die beste Fahrbahn zu finden.

3.1 Problemstellung

Bisher wurde eine relativ simple Gleichung für die Auswahl des besten Pfades verwendet. Die Gleichung enthält geometrische Merkmale des Pfades. So werden jedem möglichen Pfad durch diese Gleichung eine bestimmte Punktzahl zugewiesen. Der Pfad mit der höchsten Punktzahl wird als bester Pfad ausgewählt. Seien w_0,...,$w_3 \in -1, ..., 1$ einstellbare Parameter, L die Pfadlänge, B die Abweichung der Streckenbreite, G die Pfadglättung, A die Anzahl der linken und rechten Hütchen und V die Winkelvarianz der beiden Boundaries, dann wird die Punktzahl eines Pfades P_{Pfad} durch folgende Gleichung bewertet:

$$P_{Pfad} = L + w_0 \times B + w_1 \times G + w_2 \times A + w_3 \times V \tag{3.1}$$

Die Gleichung wurde mit den verschiedenen Parametern immer weiter über die Jahre optimiert. In den meisten Fällen wurde dadurch immer der längste Pfad als bester Pfad ausgewählt. Naiv hat dieser Ansatz auch keinen logischen Fehler, da ein langer Pfad viel Sicherheit für die nächste Strecke direkt vor dem Fahrzeug sicherstellt. Trotzdem gibt es in besonders engen Kurven Probleme mit dem Algorithmus. In vielen Fällen plant der Algorithmus die Ideallinie nicht korrekterweise um die Kurve herum, sondern nimmt eine Abkürzung, wie in Abbildung 3.6 rechts unten dargestellt. Diese Abkürzung schneidet die Kurve, sodass diese nicht korrekt gefahren wird. Der Algorithmus plant diesen Pfad, da dieser geometrisch und auch laut dem Formula-Student-Regelwerk zu den Streckenbedingungen plausibel ist. Zusätzlich hat dieser Pfad gegenüber den anderen möglichen Pfaden dieser Fahrzeugposition auch noch die größte Länge. Aus diesen Gründen sieht der bisherige Algorithmus diesen Pfad als unproblematisch und als den besten Pfad an. Tatsächlich würde diese Abkürzung der Strecke aber zur Disqualifikation einer Disziplin führen.

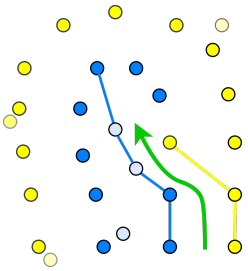

Abbildung 3.2: Ein blaues Hütchen wird fälschlich als gelbes Hütchen identifiziert. Die Rennstrecke wird verlassen, da dieser Pfad für den Algorithmus geometrisch plausibel ist.

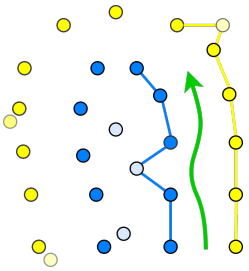

Abbildung 3.3: Einige False-Positive Hütchen werden erkannt. Die Strecke wird nur fast verlassen, das Fahrzeug bleibt auf der Strecke.

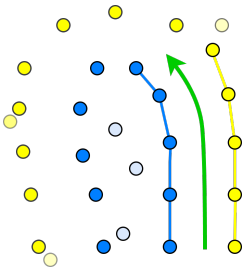

Abbildung 3.4: Keine False-Positives werden in die Bahnplanung miteinbezogen. Das Fahrzeug folgt dem optimalen Pfad.

Um dieses Problem zu lösen, wurde ein Machine Learning Ansatz gewählt. Dieser Ansatz wurde gewählt, da zu diesem Zeitpunkt über ein Jahr lang keine einfache und algorithmisch darstellbare Lösung für dieses Problem gefunden wurde. Stattdessen sollte ein Machine Learning Modell trainiert werden, um den Teil der Bahnplanung zu ersetzen, welcher für die Auswahl des korrekten Pfades zuständig ist.
Das Modell sollte also als Eingabe eine Menge möglicher Pfade erhalten und den Index des besten Pfades ausgeben. Der ausgegebene Pfad wird dann zeitweise bis zur neuen Iteration des Algorithmus als korrekte Fahrbahn für das Fahrzeug verwendet.

