HAW HAMBURG

BACHELOR THESIS
Sofia Halbach

Implementierung und Vergleich verschiedener Algorithmen zur Rundenzeitoptimierung eines autonomen Formula Student Fahrzeugs

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Sofia Halbach

Implementierung und Vergleich verschiedener Algorithmen zur Rundenzeitoptimierung eines autonomen Formula Student Fahrzeugs

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Peer Stelldinger
Zweitgutachter: Prof. Dr. Christian Lins

Eingereicht am: 18. September 2025

Sofia Halbach

Thema der Arbeit

Implementierung und Vergleich verschiedener Algorithmen zur Rundenzeitoptimierung eines autonomen Formula Student Fahrzeugs

Stichworte

Rundenzeitoptimierung, Formula Student, Trajektorie mit geringster Krümmung, Geschwindigkeitsplanung, Evolutionäre Algorithmen, Formula Student Driverless Simulator

Kurzzusammenfassung

In dieser Arbeit werden verschiedene Möglichkeiten zur Rundenzeitoptimierung bei der autonomen Disziplin Trackdrive betrachtet und miteinander verglichen.
Bei dem ersten Ansatz wird zunächst die Trajektorie mit der geringsten Krümmung ermittelt. Im Anschluss werden die Geschwindigkeiten unter der Berücksichtigung der Kurvenradien und Fahrzeugspezifikationen für jeden Punkt auf der Strecke berechnet.
Der zweite Ansatz nimmt die Optimierung mittels eines genetischen Algorithmus vor. Die Evaluierung der Individuen erfolgt über den Formula Student Driverless Simulator.
Zuletzt wird ein hybrider Ansatz betrachtet, bei dem die Ergebnisse aus der Berechnung mithilfe des genetischen Algorithmus weiter entwickelt werden.
Die Ergebnisse zeigen, dass mit dem hybriden Ansatz die besten und zuverlässigsten Rundenzeiten erzielt werden können. Des weiteren zeigt sich, dass der genetische Algorithmus in Kombination mit der verwendeten Echtzeitsimulation aufgrund der langen Trainingsdauer nicht wettbewerbstauglich ist. Bezüglich der Rechenzeit ist somit der Ansatz mit der minimalen Krümmung zu bevorzugen.

Sofia Halbach

Title of Thesis

Implementation and Comparison of different Algorithms to Improve the Lap Time of an Autonomous Formula Student Race Car

Keywords

Lap time optimization, Formula Student, minimum curvature trajectoy, velocity planning, evolutionary algorithms, Formula Student Driverless Simulator

Abstract

In this bachelor's thesis different options for laptime optimization are compared for the autonomous trackdrive discipline.
The first approach determines the minimum curvature trajectory and then plans the velocities for every point on the trajectory by taking into consideration the radii of the curves and the specifications of the race car.
The second approach optimizes the trajectory and velocities with a genetic algorithm. The evaluation of the individuals is done within the Formula Student Driverless Simulator.
Lastly, a hybrid approach is considered that takes the results from the first approach and evolves them further with the genetic algorithm.
The results reveal that the hybrid approach achieves the best and most reliables lap times. Furthermore, it is shown that the genetic algorithm in combination with the real time simulation is not suitable for the competition due to the long training period. Regarding the computing time of the different algortihms the minimum curvature approach is preferable.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Autonomes Fahren stellt eine der zentralen Entwicklungen im Bereich der Mobilität dar und gewinnt sowohl in der Forschung als auch in der Industrie zunehmend an Bedeutung. Bereits heute sind Fahrerassistenzsysteme wie Abstandsregeltempomaten oder automatische Notbremsfunktionen in zahlreichen Serienfahrzeugen etabliert und bilden damit einen wichtigen Zwischenschritt auf dem Weg zum fahrerlosen Fahren.
Seit 2021 sind in Deutschland die rechtlichen Rahmenbedingungen für den Betrieb von Fahrzeugen nach SAE-Level 4 geschaffen [4] und weltweit werden in über 50 Ländern Konzepte für hoch- und vollautomatisiertes Fahren erprobt (Beispiele siehe [17]).
Durch das wachsende Interesse in der Industrie steigt auch die Nachfrage nach qualifizierten Ingenieur*innen und Software-Entwickler*innen.
Eine praxisnahe Möglichkeit, die benötigten Kompetenzen bereits während des Studiums zu entwickeln, bietet der Wettbewerb Formula Student.

Formula Student
Formula Student (FS) ist ein internationaler Konstuktionswettbewerb für angehende Ingenieur*innen. Ziel des Wettbewerbs ist es, innerhalb eines Jahres einen Rennwagen zu entwerfen und zu fertigen und damit an den Wettkämpfen teilzunehmen. Es gibt weltweit insgesamt 26 verschiedene Austragsorte, 18 davon in Europa, die als Formula Student Events bezeichent werden. Auf den Events treten Teams aus aller Welt in verschiedenen Vorträgen, wie auch fahrdynamischen Disziplinen gegeneinander an.
Formula Student Germany (FSG) ist mit über 3000 Teilnehmenden aus 22 Ländern (2025) das größte Event. Die Veranstalter*innen von FSG geben das allgemeine Regelwerk heraus [7] und veröffentlichen darüber hinaus auch eventspezifische Regeln [8].
Seit dem Jahre 2017 gibt es in FSG zusätzlich die Möglichkeit, an vier fahrerlosen Rennen teilzunehmen. Die meisten Punkt bringt die Disziplin Trackdrive, bei der ein Rennwagen 10 Runden auf einer zuvor unbekannten Rundstrecke fährt.

HAWKS Racing e.V.
HAWKS Racing ist das Formula Student Team der HAW Hamburg und wurde 2003 gegründet. Das interdisziplinäre Team besteht aus ca. 50 Mitgliedern.
Aktuell wird das autonome System (AS) am 19. Rennwagen (H19, „NOVA“) weiterentwickelt und getestet. Es ist das insgesamt dritte Fahrzeug mit elektrischem Antrieb und das zweite, das in der Lage ist autonom zu fahren.

Abbildung 1.1: NOVA, bei einem autonomen Fahrversuch in FSG 2024, ©FSG Ernst

NOVA verfügt über zahlreiche Sensoren zur Erfassung der Strecke und Fahrbewegung, darunter ein LiDAR Sensor, eine Kamera und ein Ground-Speed-Sensor. Die Steuerung des Fahrzeugs nimmt das autonome System über einen Lenkmotor, die Ansteuerung des Inverters und ein Notbremssystem vor. Eine Möglichkeit kontrolliert auf eine Zielgeschwindigkeit herunter zu bremsen besitzt das Fahrzeug nicht.

1.1 Zielsetzung

Um die eigene Punktzahl beim Trackdrive in FSG zu maximieren, muss der Rennwagen möglichst schnell die vorgegebene Strecke abfahren. Ziel dieser Arbeit ist es, zu ermitteln, welcher Algorithmus für die Optimierung der Rundenzeit in diesem Anwendungsfall am

geeignetsten ist.
Die Rundenzeit wird von zwei hauptsächlichen Faktoren beeinflusst: Der Wahl der Ideallinie, also der Trajektorie, die von dem Fahrzeug abgefahren wird, und der Geschwindigkeitsplanung für jeden Streckenabschnitt. Bei dem daraus resultierenden Optimierungsproblem muss infolgedessen sowohl die Streckengeometrie als auch die Fahrzeugdynamik beachtet werden.

1.2 Eingrenzung des Themas

Auswahl der Algorithmen
Ein weit verbreiteter Ansatz in der Formula Student ist die Bestimmung der Trajektorie mit der geringsten Krümmung für eine bestimmte Strecke (siehe beispielhaft KA RaceIng [11], FSB Racing [13], LiU [10]).
Vorteilhaft an diesem Vorgehen ist, dass die Minimierung der Krümmung die Bildung langer Geraden mit sich führt, die hohe Beschleunigungen ermöglichen. Des weiteren verliert der Rennwagen durch die geringe Anzahl an Lenkbewegungen weniger Geschwindigkeit in den Kurven, als es z.B. bei der kürzesten Strecke der Fall wäre.
In Abschnitt 2.6 ist eine Möglichkeit zur Definition von Trajetkorien beschrieben. Dabei werden die Wegpunkte, die in ihrer Gesamtheit die Trajektorie bilden, mithilfe eines Parameters α definiert. Dieser Parameter bestimmt die Position des Wegpunktes zwischen der linken und rechten Streckenbegrenzung und sorgt dafür, dass die Soll-Trajektorie automatisch auf der Strecke liegt.
Auch die Tatsache, dass es sich bei der Berechnung der Trajektorie mit der geringsten Krümmung um ein quadratisches Problem handelt, welches effizient mittels vorgefertigter Solver gelöst werden kann, macht es zu einer attraktiven Teillösung.
Die Zielgeschwindigkeiten zu den einzelnen Wegpunkten müssen bei diesem Vorgehen separat ermittelt werden. Dies geschieht oft durch die Berechnung der Höchstgeschwindigkeiten, die das spezifische Auto bei einem bestimmten Kurvenradius aufbringen kann, ohne von der Strecke abzukommen.

In dem Bereich der künstlichen Intelligenz gibt es verschiedene Algorithmen, die bei der Rundenzeitoptimierung anwendbar sind.
Überwachtes Lernen in Kombination mit neuronalen Netzen bietet das Potenzial, die intelligente Entscheidungsfindung von Rennfahrer*innen nachzuahmen. Dieser Ansatz

erfordert eine große Anzahl an gelabelten Trainingsdaten, die das zu lernende Verhalten widerspiegeln. In diesem Fall bedeutet das, dass Daten von verschiedenen Fahrsituationen und Streckenabschnitten mit der Zielfahrweise benötigt werden. Dafür können einerseits aufgenommene Daten von realen Testfahrten mit Fahrer*in oder künstlich generierte Daten verwendet werden.
Die realen Testfahrten bergen zwei Probleme. Einerseits ist die Datenaufnahme mit sehr viel zeitlichem und personellen Aufwand verbunden. Andererseits lernt das neuronale Netz lediglich die Fahrweise mit Fahrer*in und kann die veränderte Verhaltensweise des Fahrzeugs ohne Fahrer*in nicht miteinbeziehen. Das unterschiedliche Fahrverhalten ergibt sich durch das geringere Gewicht und die Abwesenheit einer Möglichkeit kontrolliert zu Bremsen. Die künstliche Generierung ist je nach Grad der Automatisierung weniger zeitaufwendig, bietet jedoch die Hürde die Trainingsdaten realistisch genug zu gestalten. Dieses Thema lässt wiederum sehr viel Raum für Diskussionen.
Ein weiterer Nachteil in diesem Fall ist, dass überwachtes Lernen auf die Nachahmung von bestehenden Lösungen begrenzt ist. Das neuronale Netz kann somit nie besser werden als sein (menschliches) Vorbild. Die Entwicklung neuer, kreativer Strategien ist aufgrund der Komplexität des Rennfahrens jedoch von großem Interesse.

Evolutionäre Algorithmen (EAs) bieten sich in diesem Anwendungsfall an, da kein Ergebnis oder eine Lösungsstrategie vorgegeben werden muss. Es ist lediglich eine Möglichkeit erforderlich, die gegebenen Lösungskandidaten zu bewerten. Die Bewertung kann in diesem Fall mit Hilfe von objektiven Kriterien wie der Rundenzeit vorgenommen werden. Dadurch wird der Lösungsraum nicht auf eine Lösungsstrategie wie zum Beispiel die Krümmungsminimierung begrenzt, obwohl möglicherweise Optimierungsstrategien existieren, mit denen bessere Ergebnisse erzielt werden können.
Des weiteren sind EAs bei Problemen geeignet, bei denen viele Parameter (Positionen der Wegpunkte und Geschwindigkeiten) aufeinander abgestimmt werden müssen und ein Ausprobieren der Lösungen nicht praktikabel ist.

Eine weitere Möglichkeit neue Fahrstrategien zu entwickeln, bietet Reinforcement Learning. Bei passender Modellierung der Zustände und Aktionen kann der Agent auch hier aufgrund der Bewertung seiner Handlungen lernen ein Rennauto zu fahren wie es zum Beispiel in [5] der Fall ist. Dieses Verfahren birgt jedoch die Gefahr, dass der Zustandsraum sehr groß werden kann, was den Lernprozess sehr langwierig macht.

Nichts desto trotz handelt es sich hierbei um einen gute Option.

In dieser Arbeit werden ein Ansatz zur numerischen Berechnung mittels der Minimierung der Krümmung einer Trajektorie, wie auch genetische Algorithmen zur Lösung des Problems der Rundenzeitoptimierung untersucht und die Ergebnisse miteinander verglichen.

Abgrenzung des Themas
Zur Evaluation und Ergebniserhebung innerhalb dieser wissenschaftlichen Auseinandersetzung wird der Formula Student Driverless Simulator (FSDS) verwendet, eine Open Source Simulation, auf die in Abschnitt 2.3 genauer eingegangen wird. Die Entwicklung einer aussagekräftigen Simulation, die akkurat die Fahrzeugdynamik des HAWKS-Fahrzeugs widerspiegelt ist somit nicht Teil der Arbeit.

Ebenso werden die Ergebnisse nicht mit dem realen Fahrzeug validiert, wenngleich alle Vorkehrungen getroffen werden, um die Software mit dem bestehenden System kompatibel zu machen und dem Regelwerk zu entsprechen.

In dieser Arbeit geht es um die Entwicklung eines Softwaresystems, welches die verschiedenen Optimierungsansätze umsetzt und um den Vergleich der erzielten Ergebnisse. Am Ende wird eine Einschätzung getroffen, welcher Ansatz für den gegebenen Anwendungsfall am geeignetsten ist.

1.3 Gliederung der Arbeit

Die vorliegende Arbeit ist in fünf Kapitel unterteilt, die logisch aufeinander aufbauen und gemeinsam zu der Beantwortung der zentralen Forschungsfrage beitragen.

Kapitel 1 - Einleitung
Im ersten Kapitel wird das Thema eingeführt und die Problemstellung erläutert. Dabei wird aufgezeigt, weshalb das Thema relevant ist und welche konkrete Fragestellung im Mittelpunkt der Untersuchung steht.

Kapitel 2 - Grundlagen und Rahmenbedingungen
In diesem Kapitel wird zunächst der Anwendungsfall konkretisiert. Darüber hinaus werden die theoretischen Grundlagen erläutert, die für das Verständnis der Thematik von Bedeutung sind.

Kapitel 3 - Entwurf und Implementierung
Das nächste Kapitel beschäftigt sich mit den Designentscheidungen und der Umsetzung der einzelnen Ansätze.

Kapitel 4 - Ergebnisse
Im Anschluss werden in Kapitel 4 die verwendeten Teststrecken vorgestellt und die Ergebnisse präsentiert und diskutiert.

Kapitel 5 - Zusammenfassung und Ausblick
Abschließend werden die gewonnenen Erkenntnisse zusammengefasst und die nächsten Schritte zur Validierung und Weiterentwicklung der Ergebnisse beschrieben.

2 Grundlagen und Rahmenbedingungen

Dieses Kapitel umfasst die Grundlagen, die für das Verständnis der darauffolgenden Kapitel relevant sind.
Zunächst wird der Anwendungsfall konkretisiert, indem die Rahmenbedingungen beleuchtet werden, die sich durch den Wettbewerb und das bestehende autonome System ergeben. Im Anschluss werden die verwendeten Fremdsysteme vorgestellt und die theoretischen Grundlagen erklärt, auf denen die restliche Arbeit aufbaut.

2.1 Trackdrive

Trackdrive ist eine fahrdynamische Disziplin des FS Wettbewerbs für driverless (DV) Fahrzeuge. Beim Trackdrive fährt der Rennwagen 10 Runden auf einer zuvor unbekannten Rundstrecke. Dabei müssen alle nötigen Berechnungen für die Streckenerkennung, Kartografierung, Trajektorienbildung und Ansteuerung der Aktorik auf der Hardware des Rennwagens geschehen.
Die Strecke ist, wie in Abbildung 2.1 dargestellt, mit Verkehrshütchen markiert. Die Hütchen links der Strecke sind blau, die rechts der Strecke sind gelb.

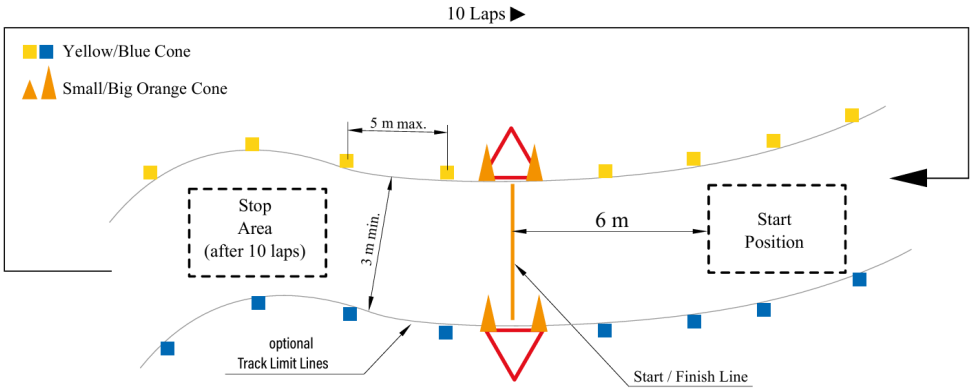


Abbildung 2.1: Trackdrive Streckenaufbau (aus [8], figure 4)

Die Autos treten auf Zeit gegeneinander an. Es gibt dementsprechend kein Kopf-an-Kopf-Rennen. Des Weiteren sitzt während der autonomen Fahrt kein Fahrer im Auto.
Der Trackdrive ist mit 200 möglichen Punkten die am höchsten gewichtete autonome Disziplin. Pro gefahrene Runde erhält das Team 5 Punkte. Wird das Rennen erfolgreich beendet, gibt es abhängig von der Gesamtzeit und der Wettbewerbs-Bestzeit zusätzliche Punkte. Die Formel zur Berechnung der Punkte ist in 2.1 dargestellt (siehe [7] S. 131).

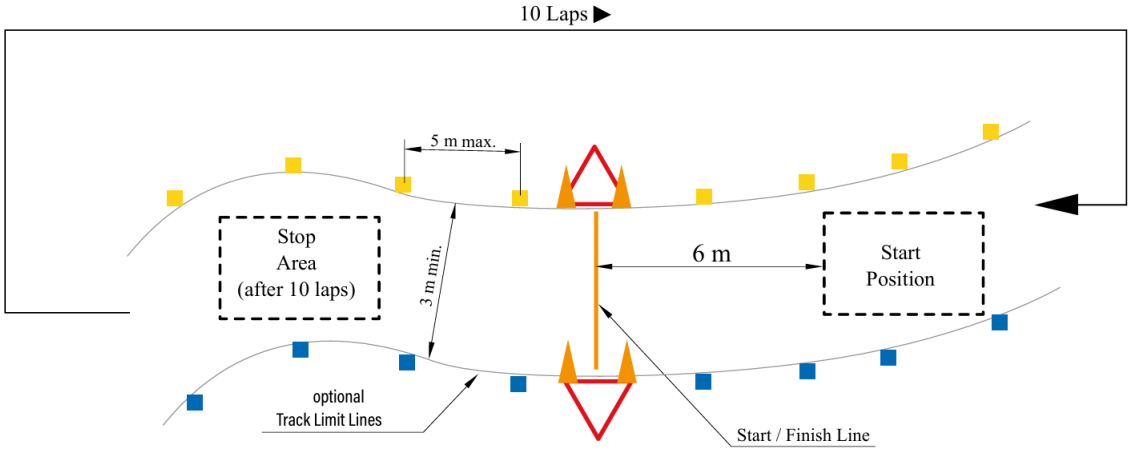
$$\begin{aligned} Score &= 0.75 * 200P * (\frac{T_{max}}{T_{team}} - 1) \\ T_{max} &= 2 * Gesamtzeit\ inklusive\ Zeitstrafen\ des\ schnellsten\ Teams \\ T_{team} &= Gesamtzeit\ inklusive\ Zeitstrafen\ des\ eigenen\ Teams \end{aligned} \tag{2.1}$$

Zeitstrafen gibt es jeweils für umgefahrene Hütchen und das zwischenzeitliche Verlassen der Strecke mit allen vier Rädern. Ein umgefahrenes Hütchen wird Down or Out (DOO) genannt und gibt zwei Sekunden Zeitstrafen. Das Fahren außerhalb der Strecke wird als Off-Course (OC) bezeichnet und gibt pro Vorkommen zehn Sekunden Strafe. Bleibt das Auto nach Abschluss der 10. Runde nicht ordnungsgemäß stehen, gibt es 50 Punkte Abzug. Das Stehenbleiben auf der Strecke für eine längere Zeitspanne als 30 s führt zu einer Disqualifizierung. Ebenso werden Teams disqualifiziert, deren Fahrzeuge die vorgegebene durchschnittliche Mindestgeschwindigkeit von 2,5 m/s in den ersten 3 Runden und 3,5 m/s in den darauffolgenden Runden unterschreitet.
Fallen während der technischen Inspektionen vor oder nach dem Lauf Regelverletzungen auf, darf das Auto entweder nicht antreten oder der Lauf wird rückwirkend annulliert.
Es gibt pro Team nur einen Versuch für diese Disziplin.
Im Notfall kann das Auto ferngesteuert über das Notbremssystem angehalten werden.

Die Regeln für die Trackdrive Disziplin ([7] S. 131f., [8] S. 15ff.) sind in Tabelle 2.1 zusammengefasst.

Trackdrive	**Punkte:** 200

Streckenverlauf [8]:



Bewertung:

$Score = 0.75 * 200P * (\frac{T_{max}}{T_{team}} - 1)$

T_{max} = 2 * Zeit + Zeitstrafen des schnellsten Teams

T_{team} = Zeit + Zeitstrafen des eigenen Teams

$Score$ += 5 Punkte für jede gefahrene Runde

Ablauf:

- Mit Hütchen markierter Rundkurs
- 10 Runden
- 1 Versuch
- Strecke zuvor unbekannt

Strafen:

- DOO: 2 s (Hütchen umgefahren)
- Off-Course: 10 s (Alle 4 Reifen außerhalb der Strecke)
- Unsafe Stop: -50 Punkte (Halten außerhalb der Stop Area)

Tabelle 2.1: Trackdrive Zusammenfassung

2.2 Das autonome System

Ziel dieses Abschnitts ist es einen Überblick über das fahrerlose System des Rennwagens NOVA zu verschaffen und aufzuzeigen, wie sich diese Arbeit in das bestehende autonome System eingliedert.

Das Programm zur Rundenzeitoptimierung findet Anwendung, sobald die Strecke kartografiert ist. Dies ist ab der zweiten Runde der Trackdrive Disziplin der Fall.
Die vorausgehenden Verarbeitungsschritte sind die Erkennung der Hütchen mittels des LiDAR Sensors, die Ermittelung der Mittellinie, die Lokalisierung des Fahrzeugs auf der Strecke und die Kartografierung der Strecke mit Hilfe des SLAM-Algorithmus. Über die verschiedene Reflektanz der Hütchen-Farben ist außerdem eine Zuordnung möglich, ob sich die Hütchen links (blau) oder rechts (gelb) der Strecke befinden.

Das in dieser Arbeit beschriebene System erhält als Input eine Karte der Strecke bestehend aus Hütchen-Koordinaten und die Einteilung in linke und rechte Hütchen. Aufgrund dieser Eingabedaten wird eine Liste an Wegpunkten und die Zielgeschwindigkeiten zu jedem dieser Wegpunkte ermittelt. Die Gesamtheit aller Wegpunkte bildet die Trajektorie, die das Auto entlangfahren soll. Pro Hütchenpaar gibt es einen Wegpunkt, der entsprechend zwischen dem linken und rechten Hütchen liegt.

Das autonome System hat nun die Aufgabe die Aktorik anzusteuern und ausgehend von den Zielgeschwindigkeiten und Regelpunkten die entsprechenden Drehmoment- und Lenkwinkel-Anforderungen zu stellen. Die Längsregelung erfolgt über einen PI-Regler und die Querregelung über einen Pure-Pursuit-Regler.
Abschließend übernimmt das AS die Verwaltung der Mission, leitet nach Vollendung der zehn Runden das Bremsmanöver ein und bringt das Auto in den sicheren Endzustand.
Tiefgreifendere Details über das autonome System sind in der Masterarbeit von Michel Wegner [14] nachzulesen.

Darüber hinaus bietet das AS eine Schnittstelle zum Formula Student Driverless Simulator, der in Abschnitt 2.3 näher beschriebenen wird.

2.3 Die Simulationsumgebung

In dieser Bachelor Arbeit wird mit dem Formula Student Driverless Simulator (FSDS, [6]) gearbeitet. Es ist eine auf Unreal Engine 4 basierende Open Source Simulation, die von Mitgliedern der Formula Student Comunity entwickelt wurde.
Die FSDS bietet eine Möglichkeit die driverless Software eines Teams in einer virtuellen Welt im voraus zu testen. Hierfür emuliert die FSDS den Output der Sensoren z.B. eines LiDARs und schickt ihn an das verbundene autonome System. Die driverless Software des Teams verarbeitet diese Daten und schickt die Fahrbefehle anschließend an die Simulation.
Es stehen einige Optionen der Individualisierung des Rennwagens zur Verfügung wie die Art, Anzahl und Position der Sensorik, nicht aber der genauen Fahrzeugeigenschaften wie Gewicht, Radstand etc.

Die FSDS wird in dieser Arbeit insbesondere bei der Evaluierung der Lösungskandidaten des genetischen Algorithmus verwendet. Die Simulation liefert die Position des Fahrzeugs und erhält die gewünschte Geschwindigkeit und den Lenkwinkel. Nach Abschluss einer Runde auf der Strecke gibt die Simulation die Rundenzeit und Anzahl der umgefahrenen Hütchen bekannt, welche in die Bewertung mit einfließen. Es wird auch hier davon ausgegangen, dass die Hütchenkoordinaten bekannt sind und die Informationen aus der Lokalisierung zur Verfügung stehen. Das konkrete Zusammenspiel aus FSDS und DV Software wird in Abschnitt 3.4.2 thematisiert.

2.4 Random Track Generator

Um ausreichend Teststrecken zur Verfügung zu haben, wurde der Random Track Generator (RTG) von Mees van Löben Sels (siehe [12]) verwendet. Mithilfe des RTGs lassen sich zufällige, mit Hütchen abgesteckte Strecken unterschiedlicher Größe und Form erstellen. Dabei wird sichergestellt, dass die minimale Streckenbreite eingehalten wird und die Hütchen einen geschlossenen Rundkurs bilden. Die minimale Streckenbreite (3 m) und der maximale Abstand zwischen den Hütchen (5 m) wurden gemäß der Anforderungen aus dem Regelwerk [8] gewählt.
Die Hütchen-Koordinaten der generierten Strecke werden nach Farben unterteilt in eine csv-Datei geschrieben. Das Format der Datei ist zum FSDS kompatibel.

2.5 ROS und das Publisher-Subscriber Pattern

Sowohl das autonome System von HAWKS als auch die FSDS arbeiten mit dem Robot Operating System (ROS). ROS ist eine Sammlung von Open Source Libraries und Frameworks und dient als Middleware für Robotik-Anwendungen. Der Nachrichtenaustausch in ROS funktioniert über das sogenannte Publisher-Subscriber-Pattern.
Hierbei werden die Nachrichten aufgrund ihres Inhalts auf einem bestimmten Topic veröffentlicht (published). Kommunikationsteilnehmer (Nodes), die an diesen Nachrichten interessiert sind, können dieses Topic abonnieren (subscriben) und erhalten damit alle Nachrichten zu diesem Thema (Topic).
Dieses Pattern findet aufgrund seiner losen Kopplung, hohen Flexibilität und Skalierbarkeit viel Anwendung in verteilten Systemen.

2.6 Numerische Berechnung

In dem Paper „Race Driver Model“ [3] ist beschrieben, dass es zwei grundlegende Ansätze zur Berechnung einer Ideallinie gibt. Der erste Ansatz ist die Minimierung der gefahrenen Strecke und der zweite beschäftigt sich mit der Minimierung der Krümmung einer Trajektorie. Beiden Varianten geht voraus, dass die Strecke zuvor in Segmente unterteilt wird. Jeweils am Ende jedes Segments wird ein Wegpunkt P_i mit den Koordinaten x_i und y_i gewählt, der zwischen der rechten und linken Streckenbegrenzung $P_{r,i}$ mit $x_{r,i}$, $y_{r,i}$ und $P_{l,i}$ mit $x_{l,i}$, $y_{l,i}$ liegt. Die Position eines Wegpunkts kann wie folgt über den Parameter α_i bestimmt werden. Dabei gilt $0 \leqslant \alpha_i \geqslant 1$.

$$\begin{aligned}\vec{\mathbf{P}}_i &= x_i\vec{\mathbf{i}} + y_i\vec{\mathbf{j}} \\ &= [x_{r,i} + \alpha_i(x_{l,i} - x_{r,i})]\vec{\mathbf{i}} + [y_{r,i} + \alpha_i(y_{l,i} - y_{r,i})]\vec{\mathbf{j}} \\ &= [x_{r,i} + \alpha_i\Delta x_i]\vec{\mathbf{i}} + [y_{r,i} + \alpha_i\Delta y_i]\vec{\mathbf{j}}\end{aligned} \tag{2.2}$$

Die Trajektorie erhält man, indem man alle Wegpunkte $\vec{\mathbf{P}}_i$ über Funktionen verbindet. Die Identifizierung der kürzesten Trajektorie oder der Trajektorie mit der geringsten Krümmung kann auf die geeignete Wahl der Parameter α_1 bis α_n heruntergebrochen werden.