3.2 Trainingsdaten

Für den Datensatz wurden einerseits alte Strecken aus früheren Testfahrten benutzt, aber auch neue Strecken. Für die Generierung von neuen Strecken wurde ein eigenes Tool entwickelt. Dieses Tool erzeugt aus handgemalten Strecken auf Papier oder digital auf einem Tablet, eine regelkonforme Strecke nach dem Formula Student Regelwerk. Dabei können verschiedene Einstellungen, wie beispielsweise die minimale Streckenbreite oder die Glättung der handgezeichneten Kurven vorgenommen werden. Durch dieses Tool konnten schnell und einfach viele Strecken und somit Trainingsdaten generiert werden.

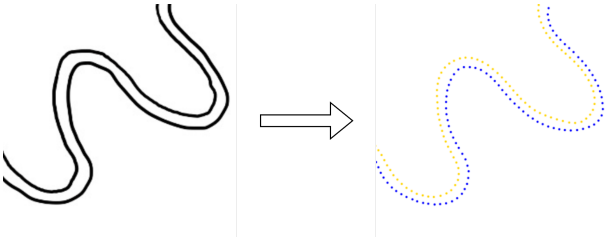

Abbildung 3.5: Die Zeichnung der Strecke wird automatisch in eine Strecke mit Hütchen konvertiert.

Da das Machine Learning Modell keine Bahn selbst planen, sondern nur bewerten soll, wurde der Bahnplanungsalgorithmus verwendet um die Boundaries und die Trajektorie zu erstellen. Zunächst wurde eine Strecke in verschiedene Teilstrecken unterteilt. Jede Teilstrecke besteht aus allen Hütchen, welche vor dem Fahrzeug in einer Entfernung von 30 Metern zu erkennen sind. Zusätzlich hat das Tool einen Regel-Parameter, welcher den Prozentsatz an künstlich generierten False-Positive Hütchen um jedes real existierende Hütchen steuern kann. Durch diese Funktion und die maximale Sichtweite von 30 Metern konnte eine reale Umgebung des Rennfahrzeugs perfekt simuliert werden. Eine Strecke wurde dabei in verschiedene Fahrzeugpositionen unterteilt, wobei das Fahrzeug zur vorherigen Position mindestens 1m entfernt sein musste. Dadurch standen dem Fahrzeug für

jede Position andere Hütchen voraus.
Der Bahnplanungsalgorithmus generiert nun für jede Teilstrecke die Boundaries und die Auswahl der möglichen Pfade für diese eine Fahrzeugposition. Die Menge der möglichen Pfade hängt stark von der Strecke ab und variiert durchschnittlich zwischen zehn und 600 möglichen Pfaden.
In Abbildung 3.6 sind für eine Fahrzeugposition die vier besten Fahrbahnen mit der höchsten *Intersection Over Union (IoU)* dargestellt. Die IoU beschreibt einen Wert, welcher die Überlappungsfläche zweier Polygome repräsentiert. Der Maximalwert 1.0 drückt die volle Überlappung der Flächen aus. Dabei beschreibt GT_{Poly} das Polygom der Ground-Truth Bahn, $Candidate_{Poly}$ das Polygom der geplanten Fahrbahn und $Intersection_{Poly}$ die Überlappungsfläche beider Polygome.