2.6.1 Berechnung der Strecke einer Trajektorie

Um die Länge einer Trajektorie zu ermitteln, werden die Wegpunkte über lineare Funktionen verbunden und ihre Längen addiert. Die Länge eines Segments ds_i kann wie folgt berechnet werden:

$$\begin{aligned} ds_i &= \vec{\mathbf{P}}_{i+1} - \vec{\mathbf{P}}_i = \Delta P_{x,i}\vec{\mathbf{i}} + \Delta P_{y,i}\vec{\mathbf{i}} \\ \Delta P_{x,i} &= x_{r,i+1} - x_{r,i} + \alpha_{i+1}\Delta x_{i+1} - \alpha_i \Delta x_i \\ \Delta P_{y,i} &= y_{r,i+1} - y_{r,i} + \alpha_{i+1}\Delta y_{i+1} - \alpha_i \Delta y_i \end{aligned} \tag{2.3}$$

Die quadrierte Gesamtlänge ist somit:

$$S^2 = \sum_{i=1}^{n} \Delta P_{x,i}^T \Delta P_{x,i} + \Delta P_{y,i}^T \Delta P_{y,i} \tag{2.4}$$

Gesucht wird damit, eine Kombination von α_1 bis α_n-Werten, die eine Trajektorie mit möglichst geringer Streckenlänge bildet. Um zusätzlich einen geschlossenen Rundkurs zu erhalten, muss außerdem $\alpha_1 = \alpha_n$ gelten (vorausgesetzt $P_{r,1} = P_{r,n}$ und $P_{l,1} = P_{l,n}$).

Über die Formel 2.4 lässt sich die quadrierte Gesamtlänge einer Trajektorie berechnen. Dies ist der Wert, den es bei der Minimierung der Strecke zu optimieren gilt. Die Optimierung selber wird über einen separaten Solver vorgenommen.

2.6.2 Berechnung der Krümmung einer Trajektorie

In diesem Optimierungsfall werden benachbarte Wegpunkte mit geschlossenen natürlichen kubischen Splines verbunden. Der Verlauf eines Streckensegments wird über die folgenden Funktionen definiert:

$$\begin{aligned} x_i(t) &= a_{i,x} + b_{i,x}t + c_{i,x}t^2 + d_{i,x}t^3 \\ y_i(t) &= a_{i,y} + b_{i,y}t + c_{i,y}t^2 + d_{i,y}t^3 \\ t(s) &= \frac{s - s_{i0}}{ds_i} \end{aligned} \tag{2.5}$$

Die Funktion $t(s)$ stellt hierbei das Verhältnis zwischen der zurückgelegten Strecke entlang der Kurve s seit dem Beginn des Streckenabschnitts s_{i0} und der Segmentlänge ds_i dar. Die Parameter a, b, c und d sind die Koeffizienten der kubischen Funktionen.

Die quadrierte Krümmung des Streckenabschnitts wird durch die zweite Ableitung der Funktionen berechnet:

$$\begin{aligned}\Gamma^2 &= \left(\frac{d^2x_i(s)}{ds^2}\right)^2 + \left(\frac{d^2y_i(s)}{ds^2}\right)^2 \\ &= \left(\frac{d^2x_i(t)}{dt^2}\left(\frac{dt(s)}{ds}\right)^2\right)^2 + \left(\frac{d^2y_i(t)}{dt^2}\left(\frac{dt(s)}{ds}\right)^2\right)^2 \\ &= \left(\frac{dt(s)}{ds}\right)^4 \left(\left(\frac{d^2x_i(t)}{dt^2}\right)^2 + \left(\frac{d^2y_i(t)}{dt^2}\right)^2\right)\end{aligned} \tag{2.6}$$

Formel 2.6 in der Lagrange Schreibweise:

$$\begin{aligned}\Gamma^2 &= x_i''(s)^2 + y_i''(s)^2 \\ &= (x_i''(t) * t'(s)^2)^2 + (y_i''(t) * t'(s)^2)^2 \\ &= t'(s)^4 * (x_i''(t)^2 + y_i''(t)^2)\end{aligned} \tag{2.7}$$

Unter der Annahme, dass alle Streckensegmente entlang der Mittellinie die gleiche Länge ds^* haben und somit $t'(s) = \frac{1}{ds^*}$ ist, lässt sich die Formel 2.6 wie folgt vereinfachen:

$$\Gamma^2 = \left(\frac{1}{ds^*}\right)^4 \left(\left(\frac{d^2x_i(t)}{dt^2}\right)^2 + \left(\frac{d^2y_i(t)}{dt^2}\right)^2\right) \tag{2.8}$$

Die gesamte Krümmung der Strecke lässt sich nun folgendermaßen berechnen:

$$\Gamma^2 = \sum_{i=1}^{n} \left(\frac{d^2x_i(t)}{dt^2}\right)^2 + \left(\frac{d^2y_i(t)}{dt^2}\right)^2 \tag{2.9}$$

Die Krümmung innerhalb eines Segments ist an dem Punkt P_i am größten. An diesem Punkt gilt $t = 0$. Addiert man bei der Berechnung der Trajektorien-Gesamtkrümmung nur die Krümmungen zu diesen Punkten, lässt sie sich für x wie folgt darstellen:

$$\Gamma_x = \left.\frac{d^2\bar{\mathbf{x}}(t)}{dt^2}\right|_{t=0} = [\mathbf{D}]\bar{\mathbf{x}} \tag{2.10}$$

Die Matrix [**D**] wird verwendet, um die zweite Ableitung von $x(t)$ zu berechnen. Diese lautet bei $t = 0$ näherungsweise:

$$\begin{aligned} x''(t) &= 2c_{i,x} + 3d_{i,x} \cdot t \\ &= 2c_{i,x} \\ &\approx 2x_i - x_{i-1} - x_{i+1} \end{aligned} \tag{2.11}$$

Die [**D**]-Matrix ist eine quadratische n x n Matrix, wobei n der Anzahl an Wegpunkten entspricht. Entlang der Hauptdiagonale befindet sich der Wert 2 und entlang der beiden Nebendiagonalen (oberhalb und unterhalb der Hauptdiagonale) steht der Wert -1. Die Werte $D_{n,1}$ und $D_{1,n}$ sind ebenfalls -1 und sorgen dafür, dass bei der Spline-Bildung ein Rundkurs entsteht. Bei $n = 4$ sieht die [**D**]-Matrix beispielsweise so aus:

$$[\mathbf{D}] = \begin{bmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix} \tag{2.12}$$

Der Vektor $\bar{\mathbf{x}}$ enthält die x-Koordinaten aller Wegpunkte. Er hängt somit linear von Vektor $\bar{\alpha}$ ab, der alle α-Werte enthält. Der Vektor $\bar{\mathbf{x}}$ ist durch Formel 2.13 definiert.

$$\bar{\mathbf{x}} = \bar{\mathbf{x}_r} + [\mathbf{dx}]\ \bar{\alpha} \tag{2.13}$$

Der Vektor $\bar{\mathbf{x_r}}$ enthält alle x-Koordinaten der Punkte auf der rechten Streckenbegrenzung und [**dx**] ist eine Diagonalmatrix, auf dessen Hauptdiagonale sich die Differenzen von $x_{r,i} - x_{l,i}$ befinden. Alle anderen Werte sind 0. Bei einer Anzahl von vier Wegpunkten sieht die [**dx**]-Matrix wie folgt aus:

$$[\mathbf{dx}] = \begin{bmatrix} \Delta x_1 & 0 & 0 & 0 \\ 0 & \Delta x_2 & 0 & 0 \\ 0 & 0 & \Delta x_3 & 0 \\ 0 & 0 & 0 & \Delta x_4 \end{bmatrix} \tag{2.14}$$

Anschließend können die Werte für die Krümmung an jedem Punkt quadriert und aufsummiert werden, wie es in Formel 2.15 dargestellt wird.

$$\Gamma_x^2 = \left(\frac{d^2\bar{\mathbf{x}}(t)}{dt^2}\bigg|_{t=0} \right)^2 = ([\mathbf{D}]\bar{\mathbf{x}})^2 = \begin{bmatrix} c_{1,x} \\ c_{2,x} \\ \dots \\ c_{n,x} \end{bmatrix}^T \begin{bmatrix} c_{1,x} \\ c_{2,x} \\ \dots \\ c_{n,x} \end{bmatrix} \tag{2.15}$$

Die Formeln 2.10 bis 2.15 gelten analog für $y(t)$. Die Ergebnisse Γ_x^2 und Γ_y^2 werden schlussendlich addiert, um die Gesamtkrümmung zu erhalten.

2.7 Fahrzeugmechanik

Im Folgenden werden die fahrmechanischen Formeln aufgelistet, die zur Entwicklung einer einfachen Rundenzeitsimulation benötigt werden. Grundlegend soll für jeden Punkt auf der Strecke die Höchstgeschwindigkeit berechnet werden, mit der das Fahrzeug diesen Streckenabschnitt befahren kann und die Geschwindigkeiten unter Berücksichtigung von Be- und Entschleunigung aufeinander abgestimmt werden. Die Formeln stammen aus dem Fachbuch „Rennwagentechnik“ von Michael Trzesniowski ([18] S. 988ff.).

Die Höchstgeschwindigkeit eines Autos in der Kurve wird durch den Radius der Kurve r und die Fahrzeugeigenschaften bedingt.
Der Abtriebsbeiwert c_A, die Luftdichte p_L und die Querspantfläche A_V werden verwendet, um den Abtrieb (englisch Downforce) zu berechnen. Das ist die Kraft, mit der die Luft das Fahrzeug auf den Boden drückt. Durch einen hohen Abtrieb kann der Rennwagen selbst bei hohen Geschwindigkeiten den Querbeschleunigungen in der Kurve standhalten. Der Abtriebsbeiwert ist ein dimensionsloser Wert und stellt eine wichtige Kenngröße in der Strömungslehre dar. Der Beiwert kann zum Beispiel im Windkanal oder

durch Computational Fluid Dynamics (CFD) Simulationen bestimmt werden (siehe Abbildung A.1). Die Querspantfläche ist die Wirkungsfläche der Aerodynamik und gleicht dem Schatten des Fahrzeugs bei frontaler, nicht-punktueller Beleuchtung. Die Luftdichte ist abhängig von Luftdruck, Temperatur, Luftfeuchtigkeit und weiteren Umwelteinflüssen, kann zur Vereinfachung jedoch als konstant angesehen werden.
Der Reifenreibwert in Querrichtung bezeichnet die Reibungskraft im Verhältnis zur Anpresskraft und wird wiederum durch verschiedene Faktoren wie z.B. Temperatur, Radlast und Oberflächenzustand beeinflusst. Auch dieser Wert wird im Folgenden als konstant betrachtet.
Die Berechnung des Krümmungsradius r wird in Formel 2.23 betrachtet. Des weiteren werden zur Berechnung die Erdbeschleunigung g und das Fahrzeuggewicht m benötigt.

Höchstgeschwindigkeit v_{max} in m/s:

$$v_{max} = \sqrt{\frac{g}{\frac{1}{\mu \cdot r} - \frac{p_L \cdot c_A \cdot A_V}{2m}}} \tag{2.16}$$

- g: Erdbeschleunigung, m/s^2
- μ: Reifenreibwert in Querrichtung
- r: Krümmungsradius (siehe Formel 2.23), m
- p_L: Luftdichte, kg/m^3
- c_A: Abtriebsbeiwert (engl. lift coefficient)
- A_V: Querspantfläche, m^2
- m: Fahrzeugmasse, kg

Als nächstes wird die Berechnung der maximalen Beschleunigung betrachtet. Diese wird durch die Antriebskraft an den Rädern F_A, die Summe aller Fahrwiderstände F_W, die Fahrzeugmasse m und den Drehmassenzuschlagsfaktor k_m ermittelt. Der Drehmassenzuschlagsfaktor ist ein dimensionsloser Wert. Er beschreibt als Faktor der Gesamtmasse den zusätzlichen Widerstand, der aufgrund des Massenträgheitsmoments überwunden werden muss.

Maximal mögliche Beschleunigung a_{max} in m/s^2

$$a_{max} = \frac{F_A - F_W}{m \cdot k_m} \tag{2.17}$$

- F_A: Antriebskraft an den Rädern (siehe Formel 2.18), N
- F_W: Gesamtfahrwiderstand (siehe Formel 2.19), N
- m: Fahrzeugmasse, kg
- k_m: Drehmassenzuschlagsfaktor

Die Antriebskraft stellt vereinfacht dar, wie viel Kraft von den Rädern auf die Fahrbahn übertragen werden kann. Dabei spielt die maximale Motorleistung P_M, abgemindert durch den Wirkungsgrad des Antriebstrangs η und die Fahrgeschwindigkeit v eine Rolle.

Antriebskraft an den Rädern F_A in N:

$$F_A = \frac{1000 \cdot P_M}{v} \cdot \eta \tag{2.18}$$

- P_M: Motorleistung, kW
- v: Geschwindigkeit, m/s
- η: Wirkungsgrad vom Antriebsstrang

Der Gesamtfahrwiderstand wird durch die Summe von fünf verschiedenen Fahrwiderständen gebildet. Der Steigungswiderstand wird unter der Annahme, dass die Rennstrecke eben ist nicht weiter betrachtet. Der Schräglaufwiderstand entsteht, wenn die Bewegungsrichtung des Reifens und die Ausrichtung des Reifens auseinandergehen. Aufgrund des hohen Aufwandes zur Bestimmung und des geringen Ausmaßes auf die Fahrgeschwindigkeit, wird dieser Widerstand nicht in die Berechnung miteinbezogen.

Gesamtfahrwiderstand F_W in N:

$$F_W = F_R + F_\alpha + F_L + F_q + F_a \tag{2.19}$$

- F_R: Rollwiderstand (siehe Formel 2.20), N
- F_α: Schräglaufwiderstand, N
- F_L: Luftwiderstand (siehe Formel 2.21), N
- F_q: Steigungswiderstand, N
- F_a: Beschleunigungswiderstand (siehe Formel 2.22), N

Beim Abrollen des Reifens entsteht durch die Deformation und die ungleiche Druckverteilung eine Kraft, die der Bewegungsrichtung des Fahrzeugs entgegenwirkt. Der Rollwiderstand pro Reifen wird über die Radlast $F_{W,Z}$ berechnet, welche mit dem Rollwiderstandsbeiwert k_R multipliziert wird. Dieser Beiwert ist das Verhältnis zwischen dem Vertikalkraftversatz und dem Reifenradius, welcher sich während der Fahrt dynamisch verhält. Auch dieser Wert wird als konstant angenommen.

Rollwiderstand F_R in N:

$$F_R = k_R \cdot F_{W,Z} \tag{2.20}$$

- k_R: Rollwiderstandsbeiwert
- $F_{W,Z}$: Radlast, N

Der Luftwiderstand (englisch Drag) errechnet sich ähnlich zur Downforce, verwendet jedoch den Luftwiderstandsbeiwert c_W anstelle des Abtriebbeiwerts. Dieser Wert kann ebenfalls durch CFD-Simulationen ermittelt werden.

Luftwiderstand F_L in N:

$$F_L = \frac{1}{2} \cdot p_L \cdot c_W \cdot A_v \cdot v_L^2 \tag{2.21}$$

- p_L: Luftdichte, kg/m^3
- c_W: Luftwiderstandsbeiwert (engl. drag coefficient)
- A_V: Querspantfläche, m^2
- v_L: Anströmgeschwindigkeit, m/s

Beim Beschleunigungswiderstand spielen wie auch bei der maximalen Beschleunigung (siehe 2.17) der Drehmassenzuschlagsfaktor k_M, die Fahrzeugmasse m und zusätzlich die Beschleunigung in Fahrtrichtung a_X eine Rolle.

Beschleunigungswiderstand F_a in N:

$$F_a = k_m \cdot m \cdot a_X \tag{2.22}$$

- k_M: Drehmassenzuschlagsfaktor
- m: Fahrzeugmasse, kg
- a_X: Beschleunigung in Längsrichtung, m/s^2

Der Radius einer Kurve hat einen entscheidenden Einfluss auf die Höchstgeschwindigkeit. Der Krümmungsradius r an einem bestimmten Punkt kann näherungsweise über die Koordinaten von dem betrachteten Punkt (x_i, y_i), dem vorherigen Punkt (x_{i-1}, y_{i-1}) und dem nachfolgenden Punkt (x_{i+1}, y_{i+1}) wie folgt errechnet werden.

Krümmungsradius r in m:

$$\begin{aligned}
r &= \frac{1}{\kappa} \\
\kappa &= \frac{|\dot{x_i}\ddot{y_i} - \dot{y_i}\ddot{x_i}|}{(\dot{x_i}^2 + y_i^2)^{\frac{3}{2}}} \\
\dot{x_i} &\approx \frac{x_{i+1} - x_{i-1}}{2s} \\
\dot{y_i} &\approx \frac{y_{i+1} - y_{i-1}}{2s} \\
\ddot{x_i} &\approx \frac{x_{i+1} - 2x_i + x_{i-1}}{s^2} \\
\ddot{y_i} &\approx \frac{y_{i+1} - 2y_i + y_{i-1}}{s^2}
\end{aligned} \tag{2.23}$$

Ab einem gewissen Kurvenradius, kritischer Kurvenradius r_K genannt, wird die maximale Fahrgeschwindigkeit nicht mehr durch die auftretende Querbeschleunigung begrenzt. Infolgedessen kann die Kurve mit jeder Geschwindigkeit durchfahren werden. Dieser Radius lässt sich wie folgt berechnen:

Kritischer Radius r_K in m:

$$r_K = \frac{2 \cdot m}{\mu \cdot p_L \cdot c_A \cdot A_V} \tag{2.24}$$

- m: Fahrzeugmasse, kg
- μ: Reifenreibwert in Querrichtung
- p_L: Luftdichte, kg/m^3
- c_A: Abtriebsbeiwert (engl. lift coefficient)
- A_V: Querspantfläche, m^2

Die nächste Formel dient der Geschwindigkeitsberechnung von v_{next}, die durch eine Beschleunigung a ausgehend von einer Startgeschwindigkeit v_0 erreicht werden kann.

Folgegeschwindigkeit v_{next} nach Beschleunigung in m/s:

$$v_{next} = \sqrt{v_0^2 + 2as} \tag{2.25}$$

- v_0: Startgeschwindigkeit, m/s
- a: Beschleunigung, m/s^2
- s: Strecke, m

Um zu berechnen, wie viel Zeit benötigt wird, um eine Strecke s zu überwinden bei einer gegebenen Startgeschwindigkeit v_0 und einer Folgegeschwindigkeit v_{next}, kommt die folgende Formel zum Einsatz.

Zum Zurücklegen einer Strecke nötige Zeit t in s:

$$t = \frac{2 \cdot s}{v_0 + v_{next}} \tag{2.26}$$

- s: Strecke, m
- v_0: Startgeschwindigkeit, m/s
- v_{next}: Folgegeschwindigkeit, m/s

Mit der letzten Formel lässt sich berechnen wie hoch die Startgeschwindigkeit v_0 sein darf, damit die Folgegeschwindigkeit v_{next} über eine gegebene Beschleunigung a und eine Strecke s erreicht werden kann.

Ausgangsgeschwindigkeit v_0 in m/s

$$v_0 = \sqrt{v_{next}^2 - 2 \cdot a \cdot s} \tag{2.27}$$

- v_0: Startgeschwindigkeit, m/s
- v_{next} Folgegeschwindigkeit, m/s
- a: Beschleunigung, m/s^2
- s: Strecke, m

2.8 Evolutionäre Algorithmen

Evolutionäre Algorithmen (EAs) sind Optimierungsverfahren, die von den Prinzipien der natürlichen Selektion und genetischen Evolution inspiriert sind und damit besonders für die Lösung komplexer Optimierungsprobleme, bei denen traditionelle analytische Methoden versagen, nützlich. [16]
Die Grundidee von EAs besteht darin, eine Population von Individuen (Lösungskandidaten) zu erzeugen und diese durch wiederholte Anwendung von genetischen Operatoren wie Selektion, Kreuzung und Mutation zu verbessern.
Die Qualität der Lösungen wird durch eine Fitnessfunktion bewertet, die das Optimierungsziel widerspiegelt. Die besten Individuen werden bevorzugt ausgewählt, um Nachkommen zu erzeugen, wodurch sich die Population über Generationen hinweg verbessert. Einige Individuen der nächsten Generation werden mutiert. Ihr Genom wird zufällig verändert, um den Genpool zu erweitern. Dadurch werden die Chancen vergrößert, dass das globale Optimum erreicht wird.
Es gibt außerdem die Möglichkeit Elitismus einzusetzen, wodurch ein definierter Prozentsatz der besten Individuen einer Population unverändert in die nächste Generation übernommen werden. Hierdurch bleibt der bisherige Fortschritt erhalten.

Genetische Algorithmen

Genetische Algorithmen (GAs) sind eine Variante von EAs. Sie basieren auf einer direkten Nachbildung der genetischen Evolution und verwenden eine binäre oder reale Kodierung von Individuen. Die Hauptoperatoren sind Selektion, Kreuzung und Mutation. GAs sind besonders effektiv bei der Lösung von Problemen mit diskreten und kombinierten Suchräumen. In Abbildung 1 ist ein genetischer Algorithmus in Form von Pseudocode (basierend auf [16]) dargestellt.

Algorithm 1 Genetic Algorithm($popSize, mutationProb$)

```
population = randomInit(popSize)
bestIndividual = getFirst(population)
while ¬isTerminationCriterion(bestIndividual) do
    calculateFitness(population)
    bestIndividual = getBest(population)
    parents = matingSelection(population)
    elites = getElites(population)
    children = elites
    while |children| < |population| do
        for p1, p2 ∈ population do
            child = crossover(p1, p2)
            child = mutateChild(child, mutationProb)
            children ∪ {child}
        end for
    end while
    population = children
end while
return bestIndividual
```

Der Algorithmus beginnt mit der Initialisierung der Population mit (meist) zufälligen Werten. Solange das Terminierungskriterium z.B. ein vorgegebener Fitness-Wert nicht erreicht ist, wird eine neue Generation an Individuen erstellt. Dafür wird die bestehende Population mittels der Fitness-Methode bewertet. Aufgrund dieser Werte werden die möglichen Eltern-Kandidaten selektiert. Wird Elitismus eingesetzt, werden die besten Individuen gleich in die Kind-Population übernommen. Es werden nun je zwei Elternteile kombiniert, um daraus die Kinder zu erzeugen. Dafür gibt es verschiedene Möglichkeiten wie z.B. Single Point Crossover. Dabei werden die Gene bis zu einem zufälligen Punkt (vorstellbar als Index) von Elternteil A und ab dann von Elternteil B übernommen. Mit einer gewissen Wahrscheinlichkeit werden die Kinder-Individuen mutiert. Anschließend wird eine neue Iteration mit der neuen Kinder-Population gestartet. Durch die Wahl der

Parameter wie beispielsweise der Mutationswahrscheinlichkeit oder durch die Art der Fortpflanzung und die Gestaltung der Fitness-Funktion kann auf die Verhaltensweise des Algorithmus und die Qualität der Ergebnisse Einfluss genommen werden.

Neuroevolution
Im Bereich der Neuroevolution werden evolutionäre Algorithmen eingesetzt um künstliche neuronale Netze zu trainieren. Dies geschieht, indem die Gewichte und Bias Werte als Teil des Genoms der Individuen den Evolutionsprozess durchlaufen. Auch hier kommt eine Fitness-Funktion zum Einsatz, welche den Output des neuronalen Netzes bewertet, der durch die Verwendung der Kantengewichte und Schwellwerte eines Individuums berechnet wird. Sie ersetzt die herkömmlichen Loss-Functions, die beispielsweise beim Lernen mittels Backpropagation eingesetzt werden. Sie ist in ihrer Definition flexibler und erfordert keine gelabelten Trainingsdaten.
Es wird in der Neuroevolution unterschieden zwischen Algorithmen, die mit einem neuronalen Netz arbeiten, das eine festen Topologie besitzt und welchen, bei denen auch der Aufbau des neuronalen Netzes mit optimiert wird. Ein Beispiel hierfür ist der Algorithmus Neuro Evolution of Augmenting Topologies (NEAT). [1]

2.9 Ray Casting

Ray Casting (ursprünglich „Position of point relative to polygon“ von M. Schimrat, [15]) ist ein Algorithmus, mit dem herausgefunden werden kann ob ein Punkt innerhalb eines Polygons liegt. Dafür wird ausgehend von dem gefragten Punk (x_a, y_a) eine Gerade, der sogenannte Ray, in positive x-Richtung gezogen und gezählt wie oft diese Gerade die Kanten des Polygons schneidet. Ist die Anzahl der Schnittpunkte ungerade, befindet sich der Punkt innerhalb des Polygons und sonst außerhalb.
Um die Schnittpunkte zu bestimmen, wird über die Liste der Polygon-Punkte interiert und je zwei aufeinanderfolgende Punkte des Polygons (x_i, y_i und x_j, y_j) betrachtet. Für diese Punkte wird nun ermittelt, ob sich die y-Koordinate des Punktes y_a zwischen y_i und y_j befindet. Ist dies der Fall, wird der x-Wert des Schnittpunktes x_s wie folgt ermittelt:

$$x_s = x_j - x_i \cdot \frac{y_a - y_i}{y_j - y_i} + x_i \tag{2.28}$$

Gilt zusätzlich $x_a < x_s$, befindet sich der Punkt links vom Schnittpunkt und die Gerade schneidet diese Kante des Polygons.

3 Entwurf und Implementierung

Das hier entwickelte und beschriebene System trägt den Namen „Global Trajectory Optimizer“ (GTO). Es umfasst die verschiedenen Ansätze zur Rundenzeitoptimierung, sowohl wie die Verwaltung der Schnittstellen zu den Fremdsystemen und der Visualisierung der Ergebnisse, wie die folgende Abbildung 3.1 zeigt.

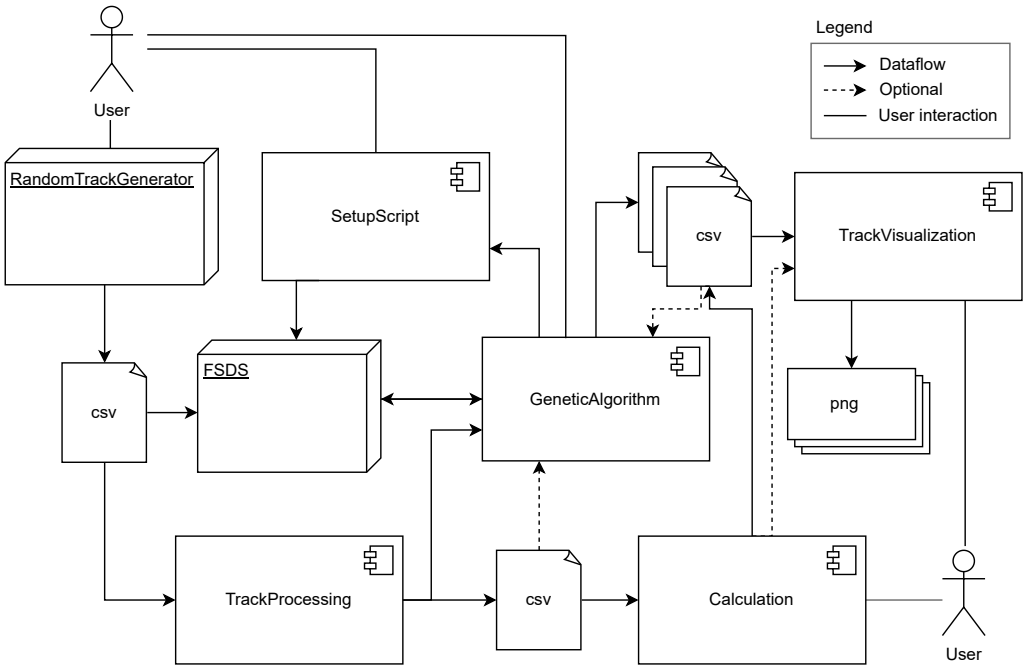


Abbildung 3.1: Kontextdiagramm: Zusammenspiel vom GTO mit externen Systemen

Die Komponente TrackProcessing verarbeitet die vom Random Track Generator (siehe Abschnitt 2.4) generierten Strecken, damit diese zur Trajektorienplanung verwendet werden können. Die Verarbeitung der Strecke wird in Abschnitt 3.2 beleuchtet und wurde in Java implementiert. Ebenso wurde der genetische Algorithmus in Java programmiert. Um ihn geht es in Abschnitt 3.4.
In Abschnitt 3.4.2 geht es um das Zusammenspiel von FSDS und dem genetischen Algorithmus. Hier findet das shell-SetupScript Anwendung.
Die in Abschnitt 3.3 beschriebene Berechnung der Ideallinie mit Geschwindigkeiten (Calculation-Komponente) wurde in Python programmiert. Da die ermittelten Trajektorien mit ihren Geschwindigkeiten in einem einheitlichen Format gespeichert werden, können diese, unabhängig von der Programmiersprache des Ansatzes, von der TrackVisualization geplottet werden. Dieser Teil der Anwendung ist in Python geschrieben.