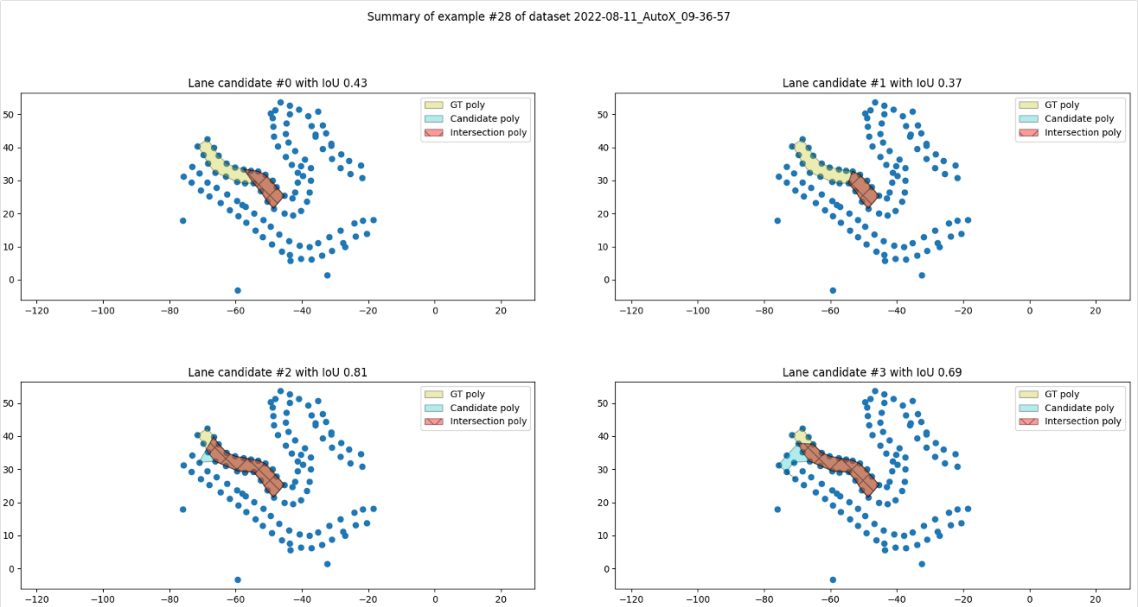

Abbildung 3.6: In diesem Beispiel besitzt Fahrbahn 2 den besten Pfad, da dort der IoU-Score am höchsten ist. Obwohl Fahrbahn 3 einen längeren Pfad plant, ist dieser nun nicht mehr die beste Wahl, da hier die Fahrbahnstrecke abkürzt wird und die Bahn somit eine niedrigere Überlappungsfläche mit der Ground-Truth Bahn hat.

3.3 Implementierung

Die Kernaufgabe des Machine Learning Modells ist das Finden des besten Pfades aus einer Menge von Pfaden. Um den besten Pfad zu finden, müssen die Pfade bewertet werden. Wie bereits erwähnt wird dieses Problem als Ranking-Problem definiert. Ranking-Probleme sind im allgemeinen mittlerweile weit verbreitet, besonders in Anwendungen für Suchmaschinen. Es gibt verschiedene Möglichkeiten eine Liste aus Einträgen zu sortieren. Der einfachste und effizienteste Ansatz für dieses Problem ist der *Pairwise-Approach* [4]. Der Vorteil dieses Ansatzes ist die Reduzierung auf ein binäres Klassifikationsproblem, was die Architektur für das Neuronale Netz einfacher gestaltet und die Rechenzeit stark mindert. Dabei werden immer paarweise zwei Pfade verglichen. Der bessere Pfad wird dann mit dem nächsten Pfad verglichen.

Das trainierte Modell ist ein Neuronales Netz, bestehend aus zwei vollvernetzten Schichten, inspiriert von *RankNet* [2]. RankNet ist eine Architektur, welche den Pairwise-Ranking Ansatz in einer einfachen Architektur umsetzt und Echtzeit-Inferenz ermöglicht. Die Eingabe besteht aus einem achtdimensionalen Feature-Vektor aus geometrischen Merkmalen der Fahrbahn. Der Feature-Vektor setzt sich aus der Fahrbahnlänge (Mittelwert der Länge

beider Boundaries) und der Anzahl der Hütchen für jede Boundary. Für die Schätzung der geometrischen Plausibilität wird die Varianz der Streckenbreite und für jede Boundary die Varianz der einzelnen Längen der Segmente, sowie die Varianz der Winkel bei aufeinander folgenden Segmenten verwendet. Das Neuronale Netz besteht in der ersten Schicht aus 8×100 Parametern und in der zweiten Schicht aus 100×1 Parametern. Als Aktivierungsfunktion wird ReLU verwendet. Wie auch bei dem RankNet Ansatz, wird die vorhergesagte Wahrscheinlichkeit p_{pred}, welche angibt ob der erste Pfad besser als der zweite Pfad ist, von den beiden Feature-Vektoren mit

$$p_{pred} = sigmoid(Net(x_1) - Net(x_2)) \tag{3.2}$$

berechnet, wobei Net$(\cdot)$ die Ausgabe der zweiten Schicht des Netzes darstellt. Als Loss-Function wird *Binary-cross-entropy (BCE-Loss)* verwendet, wobei die Ground-Truth Wahrscheinlichkeit p_{GT} über die *Intersection-Over-Union (IoU)* Funktion berechnet wird. Der IoU-Score beschreibt dabei die Überlappungsfläche der geplanten Fahrbahn und der tatsächlichen Bahn (Ground-Truth Polygon). Die Ground-Truth Wahrscheinleichkeit p_{GT} wird dabei über die beiden IoU-Werte IoU_1 und IoU_2 berechnet:

$$p_{GT} = sigmoid(\lambda_{IoU}(IoU_1, IoU_2)) \tag{3.3}$$

Der Faktor λ_{IoU} erhöht den Unterschied der IoU-Werte, um die Trainingszeit zu verringern.

Für das Training wurden mehrere Strecken mit insgesamt 307958 möglichen Pfaden auf 2220 Fahrzeugpositionen verwendet. Somit bestand ein Ranking-Problem (für eine Fahrzeugposition) durchschnittlich aus 139 möglichen Pfaden. Um die Varianz des initialen Datensatzes zu erhöhen, wurden Strecken, welche mit dem Streckengenerator-Tool generiert wurden, mit verschiedenen False-Positive-Raten angepasst. Zusätzlich wurde die maximale Sichtweite mit 30m und 50m simuliert. So wurde der initiale Datensatz nochmals deutlich erweitert.
Das Neuronale Netz wurde mit PyTorch implementiert und mit TensorFlow Lite überwacht und evaluiert. Das Modell wurde in 200 Epochen mit einer Batch-Size von 8192 trainiert. Als Optimizer wurde Adam mit einer Lernrate von 0.008 verwendet. Da das Modell aus ingsgesamt nur 1001 Parameter besteht, dauerte das Training auf einer NVIDIA GeForce RTX 3070 Ti Grafikkarte nur wenige Minuten. [1]

[1] Die Architektur des Neuronalen Netzes und das Visualisierungs-Tool in Abbildung 3.6 wurden intern im Team entwickelt.

4. Analyse und Fazit

Der vorgestellte Algorithmus wurde zunächst mit Python logisch implementiert und danach mit C++ in den gesamtem Bahnplanungsalgorithmus integriert, da die gesamte Bahnplanung in C++ programmiert ist. Das Neuronale Netz läuft auf einer Intel i7 CPU des Fahrzeugcomputers, da die GPU für rechenintensive Netze benutzt wird, wie beispielsweise die Objekterkennung. Die folgende Auswertung wurde mit verschiedenen Einstellungen der False-Positive-Rate erstellt.

4.1 Ergebnisse

Im Folgenden werden der bisherige Bahnplanungsalgorithmus CLC, welcher auf einer simplen Formel [3.1] basiert, mit dem neuen Neuronalen Netz zur Pfadbewertung verglichen. Der Vergleich wird anhand verschiedener Metriken aufgezeigt, welche in der Evaluation [4.2] zusammengefasst werden.
Die Auswertung wurde auf 666 Ranking-Problemen durchgeführt, welche im Trainingsprozess nicht verwendet wurden und dem Modell somit unbekannt waren.

4.1.1 Quantil-Analyse

Bei der Quantil-Analyse werden die verschiedenen IoU-Werte in Quantile oder Intervalle aufgeteilt. Ein kritischer Fehler klassifiziert Bahnen, welche die korrekte Fahrbahn abkürzen würden oder offensichtlich fehlerhaft geplant wurden($IoU \leq 0.7$). Dagegen entsprechen korrekte Bahnen nehezu oder genau der Ground-Truth Bahn ($IoU \geq 0.9$). Andere Bahnen wurden entweder zu kurz geplant oder sind leicht abweichend, indem beispielsweise einige False-Positive Hütchen in die Bahn miteinbezogen wurden ($0.7 < IoU < 0.9$).

	Ranking NN (Anzahl)	Ranking NN (%)
Kritischer Fehler	10	1.5
Bahn zu kurz oder abweichend	26	3.9
Korrekte Bahn	630	94.6

Tabelle 4.1: Knapp 95% der ausgewählten Pfaden des Neuronalen Netzes sind korrekt. Das Fahrzeug würde nur in sehr wenigen Fällen die Fahrbahn verlassen.