3.1 Darstellung der Strecke

Wie in Abschnitt 2.1 beschrieben ist, wird die Rennstrecke mit Hütchen abgesteckt. Sie geben die Streckenbegrenzung vor und unterteilen die Strecke gleichzeitig in näherungsweise gleich lange Abschnitte. Die Koordinaten der Hütchen lassen sich somit zur Berechnung der Trajektorie verwenden (siehe Abschnitt 2.6). Dafür wird für jedes Hütchenpaar bestehend aus einem linken $(x_{l,i}, y_{l,i})$ und einem rechten Hütchen $(x_{r,i}, y_{r,i})$ die Koordinaten mit einem Parameter α_i verrechnet und damit die Position des Wegpunktes zwischen ihnen bestimmt (siehe Abbildung 3.2).

Vorteilhaft an der Streckendarstellung aus [3] ist, dass für die Definition eines Wegpunkte nur jeweils ein Parameter (α) notwendig ist und dass der Wegpunkt zwangsläufig auf der Strecke liegt.
Für jedes zusammengehörige Hütchenpaar, wird somit ein Parameter α_i und eine Zielgeschwindigkeit v_i gesucht, die im Zusammenspiel eine möglichst gute Rundenzeit ermöglichen.

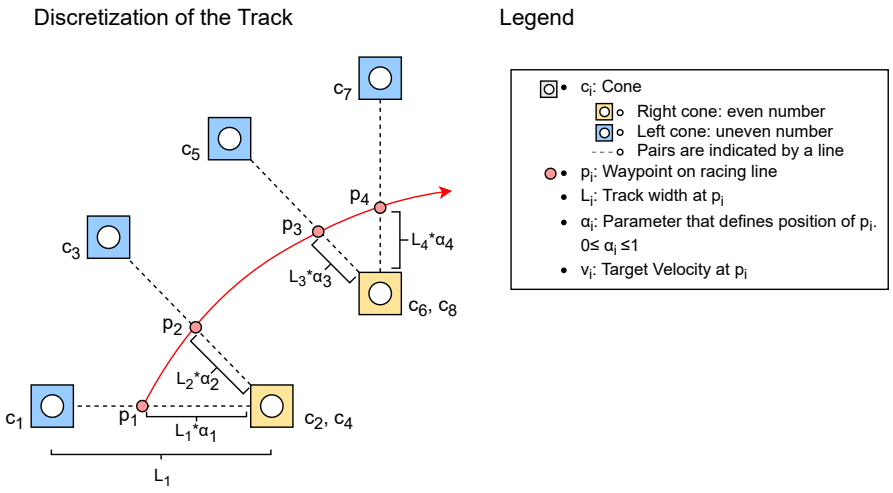


Abbildung 3.2: Darstellung der Strecke

3.2 Verarbeitung der Strecke

Der eben beschriebene Ansatz (Abschnitt 3.1) setzt voraus, dass die Hütchen in Paare unterteilt sein müssen. Dabei ergeben sich zwei Probleme: Erstens ist die Anzahl der linken und rechten Hütchen nicht immer gleich und zweitens können die Hütchen nicht der Reihe nach paarweise zugeordnet werden, da diese dann nicht unbedingt direkt gegenüber stehen.
Das liegt daran, dass es in einer Kurve mehr äußere Hütchen gibt. In einer Rechtskurve gibt es mehr linke, in einer Linkskurve mehr rechte Hütchen. Dadurch ergibt sich in einer Kurve eine surjektive Abbildung der äußeren Hütchen auf die inneren Hütchen-Partner. Jedes äußere Hütchen hat immer genau einen inneren Gegenspieler. Ein inneres Hütchen kann aber mehrere äußere Hütchen zum Partner haben.
Dies kann bei der Hütchenpaar-Zuweisung ausgeglichen werden, indem das innere Hütchen, das mehrere äußere Hütchen zum Partner hat mehrfach in die Liste hinzugefügt wird. Die Bestimmung der korrekten Partner erfolgt über den Vergleich der Distanz zueinander. Der dafür verwendete Algorithmus, ist in Form von Pseudocode 2 dargestellt. Der Algorithmus unterscheidet zwischen drei Fällen der Hütchen-Beziehungen. Diese sind in Abbildung 3.3 visuell dargestellt.

Algorithm 2 MatchConePairs($leftCones[1...n]$, $rightCones[1...m]$)

```
l = 1
r = 1
lastL = leftCones[n]
lastR = rightCones[m]
resultL = ∅
resultR = ∅
while l < n and r < m do
    dCurrent = distance(leftCones[l], rightCones[r])
    dLastR = distance(leftCones[l], lastR)
    dLastL = distance(rightCones[r], lastL)
    if dCurrent ≤ dLastR and dCurrent ≤ dLastL then                ▷ Case 1
        resultL.add(leftCones[l])
        resultR.add(rightCones[r])
        lastL = leftCones[l]
        lastR = rightCones[r]
        l = l + 1
        r = r + 1
    else if dLastR ≤ dCurrent and dLastR ≤ dLastL then             ▷ Case 2
        resultL.add(leftCones[l])
        resultR.add(lastR)
        lastL = leftCones[l]
        l = l + 1
    else                                                           ▷ Case 3
        resultR.add(rightCones[r])
        resultL.add(lastL)
        lastR = rightCones[r]
        r = r + 1
    end if
end while
return resultL, resultR
```

In Abbildung A.2 werden an einem Streckenbeispiel die Konsequenzen gezeigt, wenn keine Hütchenzuordnung erfolgt. In diesem Beispiel verläuft die berechnete Mittellinie ($\forall \alpha_i = 0,5$) teilweise außerhalb der Streckenbegrenzung.

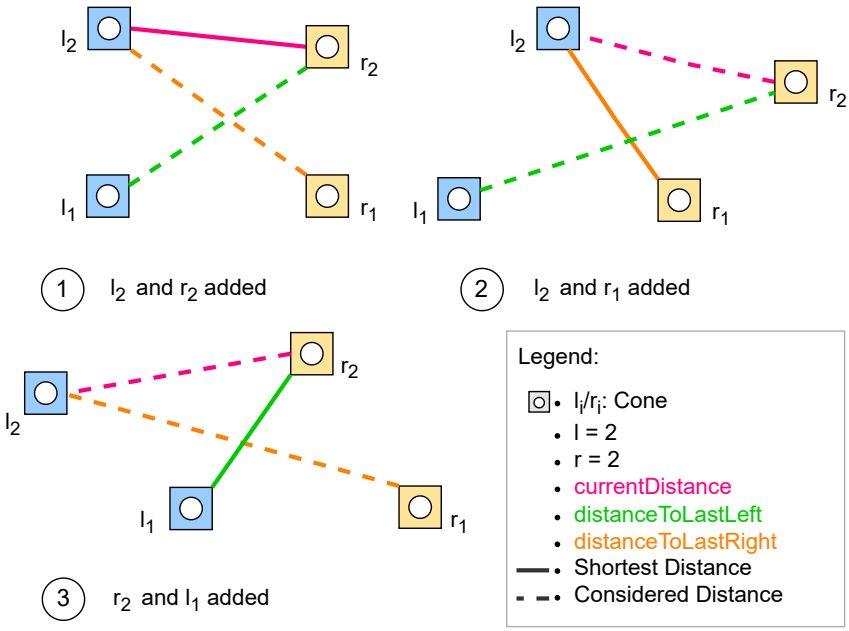


Abbildung 3.3: Darstellung der Fälle aus Algorithmus 2

3.3 Numerische Berechnung

3.3.1 Minimierung der Trajektorien-Krümmung

Im Abschnitt 2.6 der Grundlagen wird dargelegt, wie die Krümmung einer Trajektorie bestehend aus Wegpunkten berechnet werden kann. Dazu wird die zweite Ableitung der kubischen Funktionen gebildet, die jeweils zwei Wegpunkte miteinander verbinden und die Ergebnisse aufaddiert. Die Position eines Wegpunkts, wird über den Parameter α (mit $0 \leq \alpha \geq 1$) bestimmt, der für eine Verschiebung des Punktes entlang der Strecke zwischen der rechten und linken Streckenbegrenzung (Hütchen) sorgt (siehe Formel 2.2). Die Summe der Krümmungen bietet eine Möglichkeit, Trajektorien mit unterschiedlicher Wegpunkt-Anordnung zu vergleichen und zu bewerten.
Die Suche nach der Trajektorie mit der geringsten Krümmung wie sie hier definiert ist, ist ein quadratisches Problem und kann mit einem entsprechenden Solver gelöst werden. In diesem Fall, wird das Problem über die Methode scipy.optimize.minimize() und dem Verfahren SLSQP gelöst.

Die Verwendung der α-Parameter zur Bestimmung der Wegpunkt-Positionen hat den Vorteil, dass alle Wegpunkte stets auf der Strecke liegen. Da in der Disziplin Trackdrive für das Umfahren von Hütchen Zeitstrafen verhängt werden (siehe Tabelle 2.1), muss ein Sicherheitsabstand von mindestens einer halben Auto-Breite zu den Hütchen eingehalten werden. Die α-Parameter beziehen sich fortan auf den eingeschränkten Fahrbereich.
Der gewählte Abstand beträgt 839 mm bei einer Autobreite von 1449 mm und einer Hütchenbreite von 228 mm.

3.3.2 Velocity Planning

Zusätzlich zu der Trajektorienplanung muss für jeden Wegpunkt auch die Zielgeschwindigkeit bestimmt werden, mit der dieser Punkt durchfahren werden soll. Ausschlaggebend dafür ist der Krümmungsradius.
Des weiteren müssen die Geschwindigkeiten angepasst werden, sodass sie über die maximale Be- oder Entschleunigung des Autos erreichbar sind.
Es wird der folgende Algorithmus 3 verwendet (angelehnt an [2] und [18] S. 988ff.).
Laut Algorithmus 3 wird zunächst vorwärts über alle Punkte der Strecke iteriert und die Geschwindigkeit v_i für den betrachteten Punkt i gesetzt. Die Geschwindigkeit ergibt sich entweder über die maximal mögliche Geschwindigkeit v_{max} abhängig von dem Kurvenradius r oder über die Geschwindigkeit v_{ac}, je nachdem welcher Wert niedriger ist. Die Geschwindigkeit v_{ac} ist die höchste Geschwindigkeit, die mittels der Maximalbeschleunigung a_{max} von dem vorherigen Wegpunkt mit der Geschwindigkeit v_{i-1} erreicht werden kann.
Anschließend wird die Strecke rückwärts durchlaufen und die Geschwindigkeiten v_{i-1} angepasst, dessen Folgegeschwindigkeit v_{i+1} so niedrig ist, dass sie durch die Entschleunigung a_{min} und die bisherige Ausgangsgeschwindigkeit nicht erreichbar wäre. Mittels der Geschwindigkeitsverringerung an den vorherigen Punkten wird sichergestellt, dass alle folgenden Geschwindigkeitsbegrenzungen durch Kurvenradien eingehalten werden können.

Die Werte P_M, η, k_R, $F_{W,Z}$, p_L, c_W, A_V, k_m, m, c_A, g und μ sind Konstanten. Die dafür eingesetzten Werte können Tabelle 3.1 entnommen werden.

Algorithm 3 VelocityPlanning($wayPoints[1...n]$, P_M, η, k_R, $F_{W,Z}$, p_L, c_W, A_V, k_m, m, c_A, g, μ)

$v = \emptyset$
for $i = 1$ to n **do** ▷ Achievable velocities via acceleration
  $F_A = drivingForce(v[i-1], P_M, \eta)$
  $F_W = resistance(k_R, F_{W,Z}, pL, c_W, A_V, v[i-1], k_m, m, a_{i-1})$
  $a_{max} = accel(F_A, F_W, m, k_m)$
  $s = distance(wayPoints[i-1], wayPoints[i])$
  $r = radius(wayPoints[i-1], wayPoints[i], wayPoints[i+1], s)$
  $v[i] = min(v_{max}(g, \mu, m, r, p_L, c_A, A_V), v_{ac}(v[i-1], a_{max}, s))$
end for
$t = 0$
for $i = n$ to 1 **do** ▷ Achievable velocities via deceleration
  $F_W = resistance(k_R, F_{W,Z}, p_L, c_W, A_V, v[i-1], k_m, m, 0)$
  $a_{min} = accel(0, F_W, m, k_m)$
  $s = distance(wayPoints[i], wayPoints[i+1])$
  if $v[i+1] < v[i]$ and $v[i+1] < v_{ac}(v[i], a_{min}, s)$ **then**
    $v[i] = v_{last}(v[i+1], a_{min}, s)$
  end if ▷ Calculate lap time
  $t = t + traversalTime(v[i+1], v[i], s)$
end for
return v, t

Zunächst muss für jeden Punkt die Antriebskraft an den Reifen $drivingForce(v, P_M, \eta)$ anhand von Geschwindigkeit (v), Motorleistung (P_M) und Wirkungsgrad vom Antriebsstrang (η) bestimmt werden (Formel siehe 2.18). Zur Vereinfachung wird jedoch die vorherige Geschwindigkeit $v[i-1]$ eingesetzt.
Anschließend kann auch der Gesamtfahrwiderstand ($resistance(...)$) anhand von Formel 2.19 berechnet werden. Dabei werden der Rollwiderstand, Luftwiderstand und Beschleunigungswiderstand betrachtet. Da der Trackdrive auf einer nahezu ebenen Fläche stattfindet, kann der Steigungswiderstand vernachlässigt werden. Ebenso wird der Schräglaufwiderstand nicht bedacht. Auch hier finden sich Vereinfachungen ($v[i-1]$ statt $v[i]$ und a_{i-1} statt a_i).
Mittels der Methode $accel(F_A, F_W, m, k_m)$ wird die mögliche Beschleunigung bei den gegebenen Parametern berechnet: Der Antriebskraft an den Rädern F_A, dem Gesamtfahrwiderstand F_W, der Fahrzeugmasse m und dem Drehmassenzuschlagsfaktor k_m. Ist die

Antriebskraft 0, wird statt der maximalen Beschleunigung die minimale Beschleunigung, also die negative Beschleunigung berechnet, die beim Ausrollen das Auto entschleunigt. Berechnet wird diese über Formel 2.17.
Der Krümmungsradius am Punkt P_i ($radius(P_{i-1}, P_i, P_{i+1}, s)$) wird näherungsweise über die Koordinaten des vorherigen Punktes P_{i-1}, dem Folgepunkt P_{i+1} und der Segmentlänge s ($distance(P_i, P_{i+1})$) berechnet. Dabei findet die Formel 2.23 Anwendung.
$v_{max}(g, \mu, m, r, F_D)$ dient der Berechnung der Höchstgeschwindigkeit (siehe Formel 2.16) bei dem Kurvenradius r, der Erdbeschleunigung g, dem Reifenreibwert in Querrichtung μ, der Fahrzeugmasse m, der Luftdichte p_L, dem Abtriebsbeiwert c_A und der Querspantfläche A_V.
Die Methode $v_{ac}(v_0, a, s)$ berechnet die Folgegeschwindigkeit mit der Formel 2.25, die von der Ausgangsbeschleunigung v_0 mit der Beschleunigung a über die Strecke s erreichbar ist.
Über die Funktion $v_{last}(v[i+1], a_{min}, s)$ lässt sich die höchste Geschwindigkeit ermitteln, aus der die Entschleunigung (mittels a_{min}) auf die Folgegeschwindigkeit $v[i+1]$ über die Strecke s noch möglich ist (siehe Formel 2.27).
Die Rundenzeit wird ermittelt, indem für jeden Streckenabschnitt über die Methode $traversalTime(v_1, v_2, s)$ berechnet wird wie lange das Auto bei konstanter Beschleunigung braucht, um die Strecke s zu überwinden, während es von v_1 auf v_2 be- bzw. entschleunigt.

Konstante	Bezeichnung	Eingesetzter Wert
g	Erdbeschleunigung	9,81 m/s^2
μ	Reifenreibwert in Querrichtung	1,76
m	Fahrzeugmasse	215 kg
p_L	Luftdichte	1,225 kg/m^3
c_A	Abtriebsbeiwert (engl. lift coefficient)	3,9
c_W	Luftwiderstandsbeiwert (engl. drag coefficient)	1,6
A_V	Querspantfläche	1 m^2
k_m	Drehmassenzuschlagsfaktor	1,2
P_M	Motorleistung	108 kW
η	Wirkungsgrad Antriebsstrang	0,88
k_R	Rollwiderstandsbeiwert	0,013
$F_{W,Z}$	Radlast	527,29 N

Tabelle 3.1: Verwendete Konstanten bei der Rundenzeitberechnung

3.4 Genetischer Algorithmus

Beim genetischen Algorithmus soll wie beim vorherigen Ansatz eine geeignete Kombination aus Geschwindigkeiten und Alpha-Werten für alle Hütchenpaare gefunden werden. Die einzelnen Lösungskandidaten erhalten ihre Bewertung über die in der FSDS erreichte Rundenzeit und erhaltene Zeitstrafen.
In dieser Arbeit werden mehrere Versionen des Algorithmus mit jeweils verschiedenen Optimierungszielen entwickelt. Die erste Variante beschäftigt sich mit der Optimierung der Trajektorie und der dazugehörige Geschwindigkeiten für eine spezifische Strecke. Das bedeutet, dass der Evolutionsprozess für jede neue Strecke von vorne beginnen muss. Alternativ soll versucht werden eine übergeordnete Fahrstrategie zu finden, die auch bei unbekannten Strecken sinnvolle Ergebnisse berechnet. Hierfür werden die Gewichte und Biases eines neuronalen Netzes mit fester Topologie mithilfe des genetischen Algorithmus optimiert.
Zunächst wird in diesem Kapitel auf den Entwurf der Software eingegangen und die Implementierungen der beiden Varianten erläutert. Im Anschluss soll dargestellt werden, wie die Schnittstelle zur FSDS und der Driverless Software realisiert wurde.

3.4.1 Software Entwurf

Die beiden Varianten des genetischen Algorithmus haben trotz ihrer Unterschiede im Optimierungsziel und der Modellierung ihrer Individuen die gleiche Funktionsweise. Daher gibt es einen evolutionären Algorithmus der mit verschiedenen Arten von Individuen und Zielen arbeiten kann. Konkret bedeutet es die Verwendung eines Strategy Patterns bei den Zielen *OptimalLapTimeStrategy* (streckenspezifisch) und *DrivingTechniqueStrategy* (streckenübergreifend), die mit den Individuen *SpeedTrajIndividual* und *DrivingTechIndividual* arbeiten. Die Vererbungsbeziehungen werden im folgenden UML-Diagramm dargestellt. Einen ausführlicheren Überblick bietet Abbildung A.3.

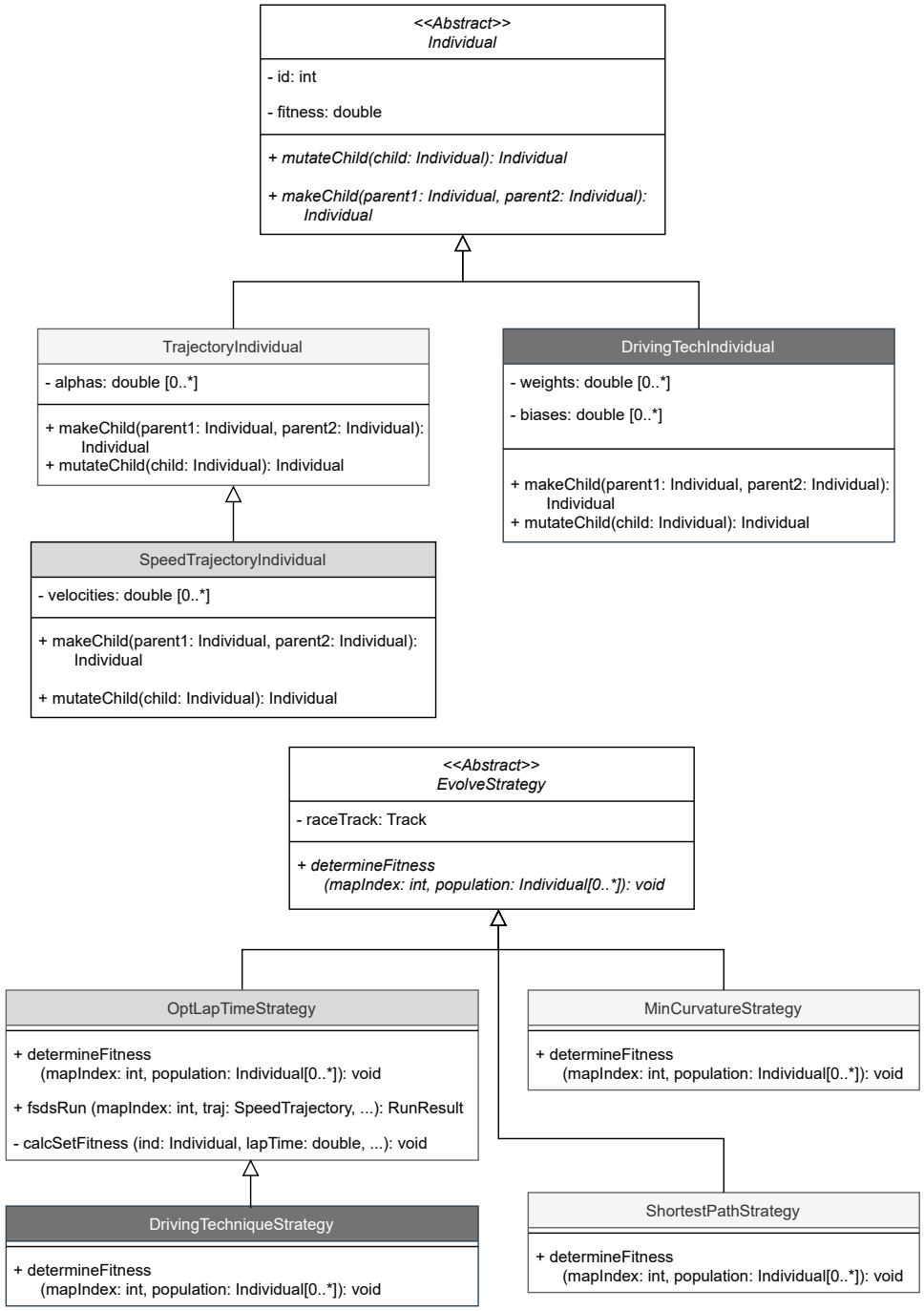


Abbildung 3.4: Vererbungsbeziehungen zwischen den Strategien- und Individuen-Klassen.

In dem Entwurf sind außerdem die Möglichkeiten zur Optimierung der Strecke oder Krümmung einer Trajektorie (*TrajectoryIndividual*) zu finden. Sie wurden der Vollständigkeit halber mit aufgeführt, werden aber nicht weiter beschrieben.

3.4.2 Schnittstelle zur Simulationsumgebung

Die komplette Kommunikation zwischen den Komponenten FSDS, Driverless Software und Global Trajectory Optimizer läuft über ROS.
Im folgenden Absatz wird der Einfachheit halber von Nachrichten geschrieben, die von einem Sender an einen Empfänger gesendet werden. Nach dem Publisher-Subscriber Pattern (siehe 2.5) bedeutet dies das publishen einer Nachricht durch den Sender auf einem vom Empfänger abonnierten Topic.
Das Zusammenspiel der Komponenten wird in Abbildung 3.5 veranschaulicht. Zunächst wird die Trajektorie, bestehend aus Wegpunkten und Geschwindigkeiten wird vom GTO an die DV Software geschickt, welche die Regelung übernimmt und den ermittelten Lenkwinkel und die Geschwindigkeitsanforderung an die FSDS weiterleitet. Die Fahrbefehle werden in der FSDS ausgeführt und die aktuelle Fahrzeugposition und später auch die Ergebnisse an den GTO zurückgeschickt.
Die konkret versendeten Nachrichten und ihre Reihenfolge sind in dem Pseudocode 4 und in den Tabellen A.1 und A.2 dokumentiert.

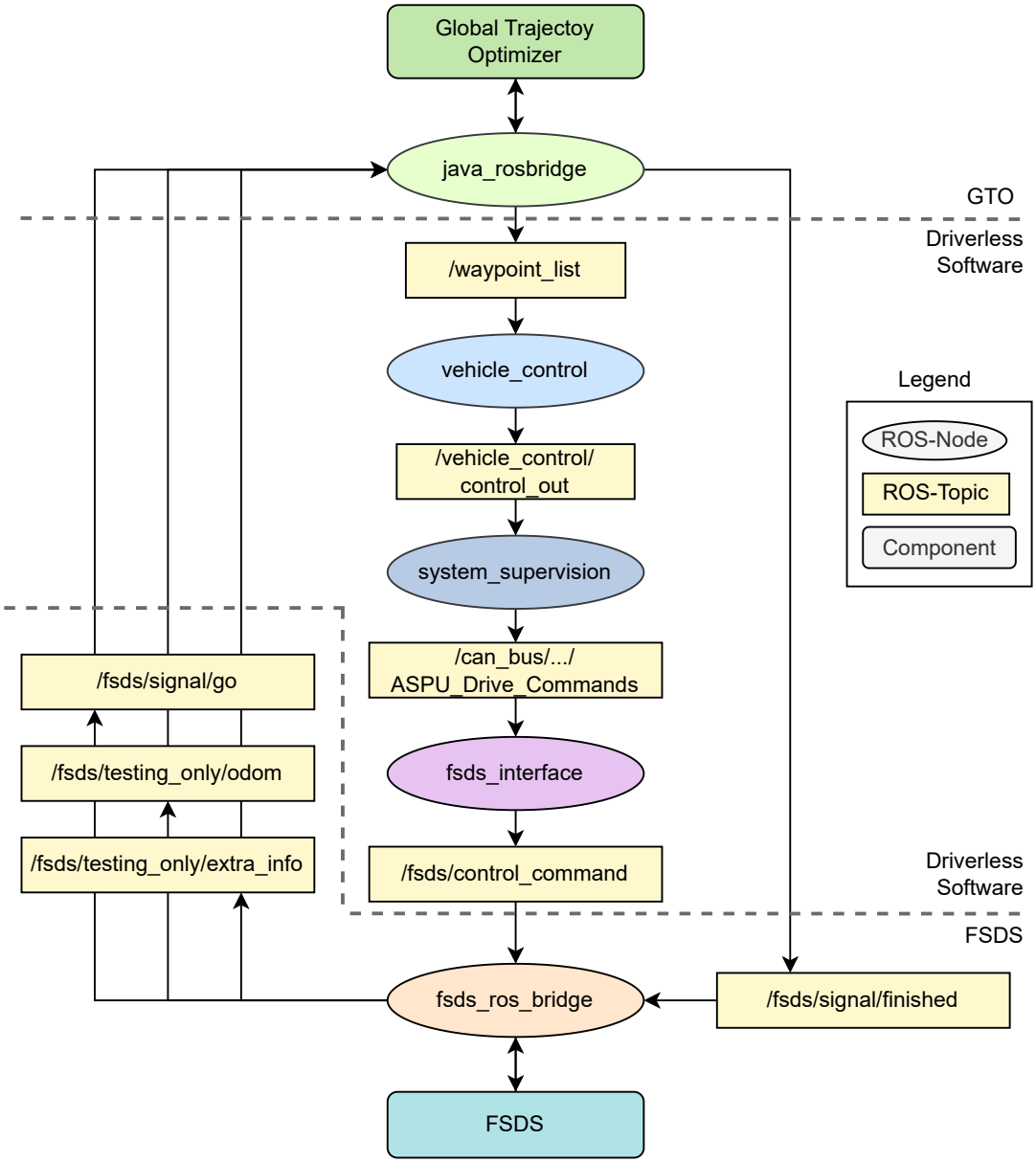


Abbildung 3.5: ROS-Schnittstelle zwischen GTO, DV Software und FSDS (Angelehnt an [14], Abbildung 6.2).

Die Individuen werden anhand ihrer Leistung in der FSDS bewertet. Jedes Individuum muss in einer neuen Instanz der Simulation fahren, damit alle Hüttchen wieder aufgestellt werden und die Bedingungen für alle gleich sind. Das (Neu-)Starten der Simulation mit der richtigen Strecke (*mapIndex*), der Start der Driverless Software und der Aufbau der Verbindungen über ROS untereinander wurde mit einem shell-Skript automatisiert und in der folgenden Abbildung veranschaulicht.

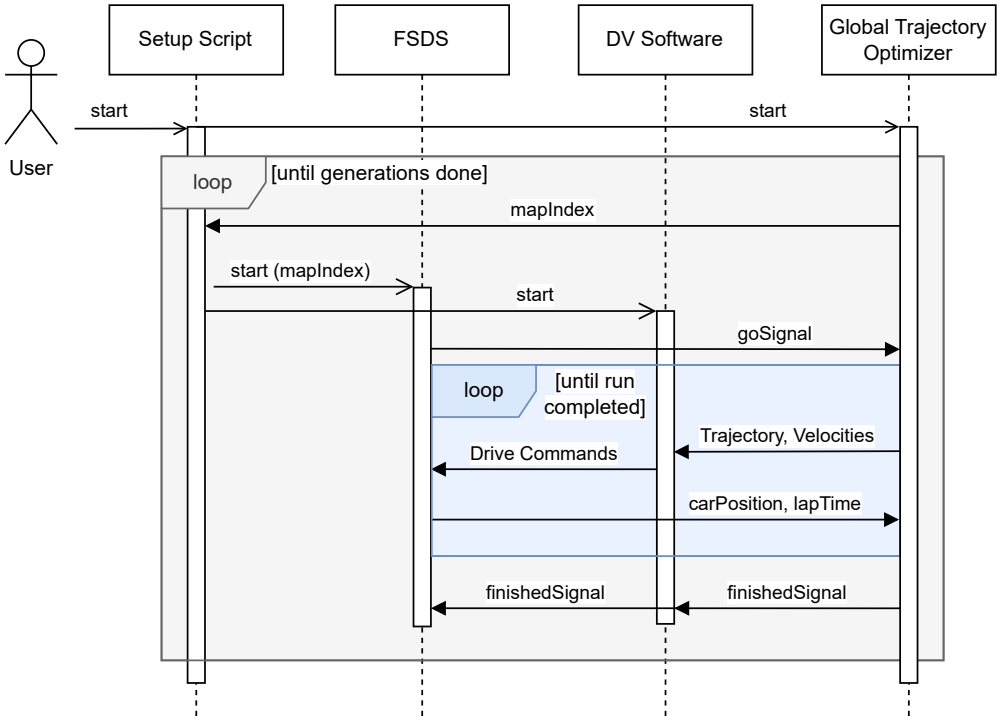


Abbildung 3.6: Laufzeitdiagramm: Zeigt die Interaktionen zwischen Setup-Script, FSDS, DV Software und GTO

Innerhalb der Java-Umgebung wird die ROS Schnittstelle über das Framework „java_rosbridge“ [9] realisiert und mittels des Adapter-Entwurfmusters gekapselt. Dadurch muss sich die restliche Anwendung nicht um die technischen Details der Nachrichtenübertragung kümmern. Hiermit wird das Architekturprinzip der Aufgabentrennung umgesetzt. Außerdem sind die Komponenten dadurch wiederverwendbar und die Übertragungsart ist austauschbar.

Bei der Übermittlung der Trajektorie gibt es drei Hauptpunkte zu beachten.
Es darf zur Zeit nur ein Teil der Wegpunkte übermittelt werden. Dies verkleinert die

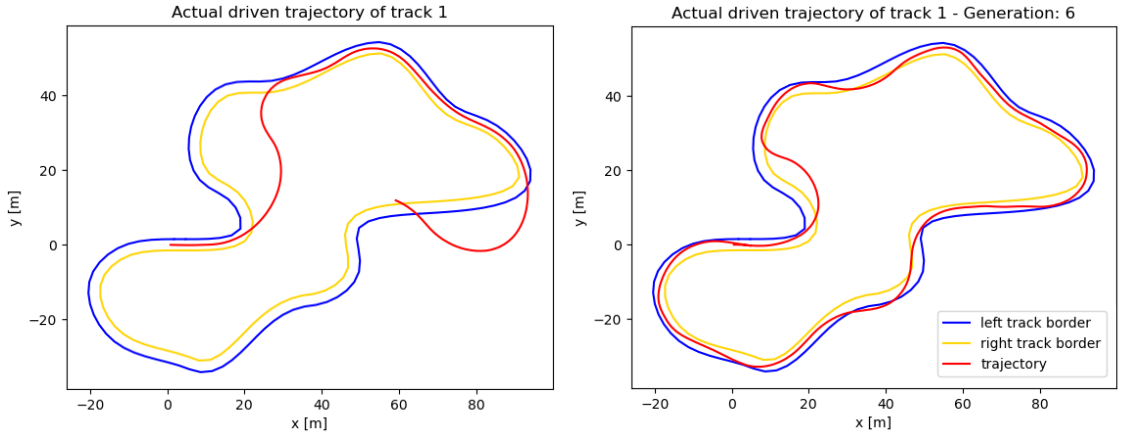
Menge der zu übertragenden Daten und sorgt zudem dafür, dass der Rennwagen die Wegpunkte in der richtigen Reihenfolge abfährt. In der aktuellen Implementierung werden jeweils die nächsten sechs Wegpunkte zur momentanen Fahrzeugposition übermittelt. Des weiteren muss überprüft werden, dass sich die Wegpunkt-Fenster von einer Iteration zur nächsten in ihren Punkten überschneiden. Ansonsten kann es sein, dass ein Fahrzeug, das von der Strecke abgekommen ist auf einem ganz anderen Streckenabschnitt wieder auf die Strecke fährt, weil dieser Abschnitt näher ist. Folglich kommt es zu Abkürzungen der Strecke, die laut Regelwerk verboten sind [7]. Abbildung A.4 zeigt ein solches Beispiel.
In dem Fall, dass sich die Wegpunkte aus der letzten Iteration nicht mit den aktuellen überschneiden, wird das alte Fenster erneut gesendet. Dadurch findet das Fahrzeug zum korrekten Streckenabschnitt zurück wie in Abbildung A.22 zu sehen ist.
Zuletzt muss die Übertragungsrate beachtet werden. Die DV Software erwartet die Wegpunkte in einem 10 Hz Takt. Allerdings ist die Java-Anwendung in der Lage die Trajektorie sehr viel schneller zu berechnen und den Befehl zur Übertragung zu geben.
Das führt jedoch dazu, dass sich die Nachrichten in der Ausgangswarteschlange der java_rosbridge stauen. Dadurch erhält die FSDS teilweise veraltete Wegpunkte und das Fahrzeug fährt Kreise auf der Strecke (siehe A.5).
Seit der Umstellung auf eine 10 Hz Übertragung, funktioniert das Zusammenspiel von GTO, DV Software und FSDS einwandfrei.

3.4.3 Gestaltung des GA und Parameterwahl

Fitnessfunktion: Die Bewertung der Individuen-Performance ist angelehnt an die Punkteverteilung bei der Trackdrive-Disziplin (siehe Tabelle 2.1). Das Auto in der FSDS wird mit den Werten eines Individuums fahren gelassen und die Ergebnisse aufgezeichnet. Es gibt drei Fälle, in denen ein Run frühzeitig abgebrochen wird:

- Stillstand: Das Auto bewegt sich zu lange nicht von der Stelle, ist also erst gar nicht losgefahren, auf der Strecke stehen geblieben oder gegen die Reifenwand am Rand der befahrbaren Fläche gefahren.
- Zu lange Off-Course: Das Auto findet nicht mehr oder nicht schnell genug zurück zur Strecke.
- Zu oft Off-Course: Es wird die festgelegt Höchstanzahl an Off-Courses überschritten.

Durch erste Experimente wurde erkannt, dass insbesondere die Begrenzung der zulässigen Off-Courses einen großen Einfluss auf das Evolutions-Ergebnis hat. Während eine generelle Höchstzahl sinnvoll ist, um Kandidaten herauszufiltern, die ständig die Strecke verlassen, ist eine zu niedrige Zahl hinderlich zum Lernerfolg. Es hat sich gezeigt, dass die Ergebnisse erheblich besser werden, wenn die Anzahl der Off-Courses erst in die Berechnung einfließen, sobald ganze Runden gefahren werden.
Welche Konsequenzen die Anzahl an erlaubten OCs auf die Performance des Algorithmus haben kann, zeigt das Beispiel in Abbildung 3.7.



(a) allowedOC: 3, Generation: 39, OC: 3 (b) allowedOC: 5, Gen.: 6, OC: 2

Abbildung 3.7: Beispiel zur Auswirkungen der OC-Begrenzung auf die Ist-Trajektorie: Keine ganze Runde nach 39 Generationen (a) oder ganze Runde ab Generation 6 (b).

Die Fitness der Individuen wird wie folgt ermittelt:

- Rundenzeit: Die benötigte Zeit in Sekunden oder ein Fixwert bei unvollständigen Runden.
- Frühzeitiger Abbruch: Wenn der Durchlauf abgebrochen wird ohne dabei die Anzahl an Off-Courses auszureizen, ist das Auto stehen geblieben oder nicht zur Strecke zurückgekehrt. In diesem Fall erhält das Individuum eine zusätzliche Bestrafung.
- Zeitstrafen: Wie beim Trackdrive erhält ein Individuum Strafen für jedes umgefahrene Hütchen und jedes Mal, wenn alle vier Autoreifen die Strecke verlassen.

- OC Zeit: Um bei gleicher OC-Anzahl feststellen zu können welches Individuum besser fährt, wird zusätzlich die Zeit außerhalb des Rundkurses auf den Fitnesswert addiert. Dadurch werden verkürzte Off-Course-Zeiten bevorzugt und die Population kann sich immer weiter der Strecke nähern. Die Zeit wird zusätzlich mit der Anzahl der Vorkommnisse multipliziert, damit nach wie vor erst die Anzahl an OCs verringert wird.
- OC Bestrafung: Es gibt noch eine zusätzliche Bestrafung für OCs, die unabhängig von der Anzahl und Länge der Vorkommnisse vergeben wird. Dadurch wird ein Individuum, dass konsequent auf der Strecke bleibt einem Individuum vorgezogen, das trotz mehrerer OCs die schnellere Rundenzeit oben genannter Strafen hat.
- Anzahl Wegpunkte: Die Anzahl abgefahrener Wegpunkte hilft insbesondere beim Vergleich von Individuen, die noch keine ganzen Runden fahren und führt zur Vermeidung von OCs.

Es handelt sich hierbei um ein Minimierungsproblem. Es gilt, je niedriger die Fitness, desto besser.

Off Course Erkennung: Während die Rundenzeit und die Anzahl umgefahrener Hütchen von der FSDS mitgeteilt werden, musste die OC-Detection selber entwickelt werden. Sie funktioniert über die Ray-Casting Methode (siehe Abschnitt 2.9), welche ermöglicht festzustellen ob ein Punkt innerhalb eines Polygons liegt.
Hierfür wird aus den äußeren Hütchen einer Strecke ein Polygon A und aus den inneren Hütchen ein Polygon B gebildet. Wenn mindestens einer der vier Punkte, welche die äußersten Punkte der Reifen darstellen innerhalb von A und außerhalb von B liegen, befindet sich das Auto noch so weit auf der Strecke, dass es vom Regelwerk her nicht als Off-Course gilt.
Abbildung 3.8 veranschaulicht das Vorgehen bei der Off-Course Erkennung.

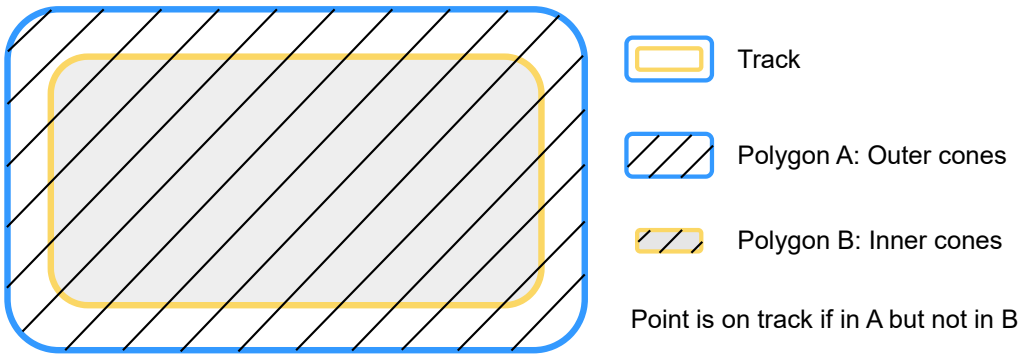
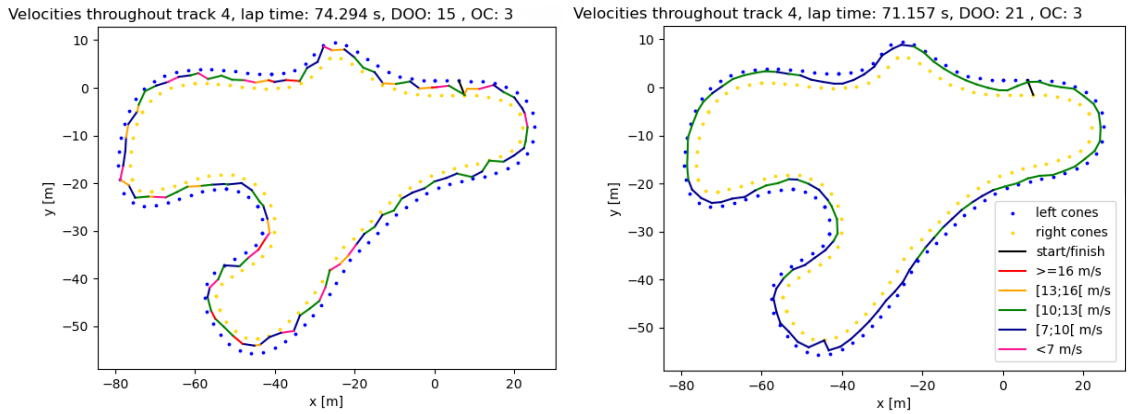


Abbildung 3.8: Visualisierung der Polygone zur OC-Detection.

Selektion & Elitismus: Die Selektion von Eltern-Individuen zur Fortpflanzung erfolgt mithilfe der ermittelten Fitnesswerte aller Individuen. Ein bestimmter Prozentanteil der Eltern-Population mit der besten Fitness wird unverändert in die Kind-Population übernommen (*elites*). Die schlechtesten Individuen pflanzen sich nicht fort. Sie fallen gewissermaßen der natürlichen Selektion zum Opfer (*selectionVictimsRate*).

Rekombination: Bei der Erstellung eines Kind-Individuums wird das Genom von zwei unterschiedlichen Eltern-Individuen kombiniert oder verrechnet. Nach einigen Versuchen hat sich die Rekombinationsvariante als beste erwiesen, bei der für jedes Gen der Mittelwert der beiden Eltern berechnet wird. Ausprobiert wurden außerdem Single Point Crossover und das zufällige Entscheiden ob ein Gen vom ersten oder zweiten Elternteil vererbt wird.

Mutation: Die Mutation wird in diesem Fall durch zwei Parameter gesteuert. Zum einen durch die allgemeine *mutationRate*, die festlegt ob ein Kind-Individuum überhaupt mutiert wird und zum anderen durch die *geneMutationRate*, die für jedes Gen von dem zur Mutation ausgewähltem Individuum entscheidet ob dieses von der Mutation betroffen wird. In einem ersten Versuch wurden einem Gen bei der Mutation ein komplett neuer Wert innerhalb des gültigen Wertebereichs zugeordnet. Dies führte allerdings zu teils sehr großen Veränderungen des Genoms. Daraufhin wurde die Mutation angepasst und eine Schrittgröße festgelegt. Ein Gen verändert sich maximal um einen zufälligen Faktor (zwischen -3 und 3) der *stepSize*. Dadurch liegen die Werte näher beieinander und können sich graduell verbessern. Abbildung 3.9 zeigt die Vorteile einer festen *stepSize* in der Mutation.



(a) Zufällige neue Werte bei Mutation (b) Wert wird um Stepsize angepasst

Abbildung 3.9: Beispiel zur Auswirkungen von verschiedenen Mutationsarten auf die Soll-Trajektorie. Kleinere Wertänderungen (b) führen zu regelmäßigeren Trajektorien und Geschwindigkeiten als bei zufälligen neuen Genen (a).

Terminierungskriterien: Es gibt zahlreiche Möglichkeiten, um diesen Algorithmus zu beenden. Man kann einen Fitness-Threshold festlegen, der versucht wird zu unterbieten oder eine maximale Anzahl an durchzuführenden Generationen vorgeben. Alternativ lässt sich bestimmen nach wie vielen Generationen ohne Verbesserung des besten Fitnesswerts abgebrochen werden soll, weil dann davon ausgegangen wird, dass die Lernkurve abgeflacht ist. Die verschiedenen Möglichkeiten lassen sich auch kombinieren und der Algorithmus terminiert, sobald eine dieser Bedingungen erreicht ist. Da die Fitness von der bestmöglichen Lösung nicht bekannt ist, wurde meist die Variante eingesetzt, bei der die Anzahl an Generationen spezifiziert wird.

Anfangspopulation: Die Population, mit der ein Evolutionsprozess startet, wird entweder neu erstellt, wobei die Werte des Genoms zufällig generiert werden oder aber es werden die Werte eines Individuums aus einer CSV-Datei eingelesen und der Rest der Population durch die Mutation dieses Individuums aufgestockt. Dadurch lässt sich beispielsweise das Training unterbrechen und zu einem späteren Zeitpunkt fortsetzen. Ein weiterer Anwendungsfall wird in Abschnitt 3.5 beschrieben.

In der folgenden Tabelle 3.2 sind die verwendeten Parameter aufgelistet, die experimentell ermittelt und für die Ergebnisgewinnung verwendet wurden.

Parameter	Wert	Erklärung
populationSize	15	Anzahl Individuen pro Generation
generations	40	Durchgeführte Iterationen
elites	0,15	In nächste Generation übernommener Populationsanteil
selectionVictimRate	0,4	Von Fortpflanzung ausgeschlossener Populationsanteil
mutationRate	0,2	Wahrscheinlichkeit zur Mutation eines Individuums
geneMutationRate	0,4	Wahrscheinlichkeit zur Mutation eines Gens
stepSizeAlphas	0,05	Maximale Abweichung bei Mutation: $\pm 0,05 \cdot 3$
stepSizeVelocity	0,3	0,3 m/s $\approx$ 1 km/h. Maximale Abweichung: $\pm$3 km/h
allowedOC	5-8	Abbruchbedingung gewählt je nach Streckenkomplexität
penaltyOC	200	Priorisierung von OC-Vermeidung statt Rundenzeit
timeIfDNF	420 s	Bei nicht abgeschlossener Runde: 7 min Rundenzeit
maxCommandsStill	100	1 DriveCommand / 100 ms -> max. 10 s still stehen
maxCommandsOC	100	1 DriveCommand / 100 ms -> max. 10 s OC
penaltyAbort	1000	Strafe für abgebrochenen Run durch Stillstand etc.

Tabelle 3.2: Verwendete Parameter des genetischen Algorithmus.

3.4.4 Streckenspezifische Modellierung

Ziel dieser Modellierung ist das Finden der bestmöglichen Positionen der Wegpunkte und Geschwindigkeiten für eine konkrete Strecke.
Daher besteht das Genom der Individuen aus den Alpha- und Geschwindigkeitsparametern. Die Dimension des Individuums ist somit abhängig von der Hütchenpaar-Anzahl der einzelnen Strecke und variiert bei den Teststrecken zwischen 184 (bei 92 Hütchenpaaren) und 286 Parametern.

3.4.5 Streckenübergreifende Modellierung

Um die Anzahl der Parameter und damit die benötigte Simulations-Zeit möglichst gering zu halten, wurde sich für ein sehr kleines neuronales Netz entschieden.
Das Genom der Individuen umfasst in diesem Fall die Gewichte und Bias-Werte des neuronalen Netzes. Als Input erhält das neuronale Netz die Koordinaten der nächsten 10 Hütchenpaare, die relativ zur Auto-Position und Blickrichtung sind. Die zwei Outputneuronen geben den Alpha-Wert und die Zielgeschwindigkeit für das erste, nächstgelegene Hütchenpaar vor. Entsprechend haben die nächsten 10 Hütchenpaare Einfluss auf die aktuelle Fahrweise. Das neuronale Netz besitzt insgesamt 40 Input-Neuronen und 2

Output-Neuronen. Durch die Vollvermaschung ergeben sich 80 Gewichte und 2 Schwellwerte. Das neuronale Netz ist in Abbildung 3.10 dargestellt.
Alternativ kann die gesamte Strecke als Input gewählt werden. Das führt jedoch zu einer deutlichen Vergrößerung des neuronalen Netzes ($weights = 4 \cdot conePairs \cdot 2$). Darüber hinaus ist die Input-Größe in diesem Fall nicht mehr einheitlich, wodurch sich das neuronale Netz nur schwer auf verschiedene Strecken anwenden lässt.
Es galt einen Kompromiss zwischen der Weitsicht der Individuen und Größe ihres Genoms zu finden.
Evaluiert werden die Individuen, indem die Trajektorie anhand ihrer Gewichte und Biases berechnet und in der Simulation abgefahren wird. Dafür wird das Eingabefenster der zehn Hütchenpaare immer um eins versetzt und die Position des nächsten Wegpunktes und seine Zielgeschwindigkeit berechnet.
Jedes Individuum wird anhand drei verschiedener Strecken getestet. Die Fitness entspricht dabei dem Mittelwert der drei erreichten Werte. Innerhalb einer Generation misst sich die Population an den selben drei Strecken.
Als letzten Schritt muss der Output normalisiert werden, damit der berechnete α-Wert zwischen 0 und 1 und die Geschwindigkeit zwischen 0 und der Maximalgeschwindigkeit von 18 m/s liegt. Dies ist Beispielsweise über die Sigmoid-Aktivierungsfunktion möglich.

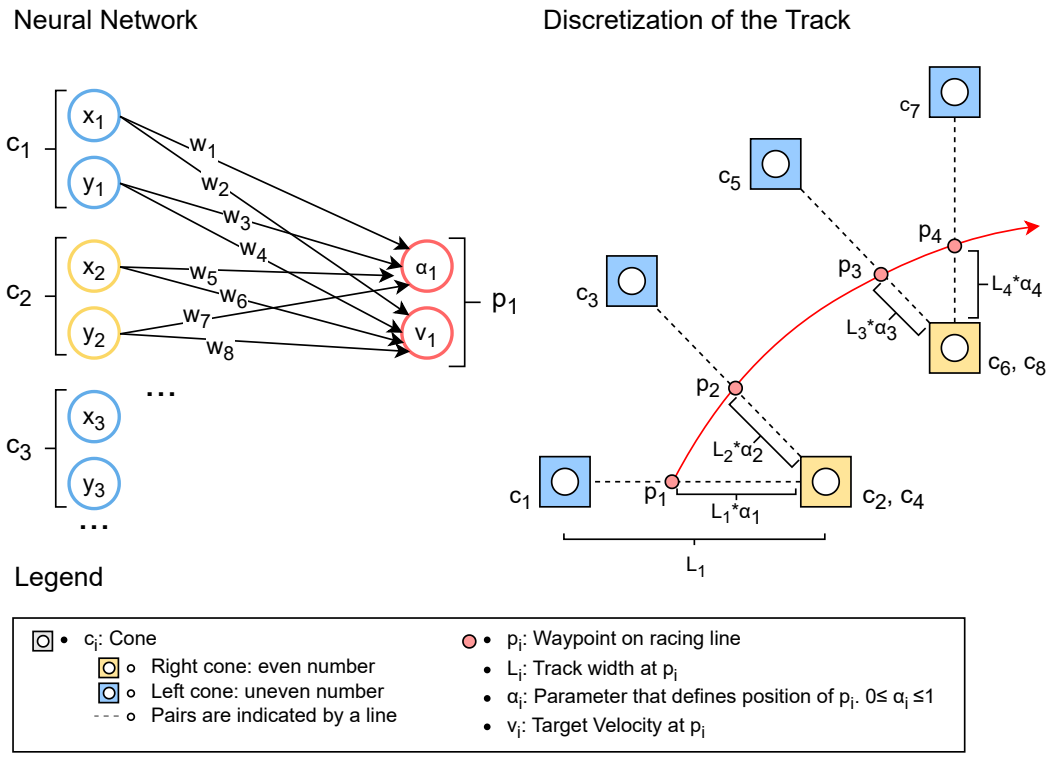


Abbildung 3.10: Darstellung des neuronalen Netzes

3.5 Hybrider Ansatz

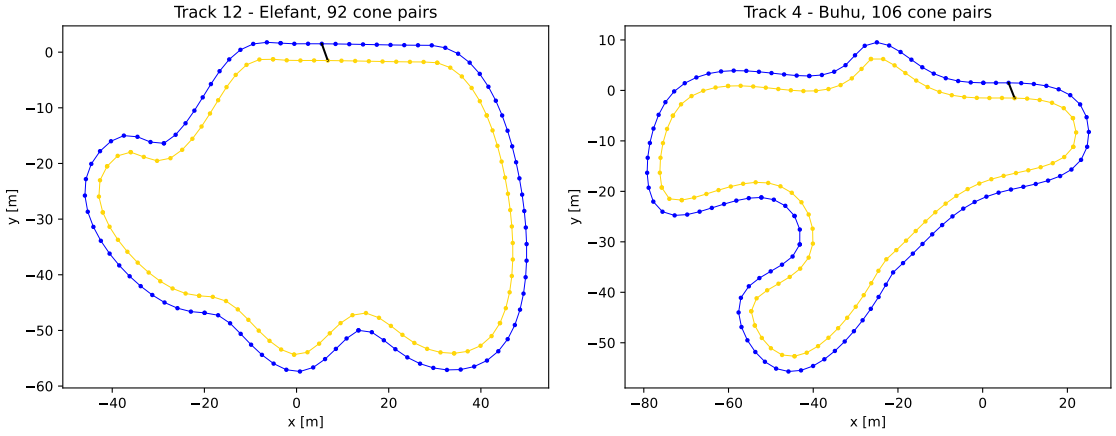
Der hybride Ansatz ist die Kombination aus der numerischen Berechnung und dem genetischen Algorithmus. Die Idee dahinter ist die Vereinigung der Vorteile der jeweiligen Ansätze: Der numerische Ansatz kommt beispielsweise schneller zu einem Ergebnis als der genetische Algorithmus. Dafür ist der genetische Algorithmus durch die Einbindung der Regelung in der Simulation realistischer.
Dafür wird für eine bestimmte Strecke erst einmal das Ergebnis bestehend aus Trajektorie und Zielgeschwindigkeiten zu jedem Wegpunkt numerisch berechnet und anschließend mit dem genetischen Algorithmus weiter optimiert. Die Werte aus der Berechnung werden das Genom für das erste Individuum aus der Population. Der Rest der Population

wird gebildet, indem das existierende Individuum mutiert wird.

Es hat sich als vorteilhaft erwiesen auch das zweite Individuum vorzugeben. Hierfür werden die gleichen Alpha-Werte wie für das erste Individuum verwendet. Die Geschwindigkeitswerte werden allerdings alle auf die gleiche Geschwindigkeit gesetzt. Ziel ist es, eine Geschwindigkeit zu wählen, mit der sich die Strecke abfahren lässt, ohne OC zu fahren. Zahlreiche Versuche haben ergeben, dass eine Geschwindigkeit bis zu 7 m/s zuverlässig für verschiedene Strecken funktioniert.
Dadurch gibt es ein Individuum in der Population, dass unter Umständen zwar langsamer fährt, dafür aber deutlich sicherer. Durch die Fitnessfunktion, die OC fahren bestraft, erhält dieses Individuum einen besseren Wert als diejenigen Individuen, die OC fahren und kann seine Gene weitergeben.
Im Laufe der Evolution setzen sich nach und nach Individuen durch, die nicht von der Strecke abfahren, während weiterhin die Rundenzeit optimiert wird. Über diese Methode können OC gezielt und schnell vermindert werden.

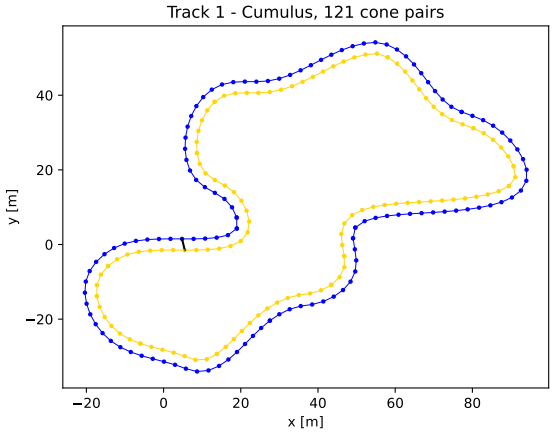
4 Ergebnisse

Für die Evaluierung der verschiedenen Ansätze wurden fünf Beispielstrecken mit steigendem Schwierigkeitsgrad ausgewählt, um an ihnen die Ergebnisse vergleichen zu können. Sie sind in Abbildung 4.1 dargestellt.
Als Bewertungskriterien werden neben der Rundenzeit auch die Zeitstrafen betrachtet, um die Punktevergabe auf den FS Events zu berücksichtigen. Darüber hinaus wird auch die Rechenzeit miteinbezogen, die zur Ergebniserstellung nötig ist.