	CLC (Anzahl)	CLC (%)
Kritischer Fehler	7	1.1
Bahn zu kurz oder abweichend	24	3.6
Korrekte Bahn	635	95.3

Tabelle 4.2: Der bishere Algorithmus *CLC* mit der einfachen Formel 3.1 klassifiziert mehr Bahnen oberhalb des 90%-Quantils.

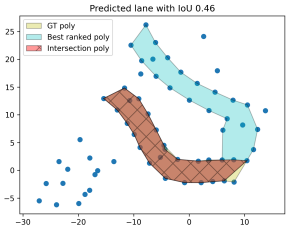

Abbildung 4.1: Die Fahrbahn wird offensichtlich falsch geplant und würde eine Abkürzung der Strecke vorschlagen.

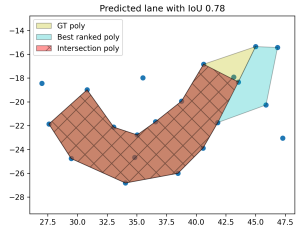

Abbildung 4.2: In der Fahrbahn werden einzelne False-Positives miteinbezogen, die Strecke wird aber nicht direkt abgekürzt.

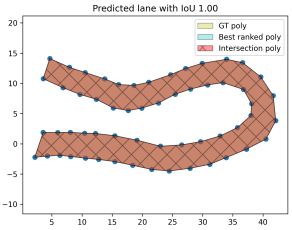

Abbildung 4.3: Die Fahrbahn wird korrekt geplant und kürzt die Strecke nicht ab.

4.1.2 IoU-Durchschnitt

Der Durchschnitt wird über alle IoU-Werte der ausgewählten Bahnen berechnet. Der maximal erreichbare IoU-Wertedurchschnitt bezeichnet dabei den best möglichen Durchschnitts-IoU-Wert, falls die Algorithmen jeweils immer die Bahn mit dem höchsten IoU-Wert gewählt hätten. Dieser Wert ist nicht maximal, da es einzelne Situationen gibt, in welchen der Bahnplanungsalgorithmus die Ground-Truth Bahn nicht als Auswahl zur Verfügung stellt.

	CLC	Ranking-NN
Erreichte IoU (⌀)	0.97669	0.97695
Maximal erreichbare IoU (⌀)	0.99722	0.99722

Tabelle 4.3: Der IoU-Durchschnitt aller gewählten Bahnen ist bei dem Neuronalen Netz etwas höher als beim bisherigen Algorithmus.

4.1.3 Korrekte Bahnen

Bisher wurden korrekte Bahnen mit einem IoU-Wert über 0.9 definiert. Diese Bahnen werden grundsätzlich richtig aufgebaut, können aber einzelne False-Positive Hütchen miteinbeziehen oder auch die Strecke teilweise abkürzen. Das Abkürzen der Strecke kann bei einem IoU-Wert über 0.9 aber nur bei sehr langen Bahnen passieren, da sich dort das Verhältnis verändert. Wenn ein langer Pfad eine kleine Abkürzung der Strecke erst am Ende des Pfades vollzieht, sind trotzdem noch der größte Teil des Pfades mit der Ground-Truth Strecke übereinstimmig. Daher ist es zudem interessant, wie viele Pfade ausgewählt werden, welche exakt der Ground-Truth-Strecke entsprechen, also einen IoU-Wert von genau 1.0 besitzen [4.4].

4.2 Evaluation und Fazit

Insgesamt kann zusammengefasst werden, dass das Neuronale Netz ungefähr der Leistung des CLC-Algorithmus entspricht. Das Neuronale Netz wählt zwar in wenigen Situationen

	1.0 IoU	0.99 IoU	0.98 IoU
CLC	331	141	64
Ranking-NN	368	117	55

Tabelle 4.4: Das Neuronale Netz wählt mehr Bahnen aus, welche exakt der Ground-Truth-Bahn entsprechen. Dagegen wählt der CLC-Algorithmus mehr Bahnen aus, die nahezu der Ground-Truth-Bahn entsprechen.

mehr kritische Pfade aus, als der Algorithmus [4.1, 4.2], erzielt dafür aber durchschnittlich bessere Ergebnisse [4.3] und wählt mehr Bahnen aus, welche exakt der Ground-Truth-Bahn entsprechen [4.4]. Die guten Ergebnisse des Neuronalen Netzes zeigen auch, dass der Trainingsprozess und die Architektur des Netzes gut gewählt wurden. Zudem zeigen die Ergebnisse eine gute Entscheidungsgrundlage für die zukünftige Integrierung des Neuronalen Netzes in die Fahrzeugarchitektur und die Motivation für weitere Entwicklungen mithilfe von modernen Machine Learning Ansätzen im Formula Student Bereich. Um eine ausführliche Bewertung der Ansätze zu gewährleisten, müssten noch weitere Metriken in Betracht gezogen werden.