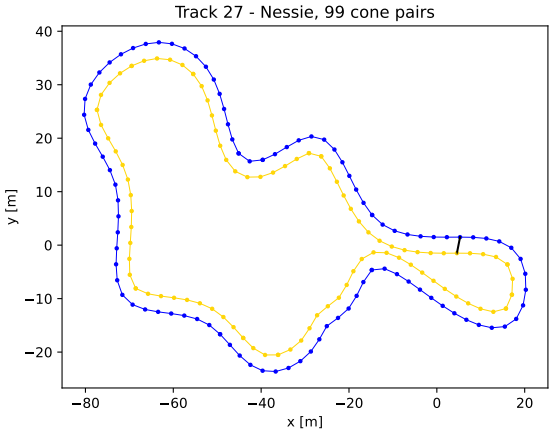


(a) Teststrecke 12 - Elefant

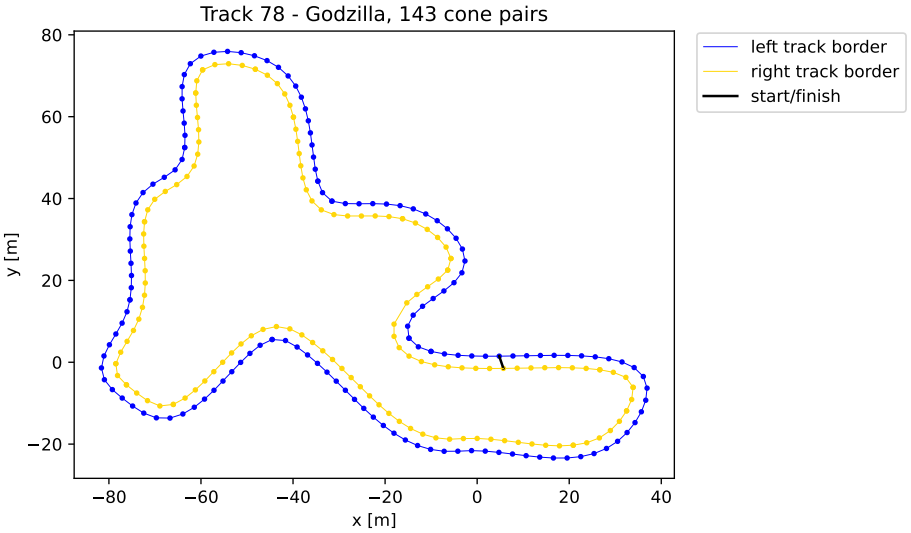
(b) Teststrecke 04 - Buhu



(c) Teststrecke 01 - Cumulus



(d) Teststrecke 27 - Nessie



(e) Teststrecke 78 - Godzilla

Abbildung 4.1: Verwendete Teststrecken für die Ergebnisevaluierung.

4.1 Ergebnisse der numerischen Berechnung

Bei dem Ansatz der numerischen Berechnung wird für eine Strecke zunächst die Trajektorie mit der geringsten Krümmung ermittelt und im Anschluss die Geschwindigkeiten anhand der Kurvenradien und Fahrzeugspezifikationen beim Velocity Planning (VP) ermittelt. Die folgende Tabelle 4.1 enthält die berechneten Rundenzeiten für jede Strecke und die benötigte Rechendauer zum Ausführen der Algorithmen. Die Ergebnisse sind in Abbildung A.6 dargestellt.

	12-Elefant	04-Buhu	01-Cumulus	27-Nessie	78-Godzilla
Rundenzeit Berechnung	21,33 s	26,962 s	29,055 s	25,213 s	36,241 s
Rechenzeit Trajektorie	0,4889 s	0,8065 s	1,2479 s	0,5492 s	1,8441 s
Rechenzeit VP	0,001 s	0,001 s	0,002 s	0,001 s	0,002 s
Rechenzeit Gesamt	0,4899 s	0,8075 s	1,2499 s	0,5502 s	1,8461 s

Tabelle 4.1: Ergebnisse des numerischen Ansatzes mit Rechenzeiten

An diesen Ergebnissen ist zu erkennen, dass die benötigte Rechenzeit steigt, je länger die Strecke ist und entsprechend je mehr Wegpunkte bestimmt werden müssen. Den Hauptanteil der Laufzeit nimmt dabei die Optimierung der Trajektorie in Anspruch.
Die Berechnungsdauer liegt selbst bei der Strecke 78, welche mit 143 Hütchenpaaren die längste Teststrecke darstellt noch unter 2 Sekunden. Damit liefert dieser Ansatz seine Ergebnisse in einer Schnelligkeit, die ihn für den Einsatz auf der Rennstrecke eignet.

Bevor die Rundenzeiten der numerischen Berechnung eingeordnet und mit den Ergebnissen der anderen Ansätze verglichen werden können, muss erst einmal sichergestellt werden, dass die berechneten Rundenzeiten mit denen aus der Simulation vergleichbar sind.
Eine Vergleichbarkeit kann festgestellt werden, indem ein Simulationsdurchlauf der FSDS gestartet wird mit der Vorgabe die generierte Trajektorie mit den Geschwindigkeiten aus dem Velocity Planning abzufahren. Die Ergebnisse dieses Testlaufs sind in der Tabelle 4.2 festgehalten.

Die Erkenntnis aus diesem Vergleich ist, dass die Rundenzeiten aus der FSDS mindestens doppelt so lang sind wie die aus der Rundenzeitberechnung. Aus der Tabelle ist außerdem zu entnehmen, dass das Auto in der Simulation der Trajektorie bei den vorgegebenen Geschwindigkeiten gar nicht folgen kann, wodurch zahlreiche Hütchen umgefahren werden und es auch mehrfach zum Verlassen der Strecke kommt.

	12-Elefant	04-Buhu	01-Cumulus	27-Nessie	78-Godzilla
Rundenzeit Berechnung	21,33 s	26,962 s	29,055 s	25,213 s	36,241 s
Rundenzeit FSDS	51,554 s	74,422 s	76,349 s	65,985 s	89,405 s
DOO FSDS	28	15	24	28	29
OC FSDS	3	4	6	6	5
Zeit + Strafen FSDS	137,554 s	144,422 s	184,349 s	181,985 s	197,405 s

Tabelle 4.2: Experiment zur Vergleichbarkeit der Rundenzeiten. Visualisierung ab Abbildung A.8

Einen Grund für diese Abweichung stellt das Fahrzeugmodell aus der Simulation dar, dessen Spezifikationen sich stark vom HAWKS-Auto unterscheiden. Das FSDS-Fahrzeug ist beispielsweise 40 kg schwerer und hat lediglich einen Luftwiderstandsbeiwert von 0,3. Dadurch hat das Auto eine geringere Kurvenhöchstgeschwindigkeit und entschleunigt langsamer.
Durch die Anpassung der Werte in der Rundenzeitsimulation des GTO and die Werte aus der Dokumentation der FSDS (siehe 4.4), konnten die Rundenzeiten und Zeitstrafen verbessert werden, wie an Tabelle 4.3 zu sehen ist. Abbildung A.7 stellt die Ergebnisse des Velocity Plannings mit den FSDS Parametern dar.

	12-Elefant	04-Buhu	01-Cumulus	27-Nessie	78-Godzilla
Rundenzeit Berechnung	24,322 s	30,416 s	31,502 s	27,645 s	39,559 s
Rundenzeit FSDS	58,578 s	81,992 s	78,601 s	68,461 s	97,479 s
DOO FSDS	16	16	28	22	21
OC FSDS	2	4	6	4	2
Zeit + Strafen FSDS	110,578 s	153,992 s	194,601 s	152,461 s	159,479 s

Tabelle 4.3: Rundenzeiten mit Parametern des FSDS-Autos. Visualisierung ab Abbildung A.13

Wie man in Tabelle 4.3 sehen kann, sorgen die hohen Geschwindigkeiten an manchen Stellen noch immer für Off-Courses. Zusätzlich kommt es durch die Regelung zu weiteren Abweichungen zwischen Soll- und Ist-Trajektorie.

Konstante	Bezeichnung	Eingesetzter Wert
g	Erdbeschleunigung	9,81 m/s^2
μ	Reifenreibwert in Querrichtung	1,76
m	Fahrzeugmasse	**255** kg
p_L	Luftdichte	1,225 kg/m^3
c_A	Abtriebsbeiwert (engl. lift coefficient)	**2,5**
c_W	Luftwiderstandsbeiwert (engl. drag coefficient)	**0,3**
A_V	Querspantfläche	1 m^2
k_m	Drehmassenzuschlagsfaktor	1,2
P_M	Motorleistung	108 kW
η	Wirkungsgrad Antriebsstrang	0,88
k_R	Rollwiderstandsbeiwert	0,013
$F_{W,Z}$	Radlast	**625** N
w	Abstand zur Streckenbegrenzung	**1** m

Tabelle 4.4: Auf die FSDS angepasste Rundenzeit-Parameter

4.2 Ergebnisse des genetischen Algorithmus

In Tabelle 4.5 sind die Ergebnisse verzeichnet, die mithilfe des genetischen Algorithmus erzielt wurden. Zu der Zeit der Ergebniserhebung war die Mutation noch zufällig und nicht über eine Schrittgröße gesteuert (siehe 3.4.3).

	12-Elefant	04-Buhu	01-Cumulus	27-Nessie	78-Godzilla
Rundenzeit	58,028 s	76,132 s	80,002 s	68,883 s	117,456
DOO	7	12	26	14	30
OC	0	2	2	4	4
Zeit + Strafen	72,028 s	120,132 s	152,002 s	136,883 s	217,456 s
Rechenzeit	13,43 h	16,73 h	14,35 h	13,5 h	20,37 h

Tabelle 4.5: Ergebnisse des genetischen Algorithmus. Visualisierung ab Abbildung A.18

Auffällig an diesen Ergebnissen ist insbesondere die Rechenzeit, die mehrere Stunden umfasst. Die hohe Zeitanforderung lässt sich über die Eigenschaften der FSDS erklären. Diese Simulation rechnet in Echtzeit und bietet auch keine Möglichkeit, um die Rechenschritte zu beschleunigen. In einer Simulationsumgebung, dessen Rechengeschwindigkeit nicht durch die Visualisierung limitiert ist, können die Zeiten sehr viel geringer ausfallen. Nichts desto trotz wird dieses Verfahren durch die wiederholte Evaluierung der Population mehr Rechenleistung als der numerische Ansatz in Anspruch nehmen.
Die Ergebnisse zeigen außerdem, dass es, insbesondere bei leichten Strecken ohne vieler

enger Kurven wie zum Beispiel der Elefanten-Strecke, möglich ist gute Rundenzeiten mit wenigen Zeitstrafen zu erzielen. Bei vier der fünf Teststrecken sind die Rundenzeiten außerdem schneller als die Zeiten von dem Berechnungs-Ansatz aus der FSDS.
Wenn man die Bewertungen der besten Individuen jeder Generation betrachtet, lässt sich die graduelle Verbesserung erkennen. In der Abbildung A.28 sind beispielhaft die Meilensteine des Evolutionsprozesses von Strecke 4 dargestellt.
Es wird sichtbar, dass die Weiterentwicklung am Anfang des Evolutionsprozesses stärker ist als am Ende. Der grundlegende Lernprozess von der Streckengeometrie spielt sich in den ersten 6 Generationen ab und es müssen weitere 43 Generationen vergehen, bis ein weiterer OC vermieden werden kann. Des weiteren schafft der Algorithmus es bei den meisten Strecken nicht gänzlich die Off-Courses wegzuoptimieren.
Das liegt nicht nur an der begrenzten Trainingszeit sondern vor allem an der Komplexität des Problems. Um einen Off-Course zu verhindern müssen in erster Linie die Geschwindigkeitsparameter aufeinander abgestimmt werden, aber auch die Position der Wegpunkte kann ausschlaggebend sein. In einem Individuum müssen demnach viele passende Parameter vereint sein. Oft passiert es jedoch, dass bei der Mutation eines Individuums zwar der eine Streckenabschnitt vorteilhaft abgeändert wird, die Änderungen an einem anderen Abschnitt jedoch weniger gut ausfallen. Dadurch erhält das Individuum insgesamt eine gleiche oder sogar schlechtere Bewertung als das bisher beste Ergebnis und die positiven Teiländerungen kann sich somit nicht oder nur schwer durchsetzen.

4.3 Ergebnisse des hybriden Ansatzes

Als Hauptproblem der beiden vorherigen Ansätze werden die Off-Courses angesehen. Die Strecke zu verlassen gilt zwar nicht als Regelverstoß, sofern die Strecke dadurch nicht abgekürzt wird, jedoch zieht jedes Vorkommen eine Zeitstrafe nach sich. Zudem sind Off-Courses mit einem Sicherheitsrisiko verbunden, da abseits der Strecke unter Umständen Absperrungen oder Messtechnik aufgebaut sind. Aus Sicherheitsgründen müssen Durchläufe daher oft vorzeitig abgebrochen werden. Der Fokus für den hybriden Ansatz liegt daher in der Vermeidung der Off-Courses.
Darüber hinaus ist die Hoffnung, dass durch die Verwendung des Berechnungs-Ergebnisses als Ausgangslage für den genetischen Algorithmus die Zielergebnisse schneller erreicht werden können.

	12-Elefant	04-Buhu	01-Cumulus	27-Nessie	78-Godzilla
Rundenzeit	63,581 s	82,671 s	92,374 s	71,754 s	111,287 s
DOO	5	3	9	11	11
OC	0	0	0	0	0
Zeit + Strafen	73,581 s	88,671 s	110,374 s	93,754 s	133,287 s

Tabelle 4.6: Ergebnisse des hybriden Ansatzes. Visualisierung ab Abbildung A.23

Wie man an der Tabelle 4.6 erkennen kann, gelingt es bei allen Strecken komplett dem Verlauf der Strecke zu folgen ohne von ihr abzuweichen. Dies liegt an dem vorgegebenen Individuum, welches mit einer konstanten Geschwindigkeit von 7 m/s die Strecke abfährt. Dadurch enthält die Population von Anfang an ein Individuum, welches der Strecke folgen kann. Alle Verbesserungen in den folgenden Generationen beziehen sich somit auf die Rundenzeit und Vermeidung von Hütchen-Umwürfen.
Die Population tastet sich nach und nach an höhere Geschwindigkeiten heran, wobei die Gene des Individuums, bestehend aus den Werten der Trajektorie mit der geringsten Krümmung und den Geschwindigkeiten aus der Rundenzeitsimulation zum Tragen kommen.
Insgesamt können mit dem hybriden Ansatz die besten Rundenzeiten und auch Zeiten mit Zeitstrafen erzielt werden. Nur bei der Elefanten-Strecke ist der reine genetische Algorithmus um 1,553 Sekunden schneller. Das kann auf die konservativere Fahrweise des hybriden Ansatzes oder den Zufall in der Mutation zurückgeführt werden.
Bei längeren Strecken mit engeren, dicht aufeinanderfolgenden Kurven wird der Vorteil des hybriden Ansatzes umso deutlicher. Die Rundenzeit inklusive der Zeitstrafen von der Strecke 78 ist um 84,169 Sekunden schneller als der genetische Algorithmus und 26,192 Sekunden schneller als die numerische Berechnung (FSDS-Zeit).

5 Zusammenfassung und Ausblick

In diesem Kapitel sollen die gewonnenen Erkenntnisse zusammengefasst werden und ein Ausblick gegeben werden, wie die Ergebnisse zukünftig weiter entwickelt werden können.

5.1 Fazit

In dieser Arbeit wurden insgesamt drei Ansätze zur Rundenzeitoptimierung dargestellt, welche allesamt vielversprechende Ausgangspunkte für den Einsatz auf den Formula Student Events erbracht haben.
Insbesondere der hybride Ansatz konnte als Kombination der effizienten Minimierung der Trajektorienkrümmung und der Fähigkeit des genetischen Algorithmus sich an die Regelung des Fahrzeugs anzupassen überzeugen. Er lieferte zudem die insgesamt schnellsten Rundenzeiten. Ein weiterer großer Vorteil des Ansatzes ist die Zuverlässigkeit durch die Priorisierung der Off-Course-Vermeidung.
Die dennoch lange Laufzeit des Evolutionsprozesses kann durch die Wahl einer beschleunigten (Faster-than-Real-Time) Simulationssoftware verbessert werden. Es wäre dabei außerdem von Vorteil, wenn das Fahrzeugmodell der Simulation personalisierbar ist, um die Diskrepanz zwischen Modell und Realität weiter zu einzuschränken.
Die schlussendliche Eignung und möglicher Verbesserungsbedarf der betrachteten Ansätze wird sich beim Testen mit dem realen Fahrzeug NOVA zeigen.

5.2 Ausblick

Wie bereits angesprochen, ist der nächste Schritte im Vorhaben der Rundenzeitoptimierung die Validierung der bisherigen Ergebnisse auf dem Fahrzeug. Hierfür kann man eine der betrachteten Strecken genau nachbauen und überprüfen ob NOVA den Anweisungen

folgen kann. Sollten an dieser Stelle Schwierigkeiten ersichtlich sein, wäre es sinnvoll eine neue beschleunigte Simulationsumgebung mit personalisierbaren Fahrzeugmodell auszuprobieren.

Im Anschluss sollen drei weitere Konzepte skizziert werden, dessen Untersuchung sich noch lohnen würde, die aus zeittechnischen Gründen jedoch nicht den Weg in diese Arbeit gefunden haben.

Streckenübergreifende Modellierung des genetischen Algorithmus
Dieses Vorhaben wurde bereits ausführlich in 3.4.5 beschrieben. Der Ansatz ist außerdem fast gänzlich programmiert und in das bestehende System eingebettet. Aufgrund der langen Trainingsphase, wurde die Priorität allerdings bewusst auf die streckenspezifische Modellierung gesetzt.
Der große Vorteil an dieser Implementierung ist, dass die Rechenzeit zur Fahrzeit selber nur minimal ist, da die Gewichte und Bias-Werte des neronalen Netzes in einem vorausgegangenen Prozess bestimmt werden. Beim Wettkampf selber müsste lediglich der Output des Netzes berechnet werden.
Welche Topologie und Größe des Netzes in diesem Fall am geeignetsten ist, muss experimentell ermittelt werden.

Optimierung in Abschnitten
Diese Idee knüpft an ein Problem des genetischen Algorithmus an. Und zwar gibt es aufgrund von der hohen Anzahl an Parametern das Phänomen, dass einige Individuen zwar in einigen Streckenabschnitten bessere Lösungen liefern, diese Teillösungen sich aber nicht durchsetzen, da die insgesamte Leistung des Individuums als zu schlecht für die Rekombination eingestuft wird.
Der Vorschlag sieht vor, dass die Strecke vor dem Optimierungsprozess in Abschnitte unterteilt wird und jedes Individuum einen Fitnesswert pro Abschnitt erhält. Die Selektion und Rekombination erfolgt ebenfalls abschnittsweise. Demnach kann ein Abschnitt des Individuums aufgrund des Fitnesswertes für die Erstellung von Kind-Individuen selektiert werden und ein anderer nicht.
Die Vermutung bei diesem Ansatz ist, dass früher im Evolutionsprozess bessere Lösungen erzielt werden können.

Weiterentwicklung des numerischen Ansatzes
Möchte man sich aufgrund von der hohen Effizienz auf den numerischen Ansatz konzentrieren, lassen sich auch hier Verbesserungen finden.

Der Auslöser für Off-Courses liegt in den zu hohen Kurvengeschwindigkeiten. Identifiziert man in der Simulation oder auch beim Befahren der Strecke Stellen, bei denen der Rennwagen aufgrund der Geschwindigkeiten von der Soll-Trajektorie abweicht, kann an diesen Wegpunkten die Geschwindigkeit gezielt reduziert werden. Dadurch kann vor dem Lauf oder für die folgenden Runden im Trackdrive das Risiko für Off-Courses gesenkt werden.
Alternativ wäre es auch denkbar mit einer insgesamt geringen Geschwindigkeit die Strecke zu befahren und die Geschwindigkeiten immer weiter anzuheben bis kleine Abweichungen von der Soll-Trajektorie entstehen und das zuletzt stabile Tempo beizubehalten.

Zusammenfassend lässt sich sagen, dass dieses Thema noch viel Raum für die Entwicklung neuer Ideen und auch potenzielle Verbesserungen lässt.
Diese Arbeit soll als Grundlage und Einblick in die vielfältigen Möglichkeiten der Rundenzeitoptimierung autonomer Rennfahrzeuge in der Formula Student dienen.

Abkürzungsverzeichnis

AS Autonomous System
CFD Computational Fluid Dynamics
CSV Comma-Separated Values
DV Driverless
DOO Down or out. Anzahl umgefahrener Hütchen in der Formula Student
DNF Did not finish. Run wurde nicht regelkonform beendet
EA Evolutionärer Algorithmus
FS Formula Student
FSDS Formula Student Driverless Simulator [6]
FSG Formula Student Germany
GA Genetischer Algorithmus
GTO Global Trajectory Optimizer. Das im Rahmen dieser Bachelorarbeit entwickelte System
HAW Hochschule für angewandte Wissenschaften
HAWKS HAWKS Racing, das FS Team der Hochschule für angewandte Wissenschaften (HAW) Hamburg
LiDAR Light Detection and Ranging
NOVA Das 19. HAWKS-Auto
OC Off Course. Alle 4 Reifen befinden sich außerhalb der Streckenbegrenzung
ROS Robot Operating System
RTG Random Track Generator [12]
SLAM Simultaneous Localization and Mapping
SLSQP Sequential Least Squares Programming
UML Unified Modeling Language
VP Velocity Planning

Literaturverzeichnis

[1] Dario Floreano, Peter Dürr und Claudio Mattiussi: Neuroevolution: from architectures to learning. (2008). – URL https://link.springer.com/content/pdf/10.1007/s12065-007-0002-4.pdf

[2] Bart van de Poel: Raceline Optimization. In: *Bachelor Thesis Universiteit van Amsterdam* (>=2007). – URL https://staff.fnwi.uva.nl/b.bredeweg/pdf/BSc/20082009/vandePoel.pdf

[3] Braghin, F. ; Cheli, F. ; Melzi, S. ; Sabbioni, E.: *Race driver model*. Computers and Structures, 2007. – URL https://www.sciencedirect.com/science/article/abs/pii/S0045794908000163

[4] Bundesministerium für Digitales und Verkehr: Die Zukunft fährt autonom. (2024). – URL https://www.bmv.de/SharedDocs/DE/Publikationen/DG/die-zukunft-faehrt-autonom.pdf?__blob=publicationFile

[5] Cleland, Benjamin G.: Reinforcement Learning for Racecar Control. (2006). – URL https://api.semanticscholar.org/CorpusID:8674882

[6] Formula Driverless Competition Simulator Contributors: Formula Student Driverless Simulation Dokumentation. (2020). – URL https://fs-driverless.github.io/Formula-Student-Driverless-Simulator/v2.2.0/

[7] Formula Student Germany: *Formula Student Rules 2025 Version 1.1*. FSG, 2025. – URL https://www.formulastudent.de/fsg/rules

[8] Formula Student Germany: *FSG Competition Handbook 2025 Version 1.0*. FSG, 2025. – URL https://www.formulastudent.de/fsg/rules

[9] Free Software Foundation: java-rosbridge. (2007). – URL https://github.com/h2r/java_rosbridge

[10] Jacob Olausson, Jacob Larsson: Optimal Control and Race Line Planning for an Autonomous Race Car. (2021). – URL https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1569136&dswid=3875

[11] KA-RaceIng, Karlsruhe Institute of T.: The Software Stack That Won the Formula Student Driverless Competition. (2022). – URL https://doi.org/10.48550/arXiv.2210.10933

[12] Löben Sels, Mees van: Random Track Generator. (2024). – URL https://github.com/mvanlobensels/random-track-generator

[13] Marko Jurić: Optimization of motion trajectories for autonomous racing vehicles. (2024). – URL https://zir.nsk.hr/islandora/object/fer%3A13273/datastream/PDF/view

[14] Michel Wegner: Entwicklung einer echtzeitfähigen Softwarearchitektur für einen autonom fahrenden Formula Student Rennwagen. In: *Masterthesis, HAW Hamburg* (2024). – URL https://reposit.haw-hamburg.de/global-search?query=michel+wegner

[15] Shimrat, M.: Algorithm 112: Position of point relative to polygon. (1962). – URL https://doi.org/10.1145/368637.368653

[16] Thomas Bartz-Beielstein, Jürgen Branke, Jörn Mehnen und Olaf Mersmann: Evolutionary Algorithms. (2014). – URL https://www.researchgate.net/publication/261842296_Evolutionary_Algorithms

[17] Tran, Bao: Autonomous Vehicle Testing: Top Countries and Cities Leading the AV Revolution (Latest Stats). (2025). – URL https://patentpc.com/blog/autonomous-vehicle-testing-top-countries-and-cities-leading-the-av-revolution-latest-stats?utm_source=chatgpt.com

[18] Trzesniowski, Michael: *Rennwagentechnik*. Bd. 4. Auflage. Springer Vieweg, 2014. – ISBN 978-3-658-04918-80

A Anhang

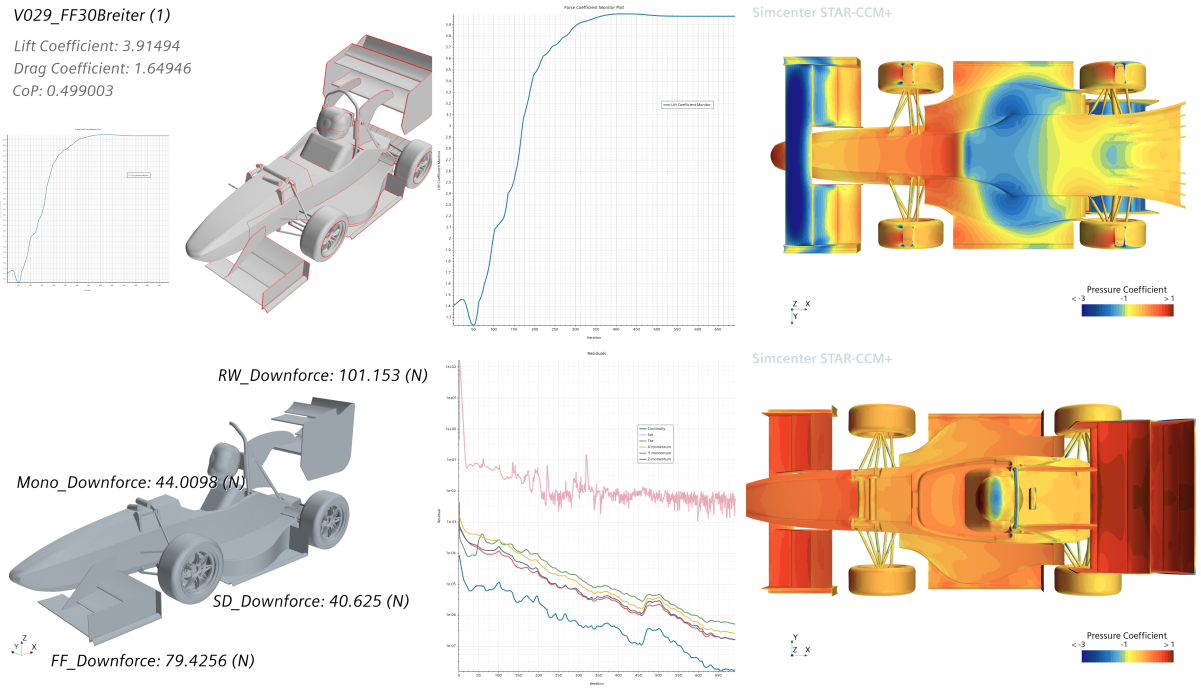
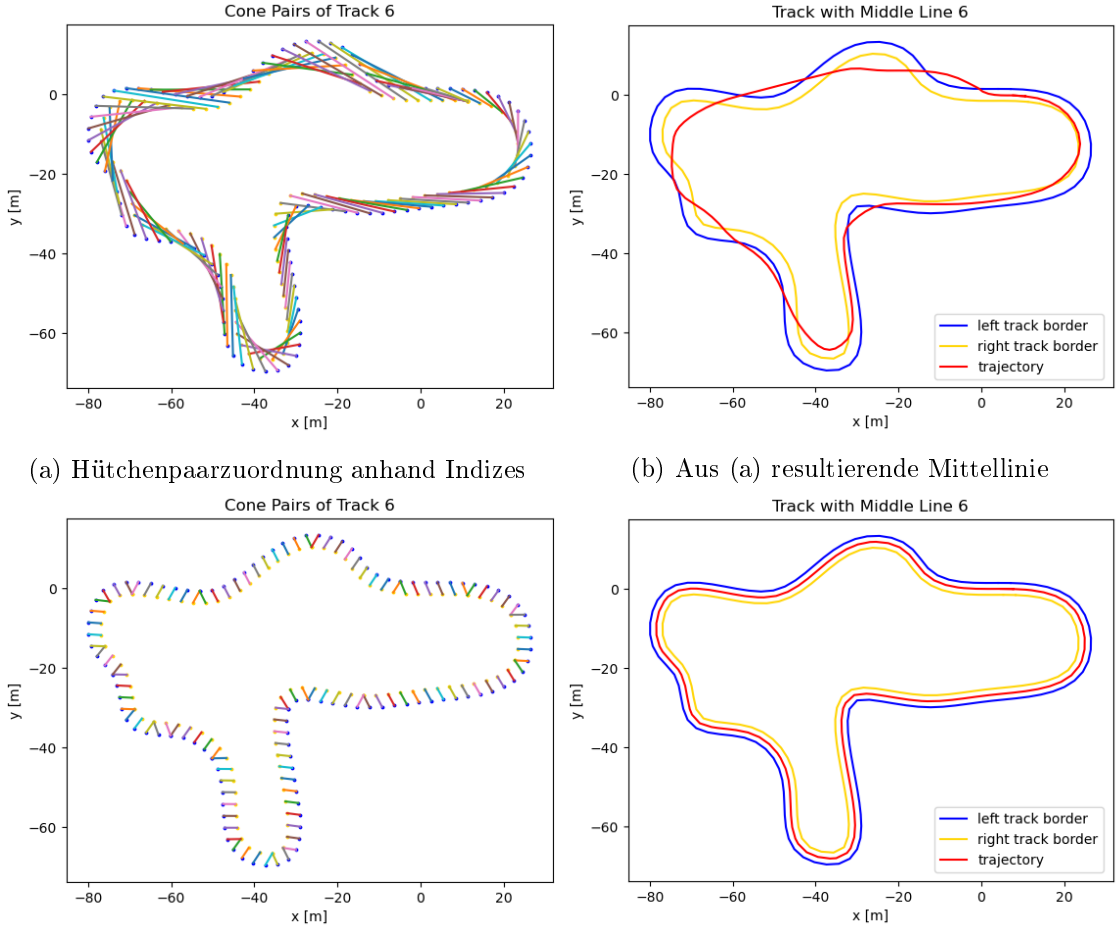


Abbildung A.1: Beispielhaftes Ergebnis aus einer CFD-Simulation für den Frontflügel



(a) Hütchenpaarzuordnung anhand Indizes

(b) Aus (a) resultierende Mittellinie

(c) Hütchenpaarzuordnung anhand Algo. 2

(d) Aus (c) resultierende Mittellinie

Abbildung A.2: Veranschaulichung weshalb eine Hütchenpaar-Zuordnung benötigt wird.