4.3 Weiterführende Arbeit

Um bessere Ergebnisse zu erzielen, könnte das Modell zunächst auf einem größeren Datensatz trainiert werden. In dem größeren Trainingsdatensatz müssten auch mehrere schwierige Streckenabschnitte enthalten sein um ein Abkürzen der Strecke künstlich zu provuzieren. Zusätzlich wird mehr Forschung im Zusammenhang und Aufbau einer Fahrbahn benötigt. Da aktuell nur acht geometrische Merkmale der Fahrbahn als Entscheidungsgrundlage dienen, würden weitere Zusammenhänge der beiden Boundaries oder spezifische Eigenschaften der Mittellinie enorm zu besseren Ergebnissen motivieren. Beispielsweise könnten die Winkel zweier korrespondierenden Segmente der Fahrbahn untersucht werden. Außerdem könnte die Architektur des Neuronalen Netzes angepasst oder neu konzeptioniert werden. Zusätzliche Schichten mit *Pooling-Layern* könnten über die Zeit besonders wichtige Features extrahieren und stärker gewichten. Die *Listwise-Ranking* Methode ist zwar komplizierter zu implementieren, verspricht aber im allgemeinen gute Ergebnisse [3], da moderne Ranking-Methoden fast ausschließlich auf dieser Methode aufbauen [4]. Schlussendlich könnte man testen, ob es mittlerweile Implementierungen für ähnliche Aufgaben auf diesem Gebiet gibt. Bereits trainierte und vielversprechende Ranking-Modelle für ähnliche binäre Klassifikationsaufgaben gibt es grundsätzlich viele, diese müssten dann hinsichtlich der Eingabe angepasst oder erweitert werden. Um eine wissenschaftliche Aussage über den Vergleich des Algorithmus und des Neuronalen Netzes zu treffen, müssen die Ergebnisse statistisch auf signifikante Unterschiede untersucht werden.
Da der Bahnplanungsalgorithmus sehr oft iteriert, könnte getestet werden, ob ein besonders guter Pfad über mehrere Iterationen hinweg beibehalten werden könnte. Falls also ein Pfad mit $IoU = 1.0$ gefunden wird, könnte dieser mit weiterer Untersuchung der Pfadlänge auch über eine längere Zeit beibehalten werden. Falls in dieser Zeit ein neuer Pfad mit $IoU = 1.0$ gefunden wird, könnte dieser weiter benutzt werden. Falls kein korrekter Pfad gefunden wird, muss der nächst bessere verwendet werden.

Literaturverzeichnis

[1] Kevin Buchin, Maike Buchin, Wouter Meulemans, and Bettina Speckmann. Locally correct fréchet matchings. In Leah Epstein and Paolo Ferragina, editors, *Algorithms – ESA 2012*, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. S. 229–240.

[2] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. *Proceedings of the 22nd international conference on Machine learning*, 2005. S. 89–96.

[3] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. *Proceedings of the 24th international conference on Machine learning*, 2007. S. 129–136.

[4] Xishuang Dong, Xiaodong Chen, Yi Guan, Zhiming Yu, and Sheng Li. An overview of learning to rank for information retrieval. *2009 WRI World Congress on Computer Science and Information Engineering, CSIE 2009*, März 2009. S. 600–606.

[5] Formula-Student-Germany. Formula student rules 2024 v1.1. https://www.formulastudent.de/fsg/rules/, 2023. [Online; aufgerufen am 28. Dezember 2023].

[6] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly, 2. edition edition, 2019.

[7] Georg Tanzmeister, Martin Friedl, Andreas Lawitzky, Dirk Wollherr, and Martin Buss. Road course estimation in unknown, structured environments. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, 2013. S. 630–635.