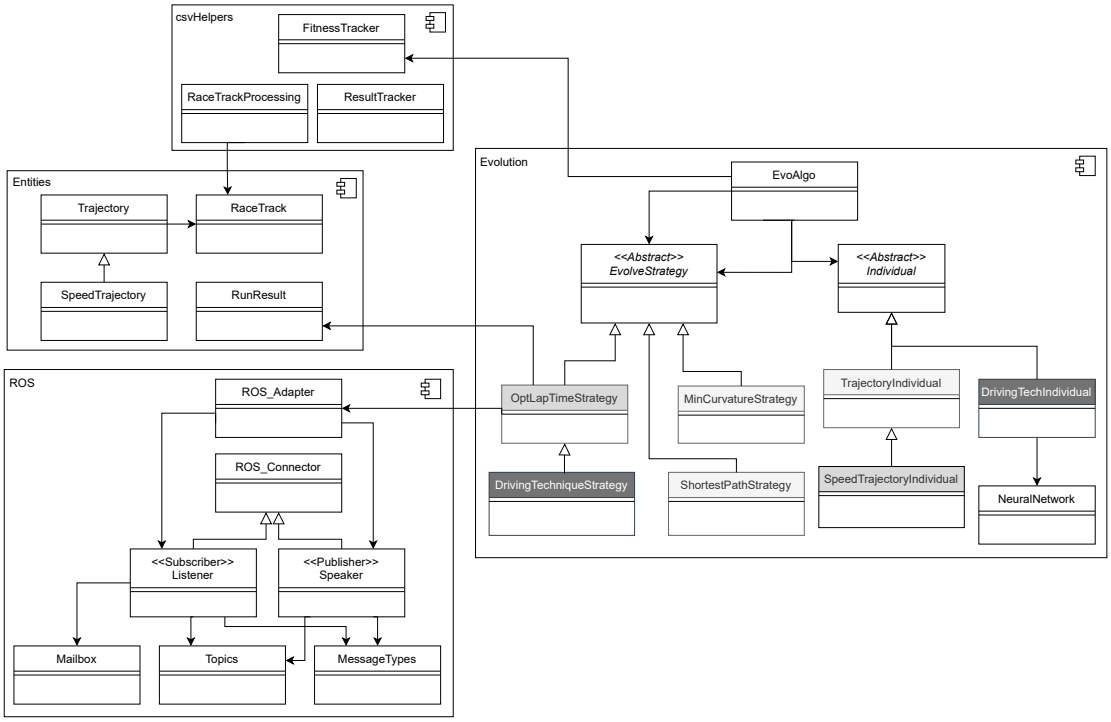


Abbildung A.3: Überblick über den Java-Anteil des GTO Projekts

Algorithm 4 Message Exchange in $fsdsRun(mapIndex)$

for each *individual* **do**
  while $\neg gotGoSignal() \vee \neg isSimReady()$ **do**
    $publishNewIndividualRequest(mapIndex)$
  end while
  $lapCompleted = false$
  $driving = true$
  while $\neg lapCompleted$ **do**
    $publishMissionState(lapCompleted)$
    $publishASState(driving)$
    $carPosition = getCarPosition()$
    $waypoints = getNearestWaypoints(carPosition)$
    $publishTrajectory(waypoints)$
    if $getLapCount() \geq 1$ **then**
      $lapCompleted = true$
    end if
  end while
  $publishFinishedSignal()$
end for

Message	Topic	Message Type	Publisher	Subscriber(s)	Use
Mission	/can_-bus/ctrl/from_-can_bus/DIU_-AMI_Request	hawks_can_ms-gs/DIU_AMI_-Request	Setup script (ja-va_rosbridge)	DV Software	Sent once during setup to choose the mission (6-autoX)
Go Signal	/fsds/signal/go	fs_ms-gs/GoSignal	FSDS	GTO	FSDS signals that it is ready for dri-ving commands
Simulation Ready	/sim_ready	std_msgs/String	Setup Script	GTO	Signals that running FSDS instance is new and not an old one.
Map Index	/next	std_msgs/String	GTO	Setup Script	The FSDS gets started with that track.
Mission State	/system_super-vision/mission_-state	hawks_ms-gs/Mission_-State	GTO	DV Software	Indicates that the mission is not fi-nished. Else the car would not mo-ve.

Tabelle A.1: Über ROS ausgetauschte Nachrichten Teil 1

Message	Topic	Message Type	Publisher	Subscriber(s)	Use
AS State	/can_-bus/ctrl/from_-can_-bus/ASCU_-Control	hawks_can_ms-gs/ASCU_Con-trol	GTO	DV Software	Sets the AS State to driving: AS-CU_AS_State = 2, ASCU_-ASMS_Status = 1. Else car does not move
Car Position	/fsds/testing_-only/odom	nav_ms-gs/Odometry	FSDS	GTO	Coordinates of the car. This will later be given by the localization
Trajectoy	/waypoint_list	hawks_ms-gs/Trajectory	GTO	DV Software	List of waypoints (coordinates and target velocity)
Lap Time Info	/fsds/testing_-only/extra_info	fs_ms-gs/ExtraInfo	FSDS	GTO	Lap time and DOOs
Finished Signal	/fsds/signal/finished	fs_ms-gs/FinishedSignal	GTO	FSDS	Shuts down FSDS_rosbridge and Scripts

Tabelle A.2: Über ROS ausgetauschte Nachrichten Teil 2

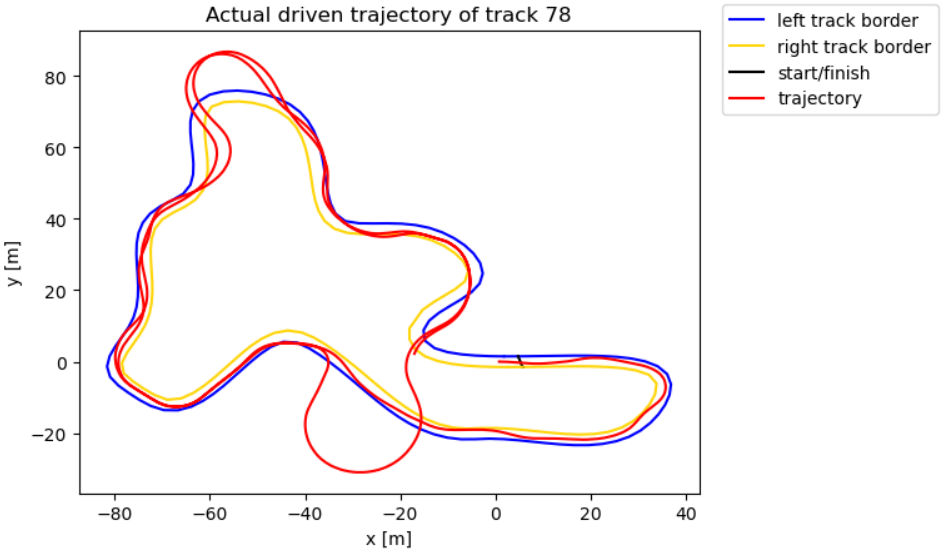


Abbildung A.4: Weiterfahrt nach OC führt zu Abkürzung des Tracks

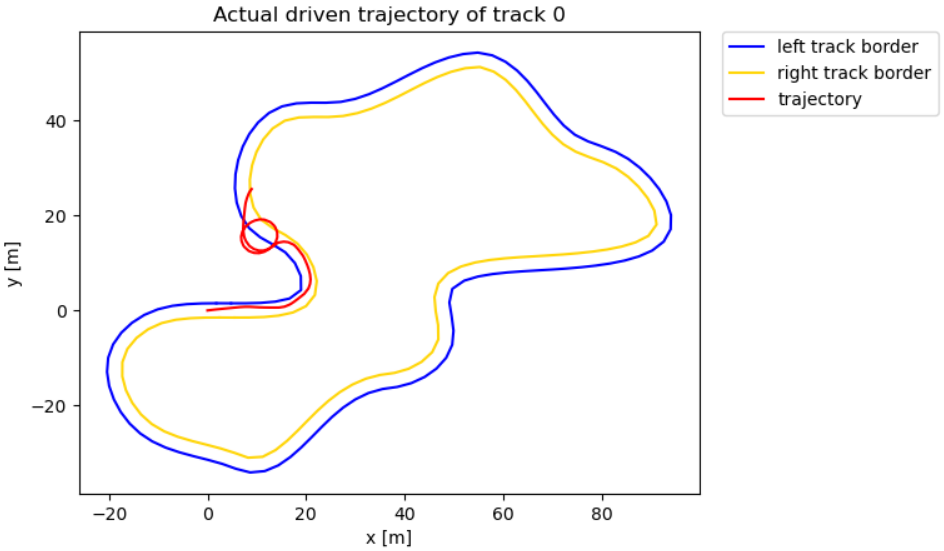
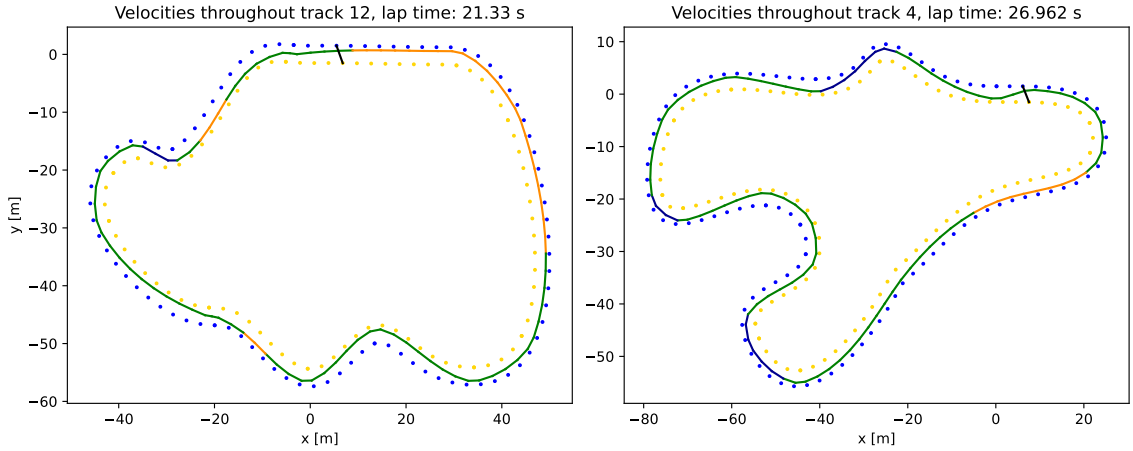
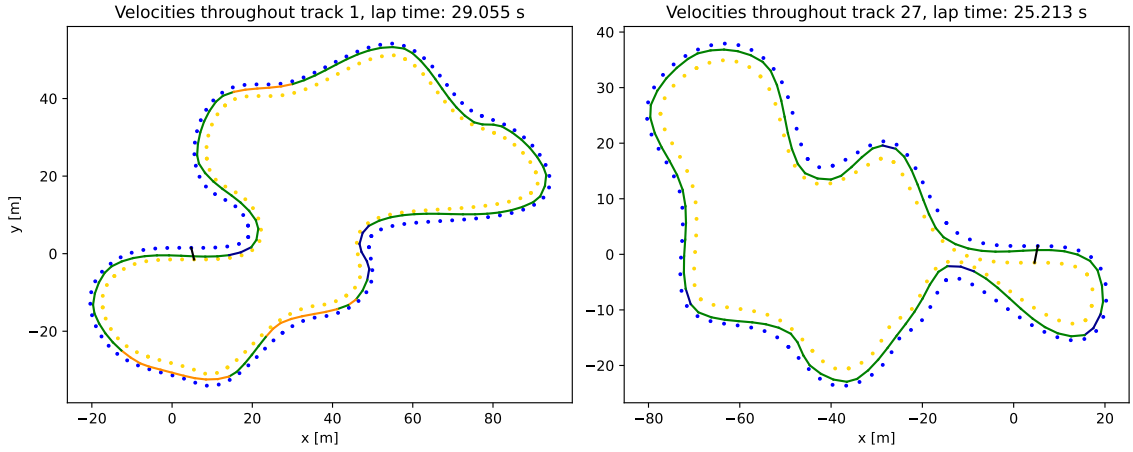


Abbildung A.5: Veraltete Wegpunkte sorgen für Kreisfahrten



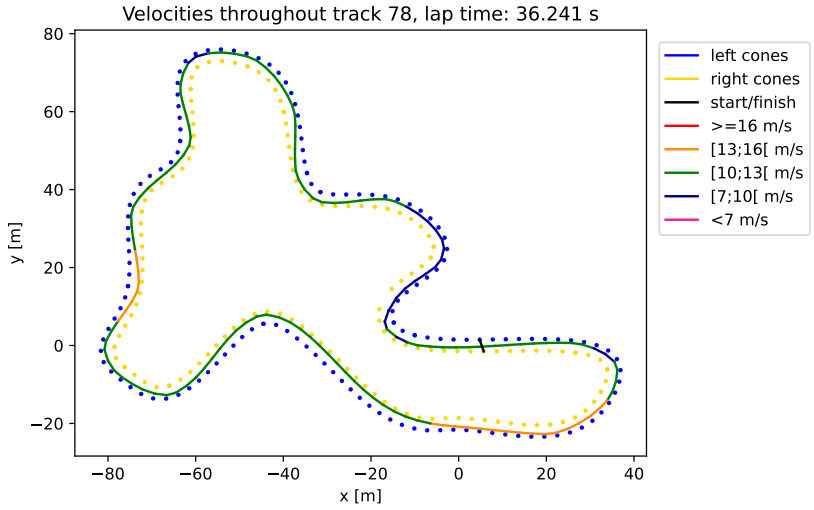
(a) Trajektorie mit Geschwindigkeiten 12

(b) Trajektorie mit Geschwindigkeiten 04



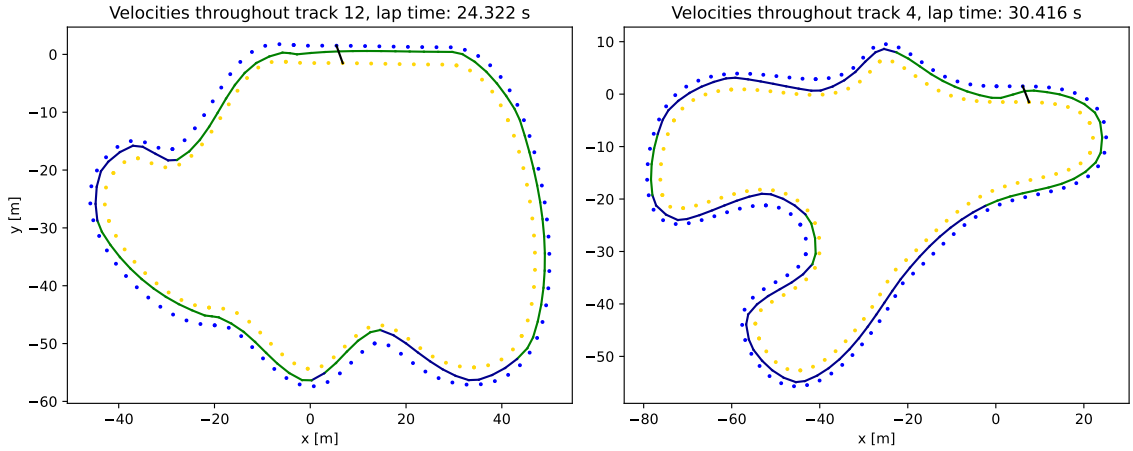
(c) Trajektorie mit Geschwindigkeiten 01

(d) Trajektorie mit Geschwindigkeiten 27



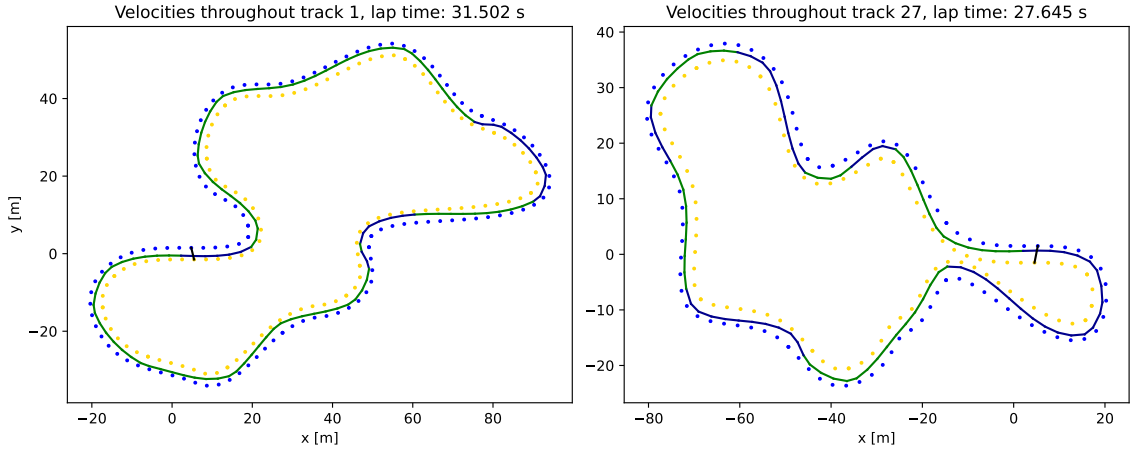
(e) Trajektorie mit Geschwindigkeiten 78

Abbildung A.6: Ergebnis der numerischen Berechnung mit NOVA-Parametern. Die Rundenzeiten sind vom Velocity Planning berechnet.



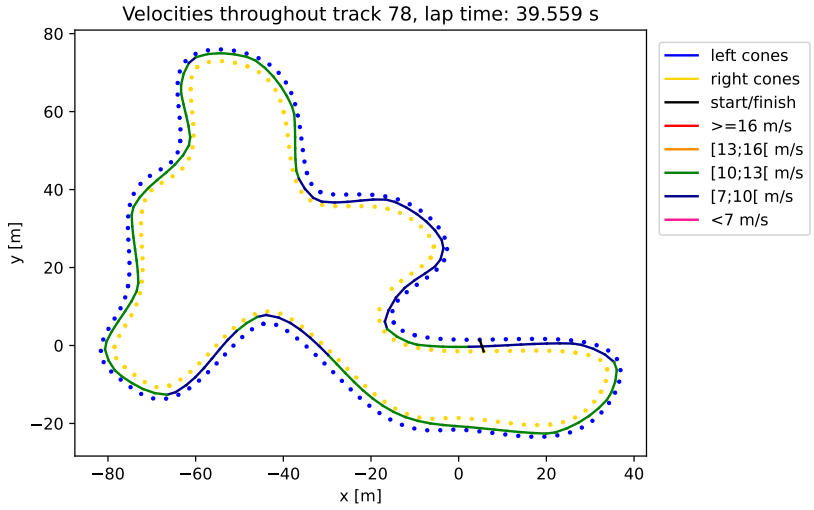
(a) Trajektorie mit Geschwindigkeiten 12

(b) Trajektorie mit Geschwindigkeiten 04



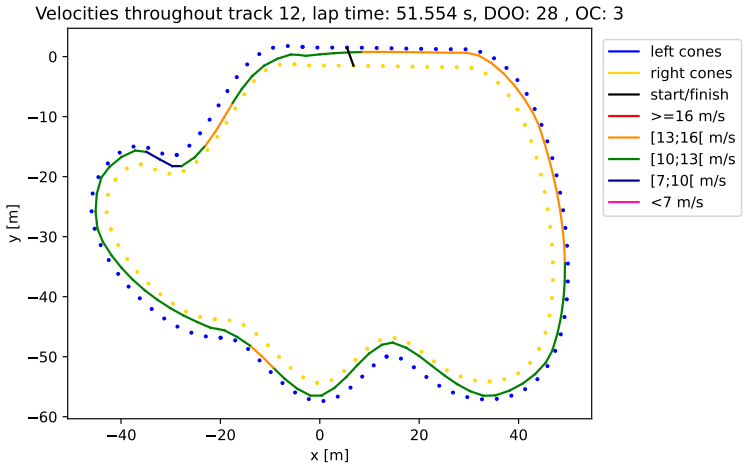
(c) Trajektorie mit Geschwindigkeiten 01

(d) Trajektorie mit Geschwindigkeiten 27

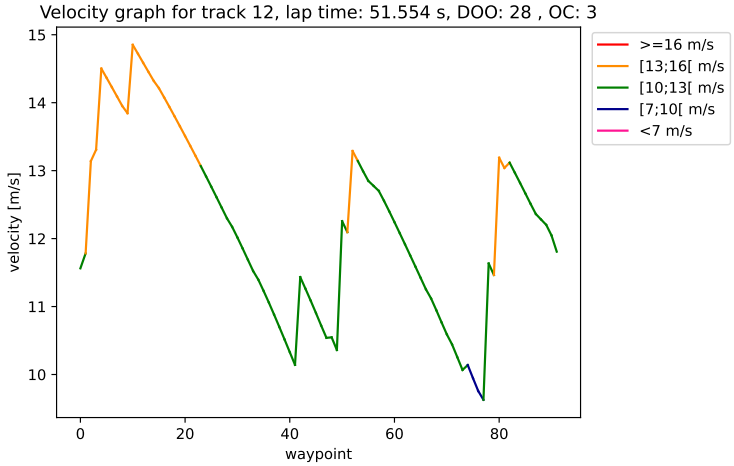


(e) Trajektorie mit Geschwindigkeiten 78

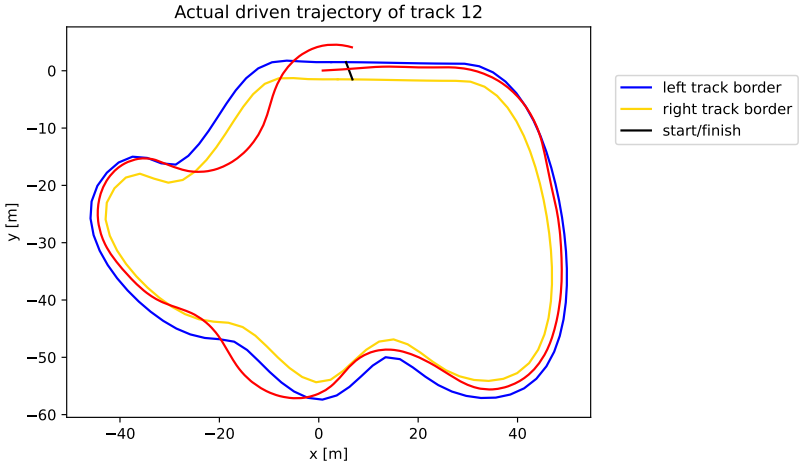
Abbildung A.7: Ergebnis der numerischen Berechnung mit FSDS-Parametern. Die Rundenzeiten sind vom Velocity Planning berechnet.



(a) Soll-Trajektorie mit Geschwindigkeiten

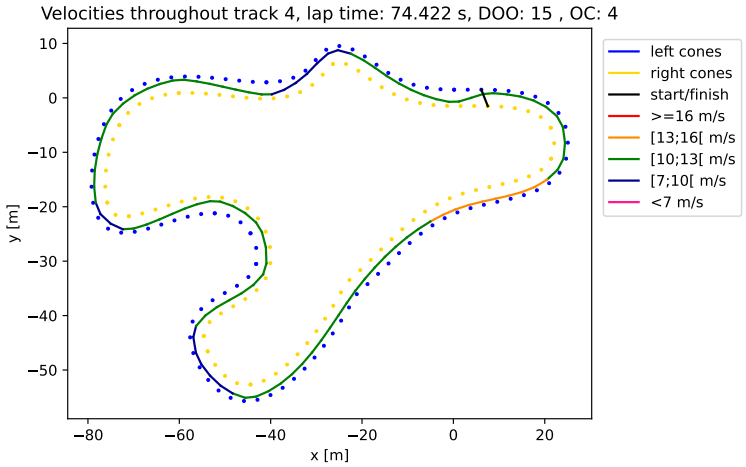


(b) Geschwindigkeitsgraph

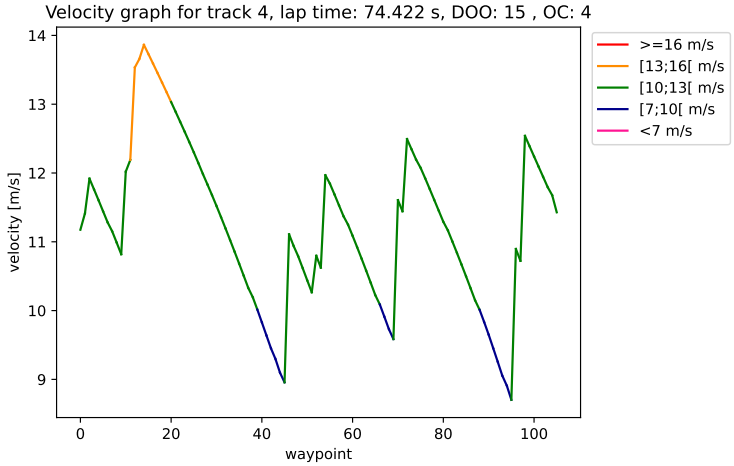


(c) Ist-Trajektorie aus der FSDS

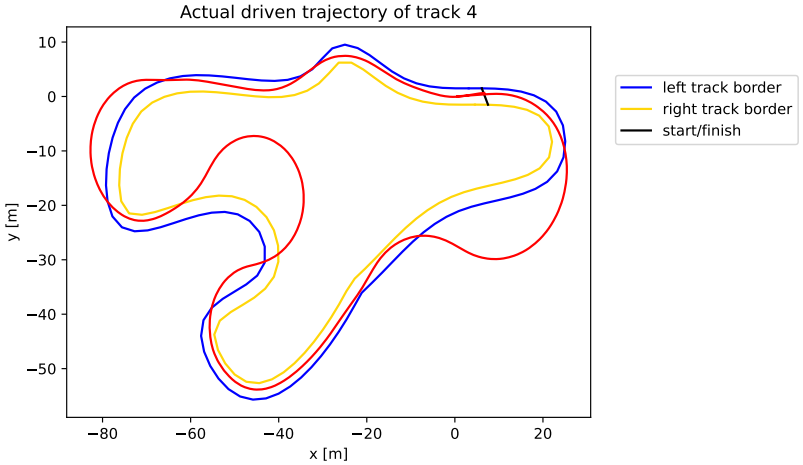
Abbildung A.8: 12-Elefant, Simulationsergebnis der Berechnung mit NOVA-Parametern



(a) Soll-Trajektorie mit Geschwindigkeiten

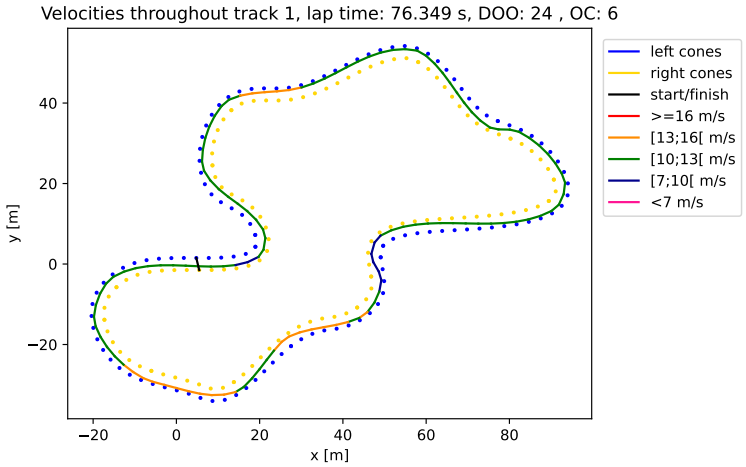


(b) Geschwindigkeitsgraph

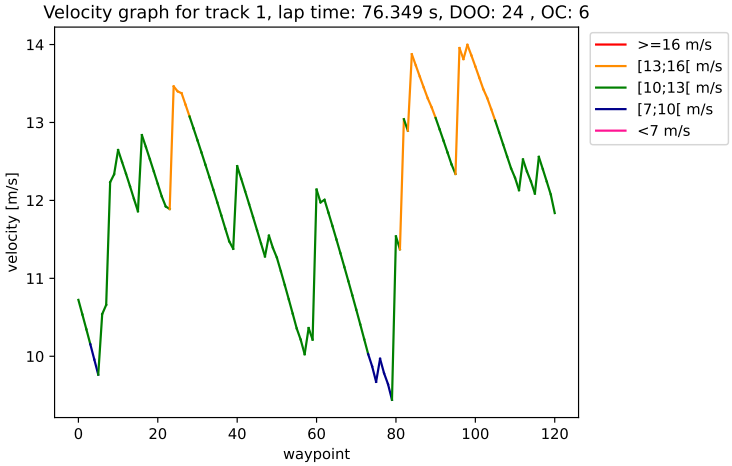


(c) Ist-Trajektorie aus der FSDS

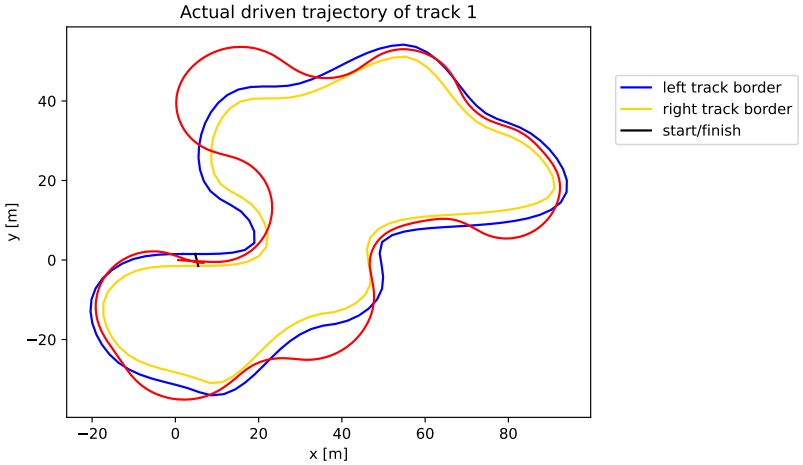
Abbildung A.9: 4-Buhu, Simulationsergebnis der Berechnung mit NOVA-Parametern



(a) Soll-Trajektorie mit Geschwindigkeiten

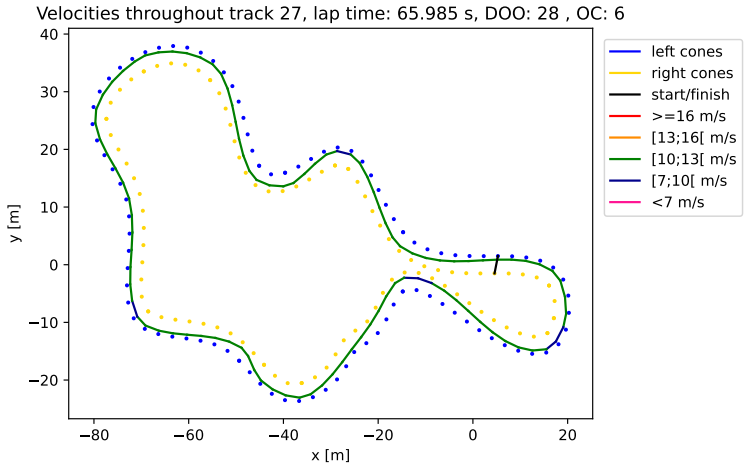


(b) Geschwindigkeitsgraph

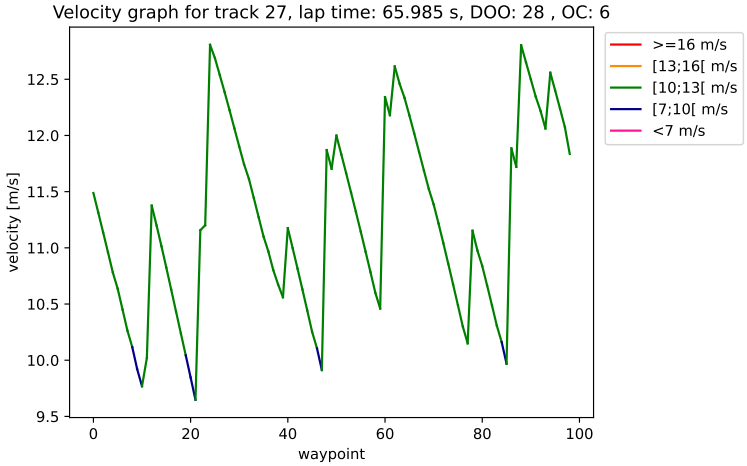


(c) Ist-Trajektorie aus der FSDS

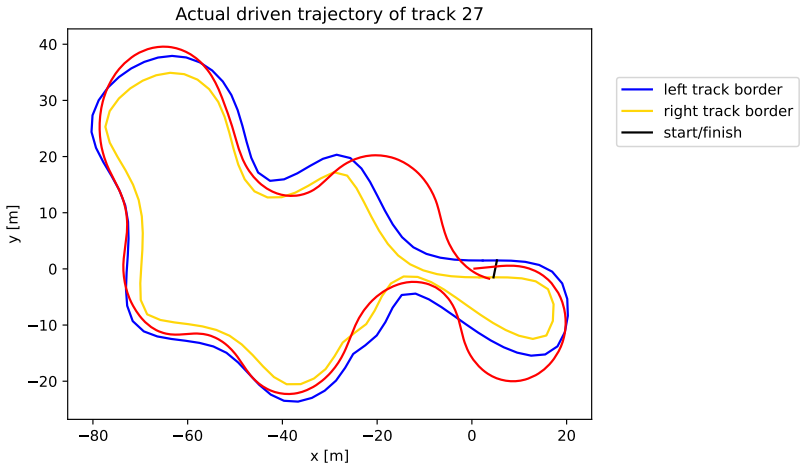
Abbildung A.10: 1-Cumulus, Simulationsergebnis der Berechnung mit NOVA-Parametern



(a) Soll-Trajektorie mit Geschwindigkeiten

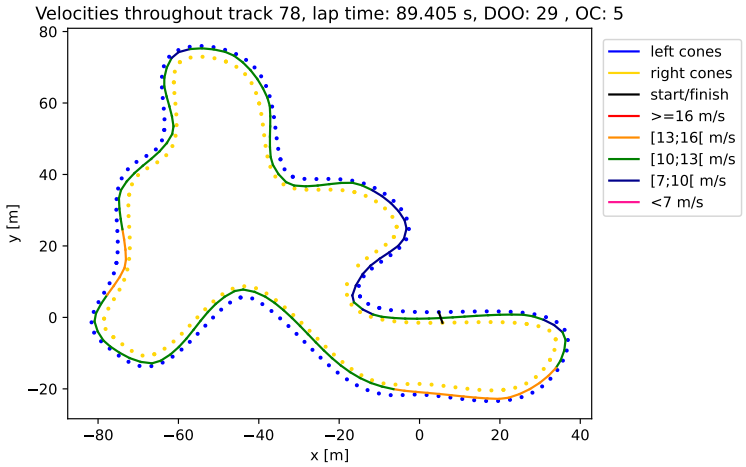


(b) Geschwindigkeitsgraph

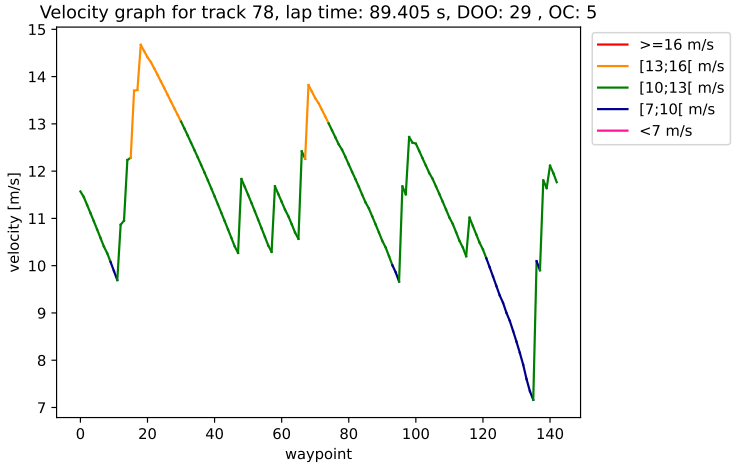


(c) Ist-Trajektorie aus der FSDS

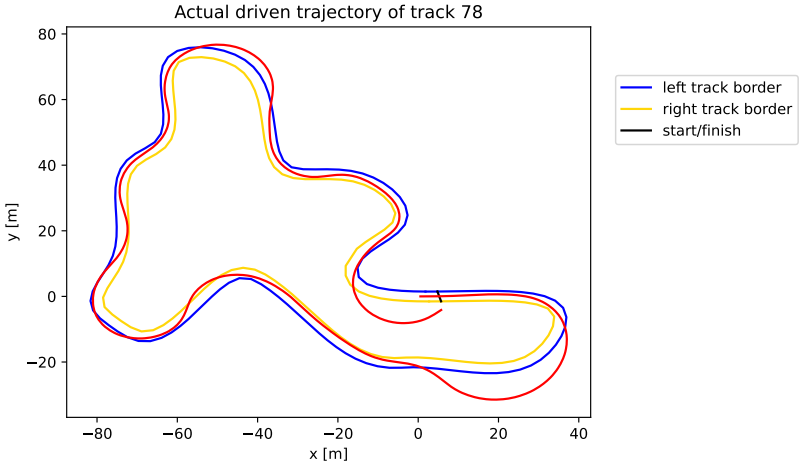
Abbildung A.11: 27-Nessie, Simulationsergebnis der Berechnung mit NOVA-Parametern



(a) Soll-Trajektorie mit Geschwindigkeiten

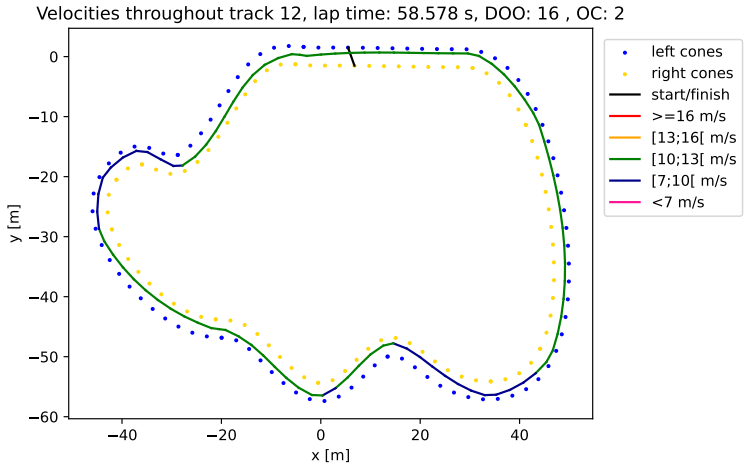


(b) Geschwindigkeitsgraph

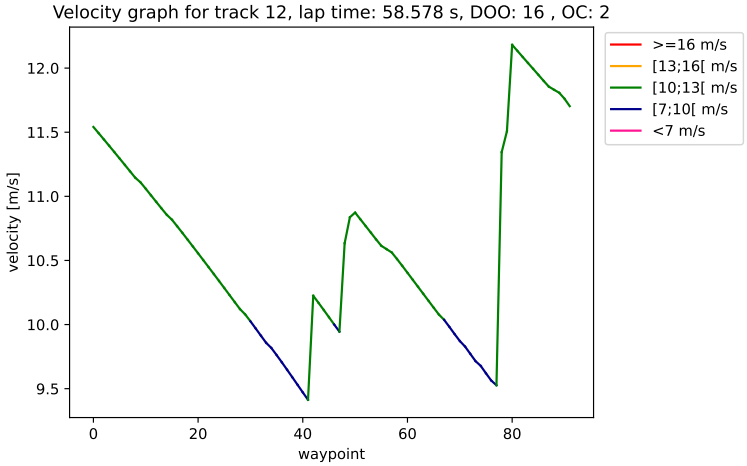


(c) Ist-Trajektorie aus der FSDS

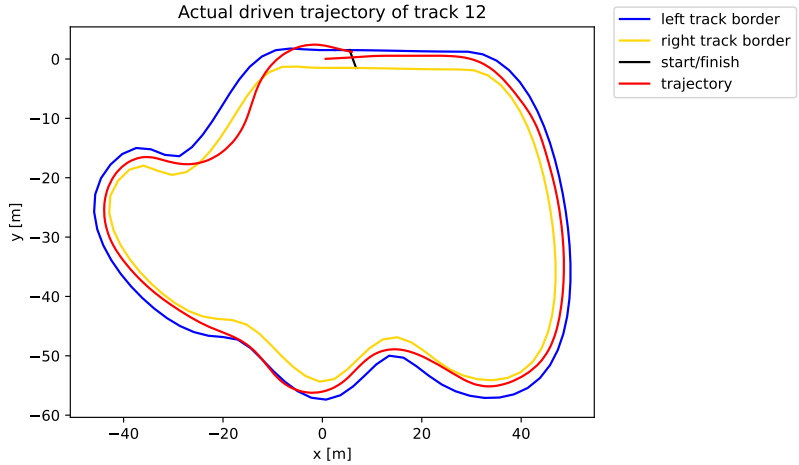
Abbildung A.12: 78-Godzilla, Simulationsergebnis der Berechnung mit NOVA-Parametern



(a) Soll-Trajektorie mit Geschwindigkeiten

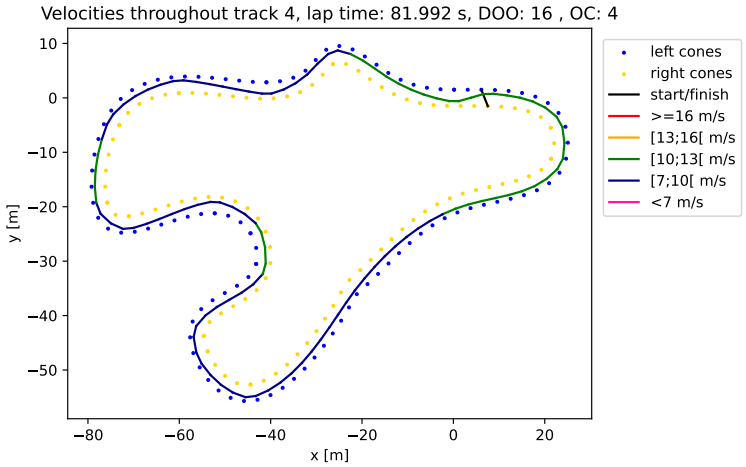


(b) Geschwindigkeitsgraph

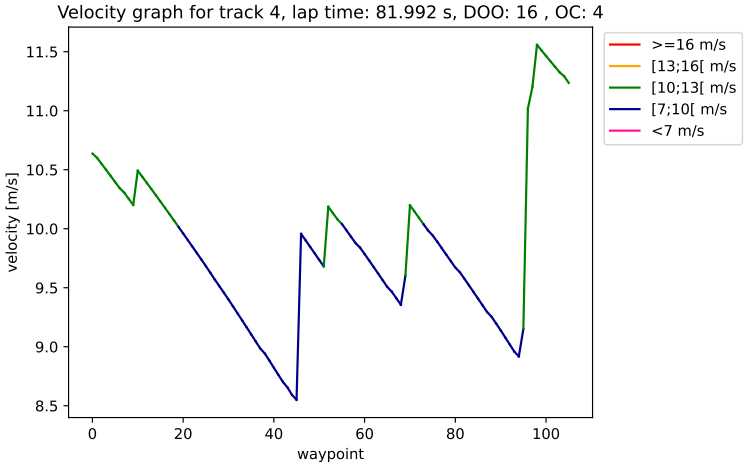


(c) Ist-Trajektorie aus der FSDS

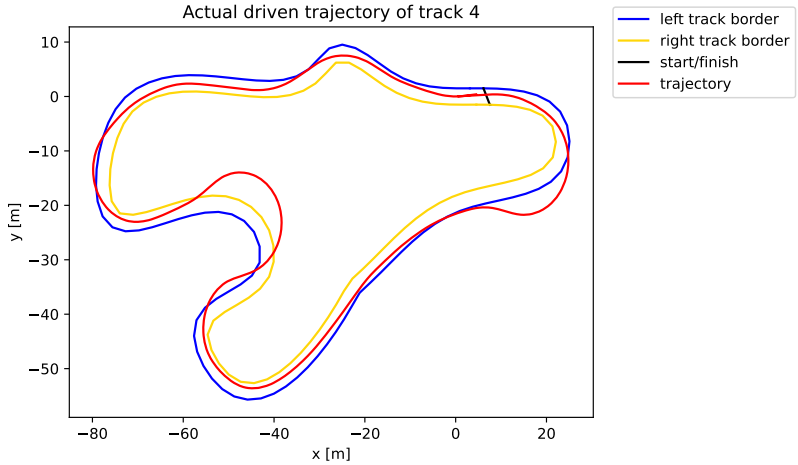
Abbildung A.13: 12-Elefant, Simulationsergebnis der Berechnung mit FSDS-Parametern



(a) Soll-Trajektorie mit Geschwindigkeiten

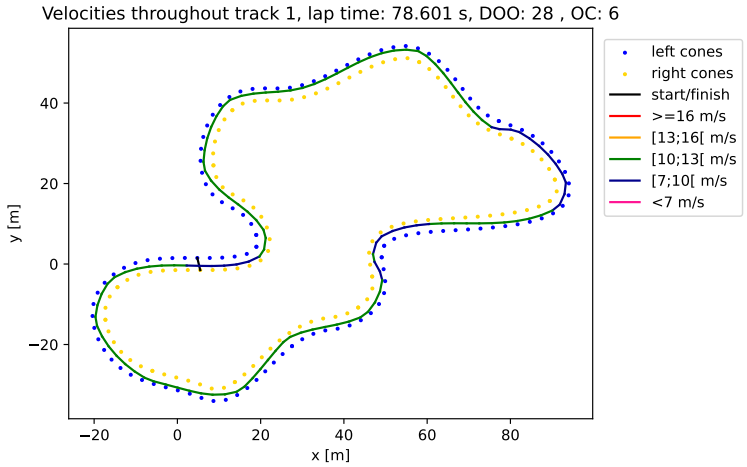


(b) Geschwindigkeitsgraph

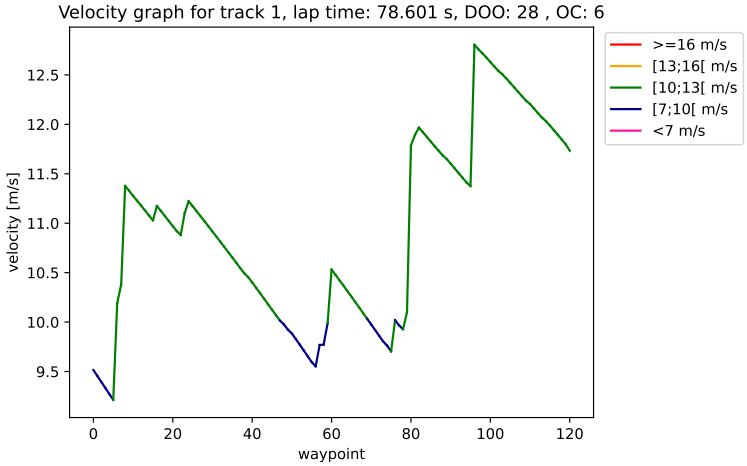


(c) Ist-Trajektorie aus der FSDS

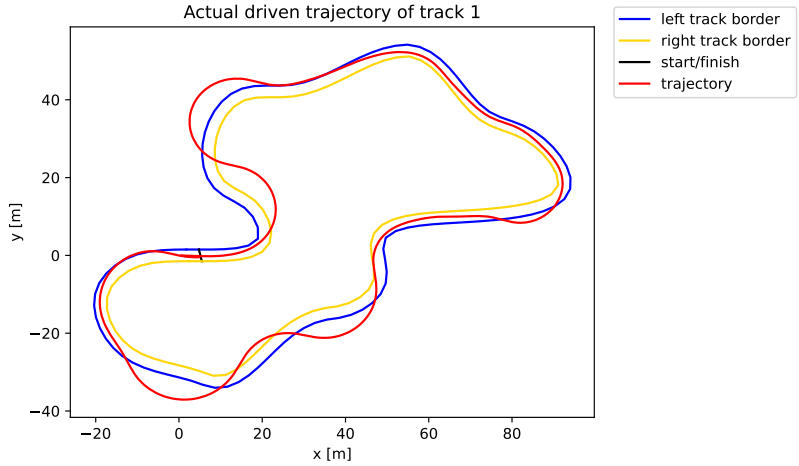
Abbildung A.14: 4-Buhu, Simulationsergebnis der Berechnung mit FSDS-Parametern



(a) Soll-Trajektorie mit Geschwindigkeiten

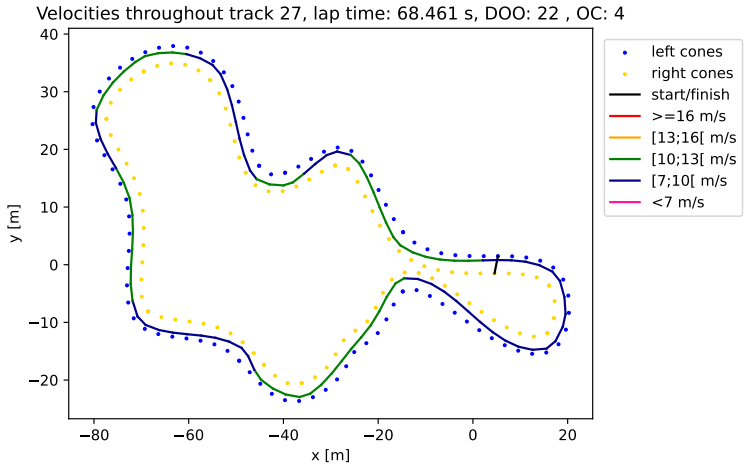


(b) Geschwindigkeitsgraph

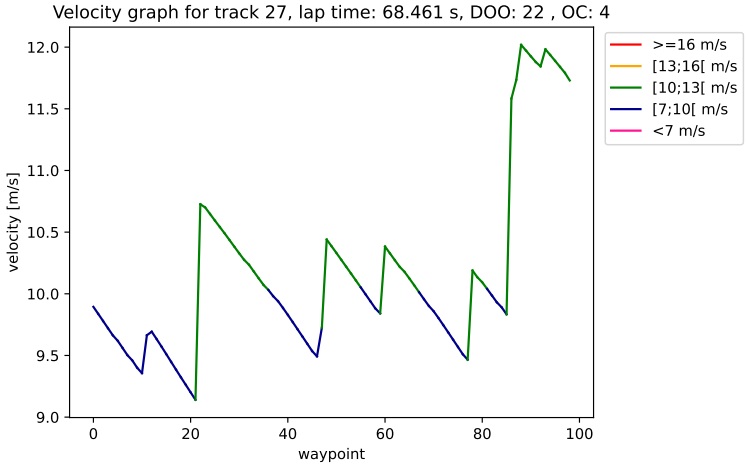


(c) Ist-Trajektorie aus der FSDS

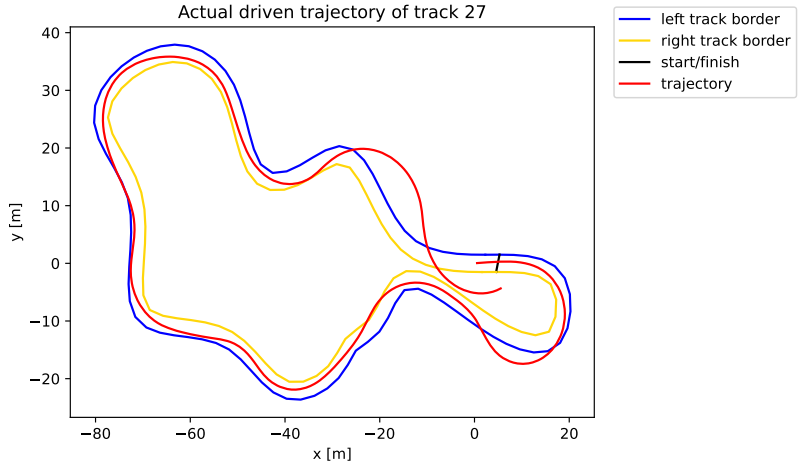
Abbildung A.15: 1-Cumulus, Simulationsergebnis der Berechnung mit FSDS-Parametern



(a) Soll-Trajektorie mit Geschwindigkeiten

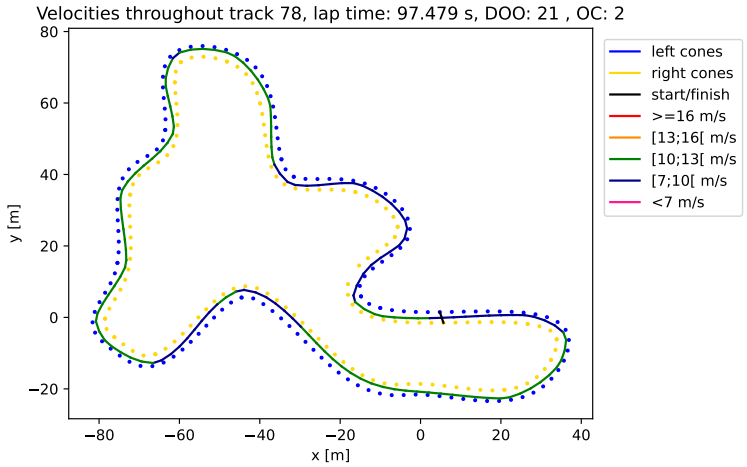


(b) Geschwindigkeitsgraph

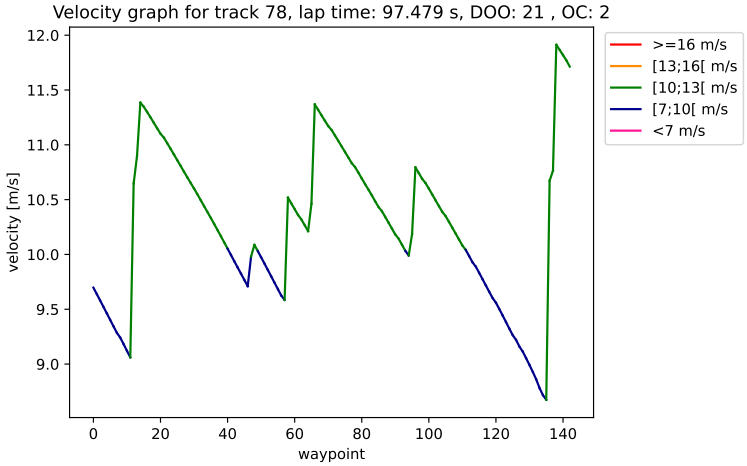


(c) Ist-Trajektorie aus der FSDS

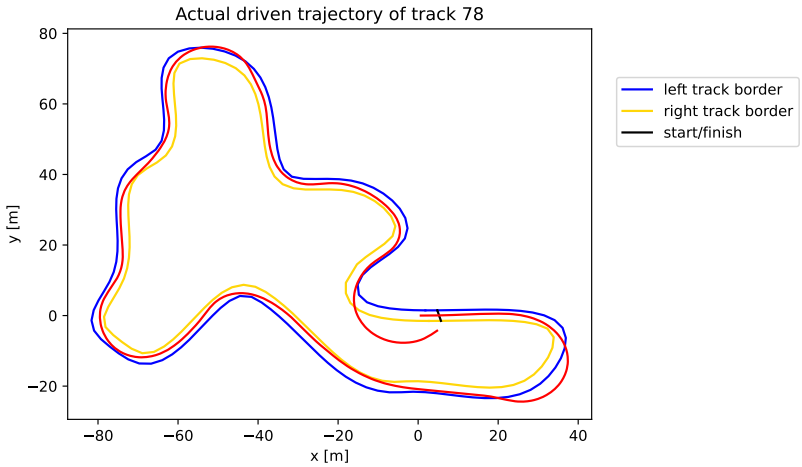
Abbildung A.16: 27-Nessie, Simulationsergebnis der Berechnung mit FSDS-Parametern



(a) Soll-Trajektorie mit Geschwindigkeiten

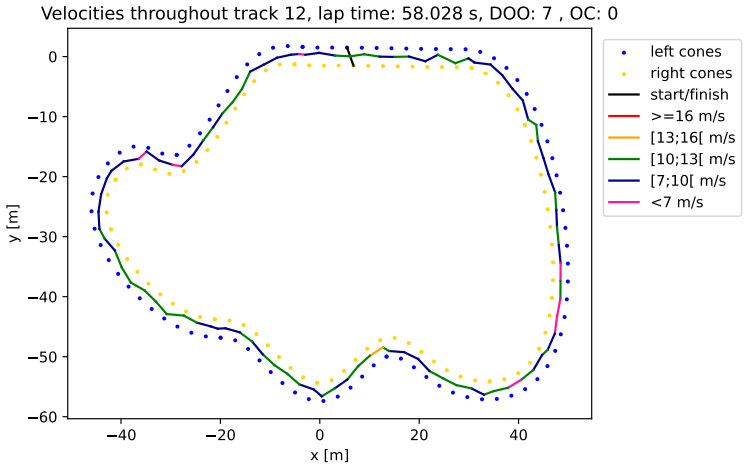


(b) Geschwindigkeitsgraph

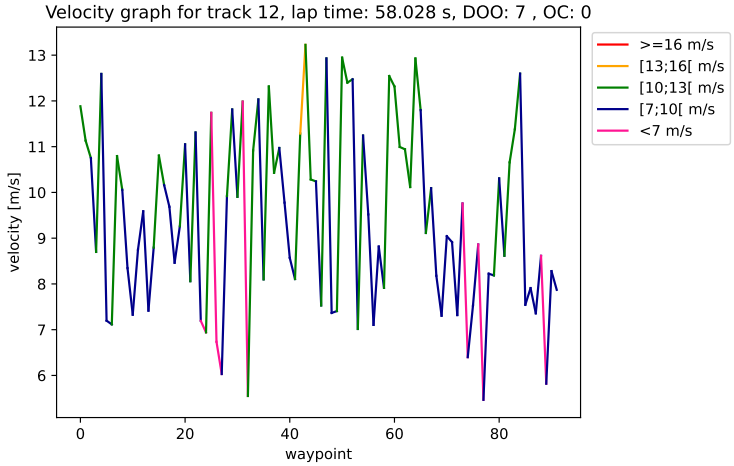


(c) Ist-Trajektorie aus der FSDS

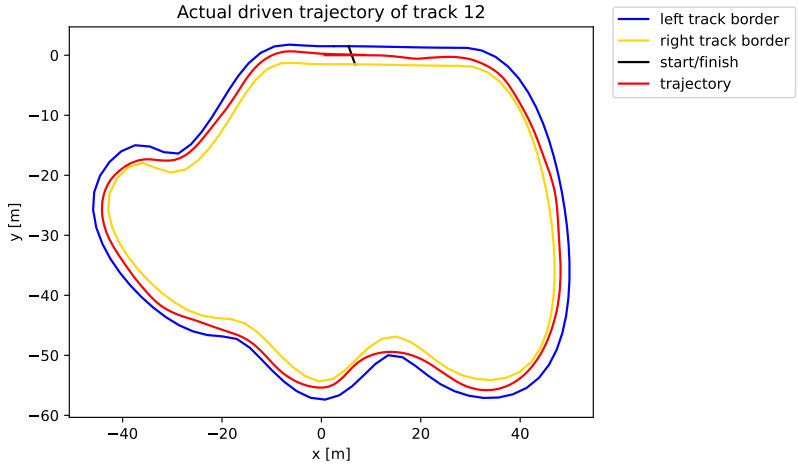
Abbildung A.17: 78-Godzilla, Simulationsergebnis der Berechnung mit FSDS-Parametern



(a) Soll-Trajektorie mit Geschwindigkeiten

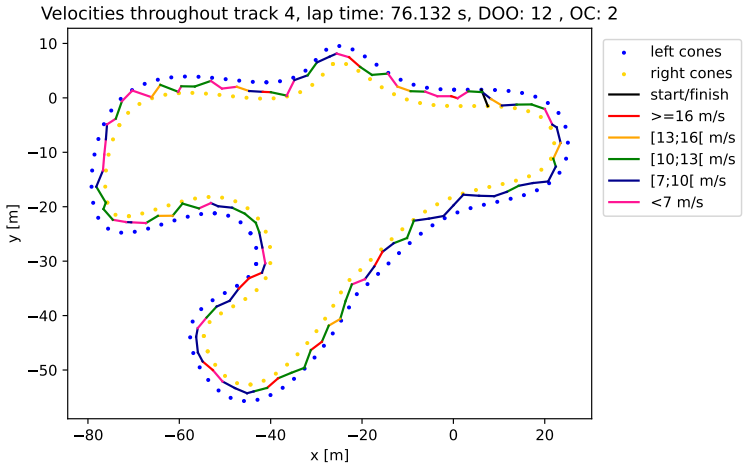


(b) Geschwindigkeitsgraph

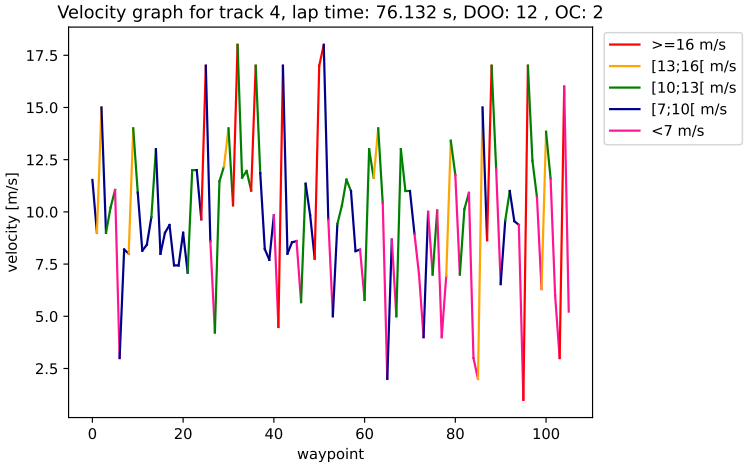


(c) Ist-Trajektorie aus der FSDS

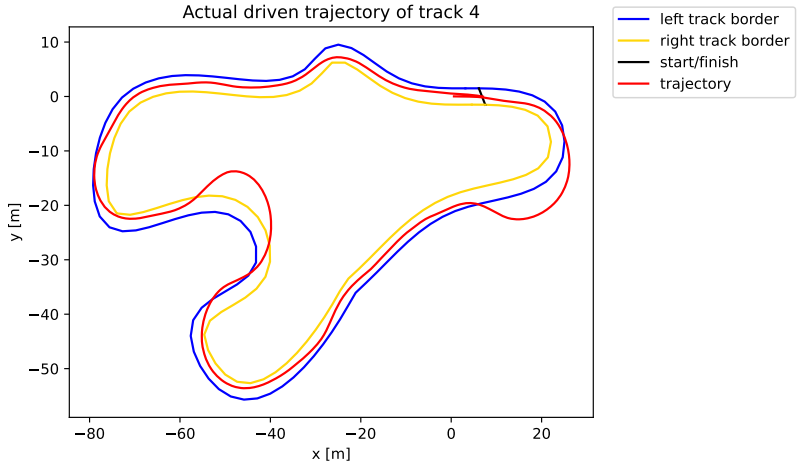
Abbildung A.18: 12-Elefant, Ergebnis des genetischen Algorithmus



(a) Soll-Trajektorie mit Geschwindigkeiten

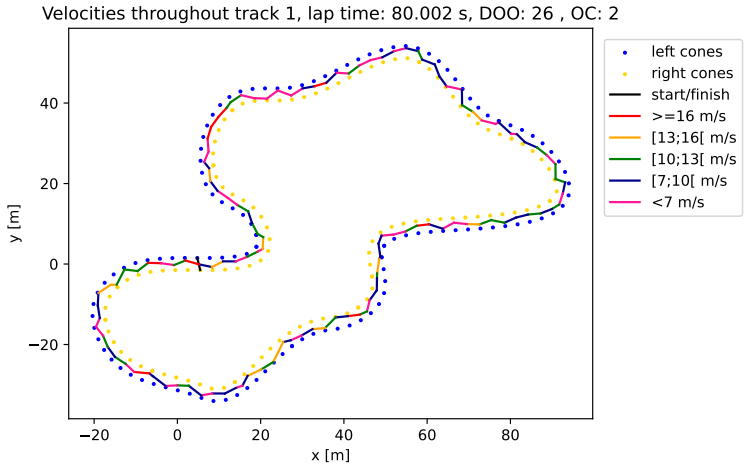


(b) Geschwindigkeitsgraph

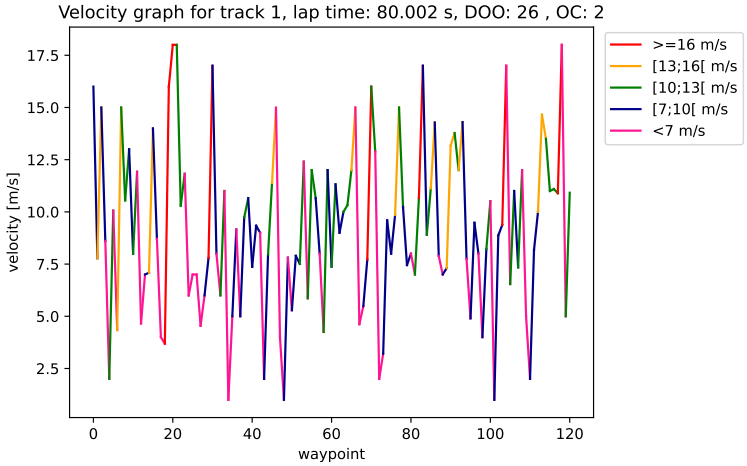


(c) Ist-Trajektorie aus der FSDS

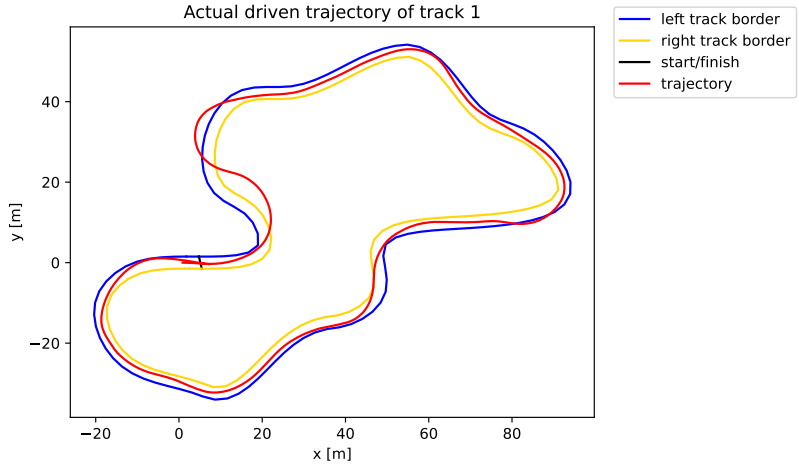
Abbildung A.19: 4-Buhu, Ergebnis des genetischen Algorithmus



(a) Soll-Trajektorie mit Geschwindigkeiten

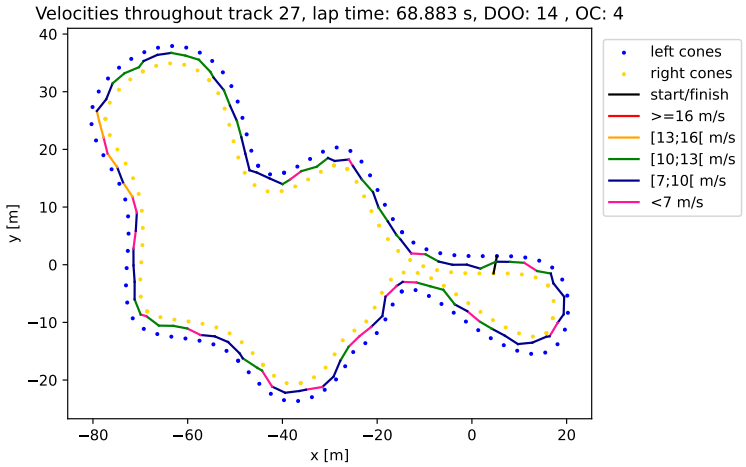


(b) Geschwindigkeitsgraph

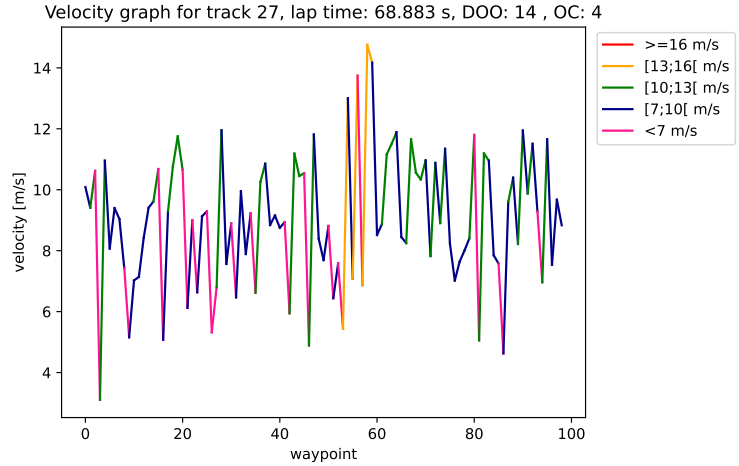


(c) Ist-Trajektorie aus der FSDS

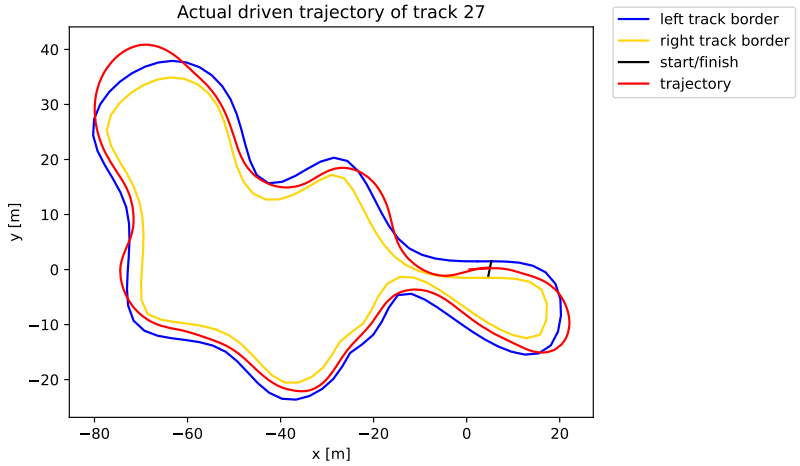
Abbildung A.20: 1-Cumulus, Ergebnis des genetischen Algorithmus



(a) Soll-Trajektorie mit Geschwindigkeiten

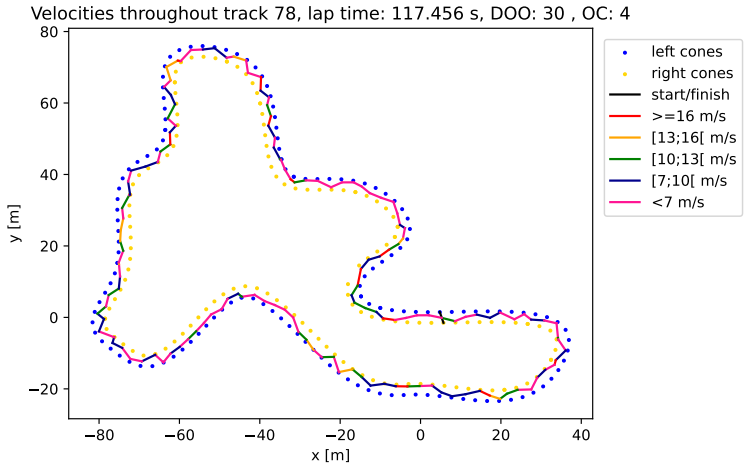


(b) Geschwindigkeitsgraph

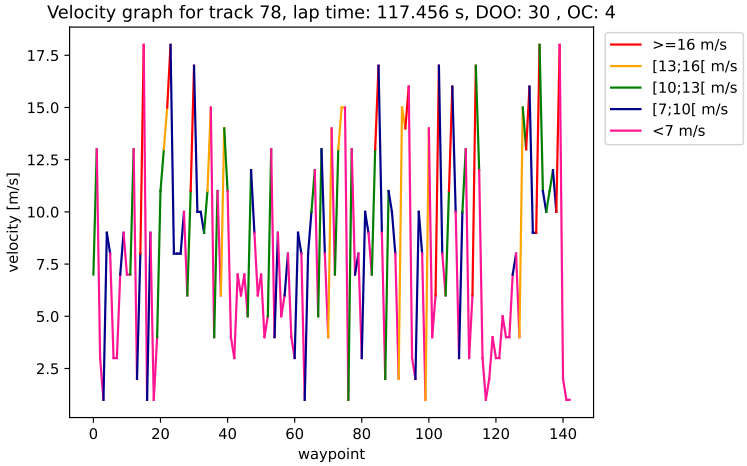


(c) Ist-Trajektorie aus der FSDS

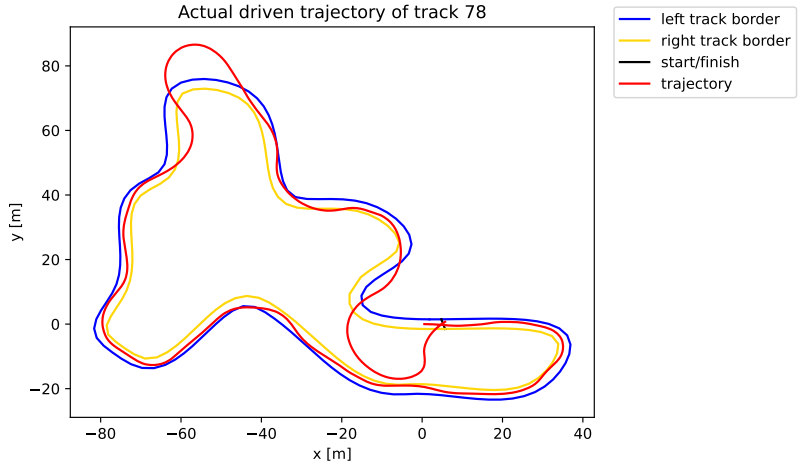
Abbildung A.21: 27-Nessie, Ergebnis des genetischen Algorithmus



(a) Soll-Trajektorie mit Geschwindigkeiten

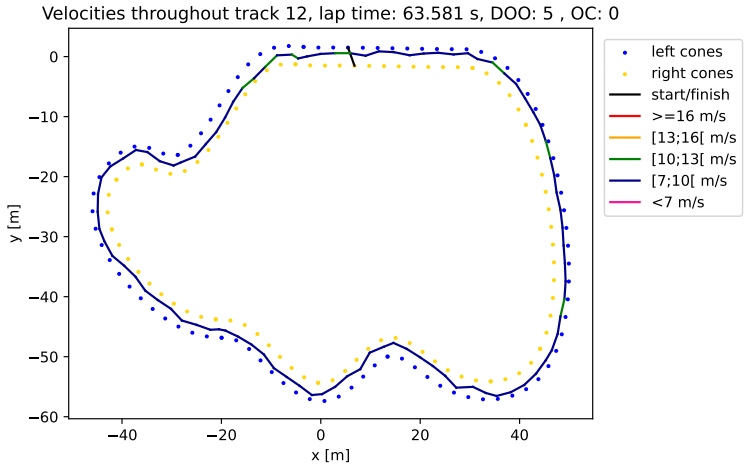


(b) Geschwindigkeitsgraph

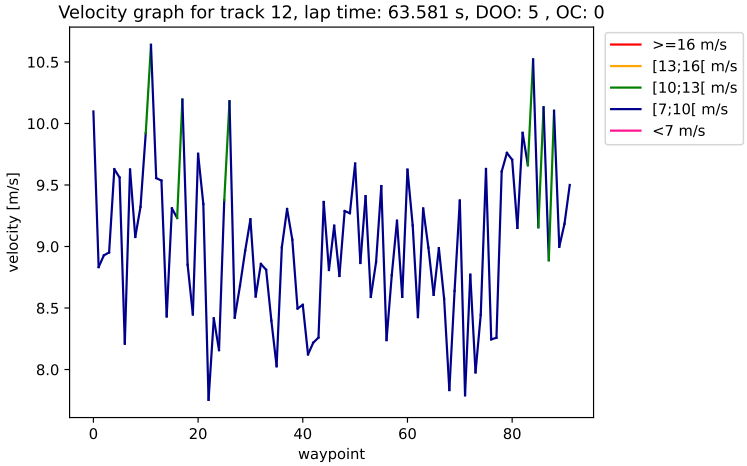


(c) Ist-Trajektorie aus der FSDS

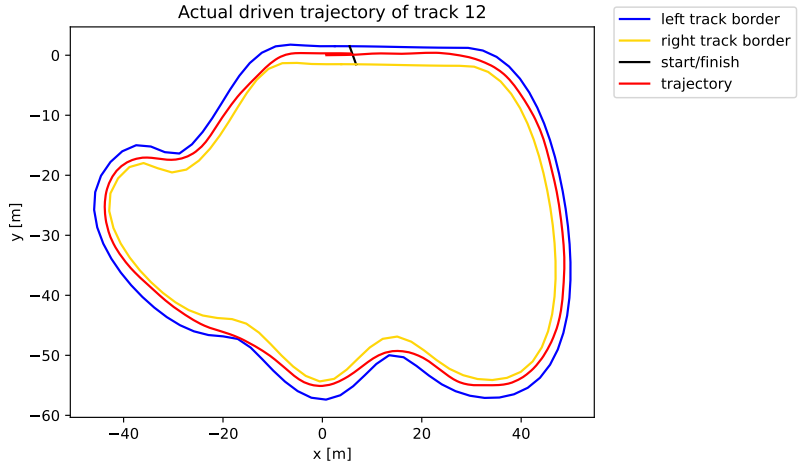
Abbildung A.22: 78-Godzilla, Ergebnis des genetischen Algorithmus



(a) Soll-Trajektorie mit Geschwindigkeiten

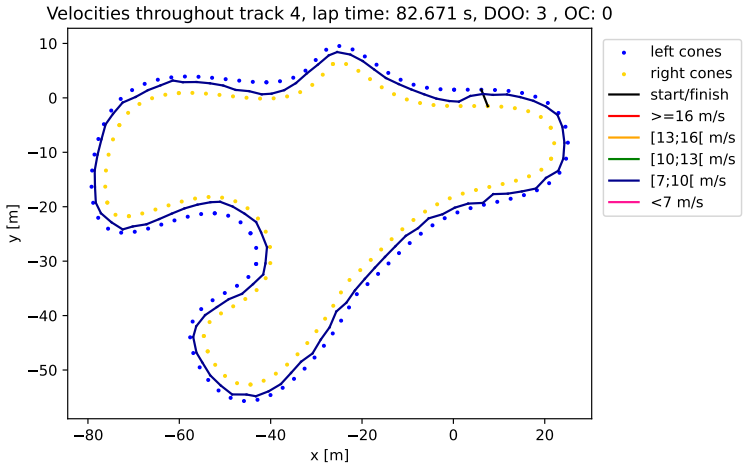


(b) Geschwindigkeitsgraph

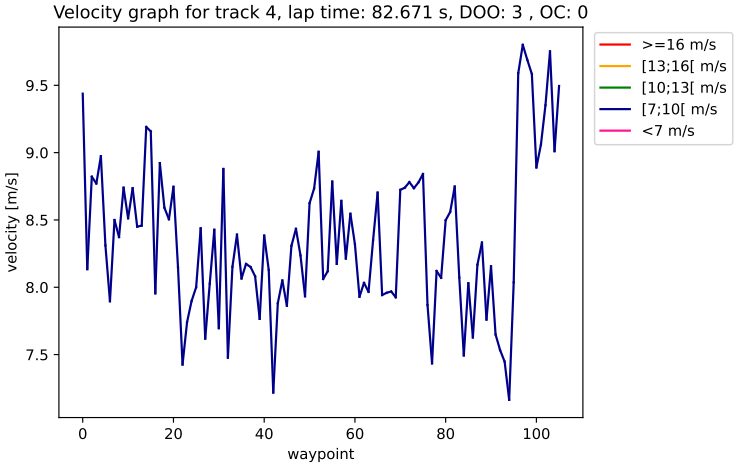


(c) Ist-Trajektorie aus der FSDS

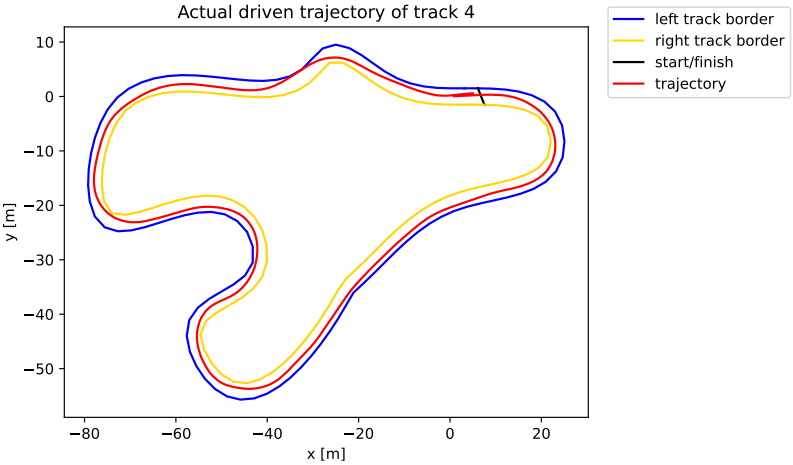
Abbildung A.23: 12-Elefant, Ergebnis des hybriden Ansatzes



(a) Soll-Trajektorie mit Geschwindigkeiten

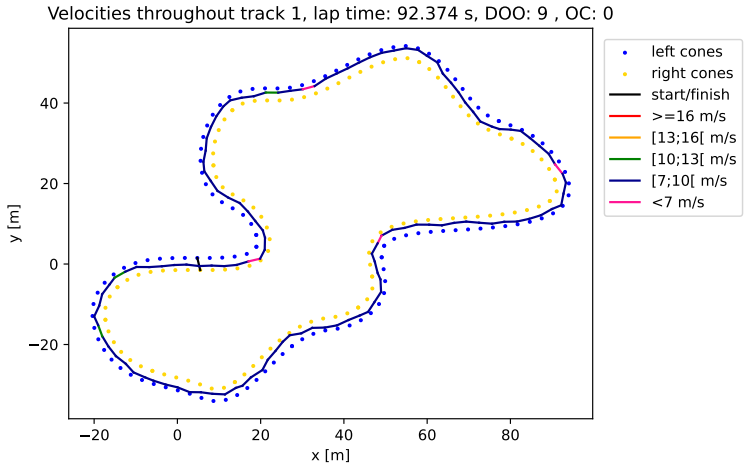


(b) Geschwindigkeitsgraph

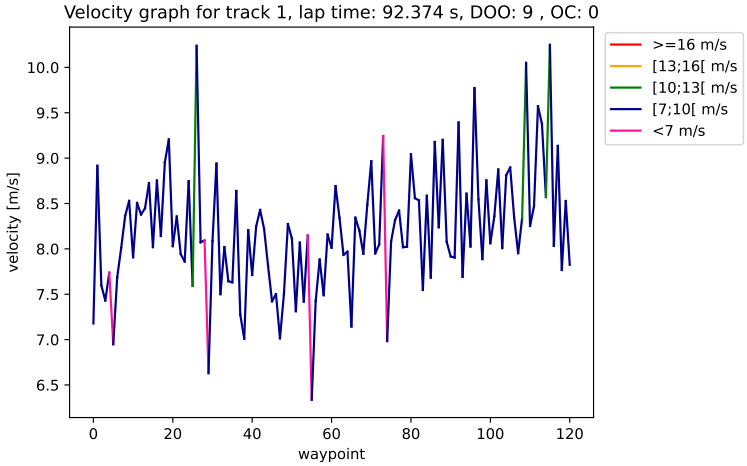


(c) Ist-Trajektorie aus der FSDS

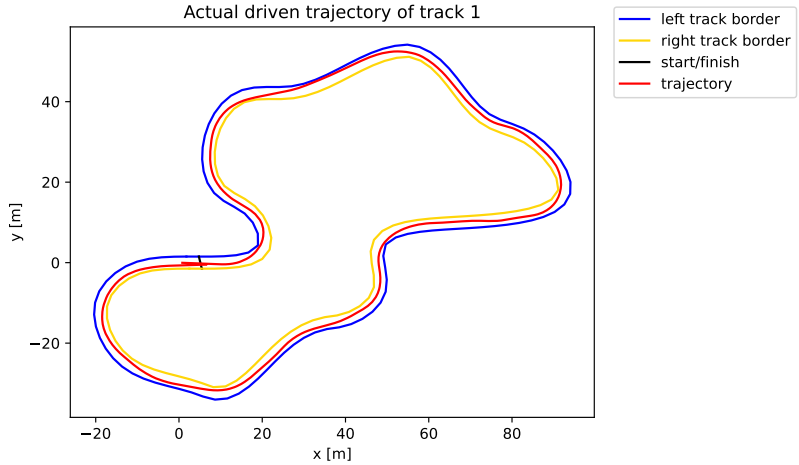
Abbildung A.24: 4-Buhu, Ergebnis des hybriden Ansatzes



(a) Soll-Trajektorie mit Geschwindigkeiten

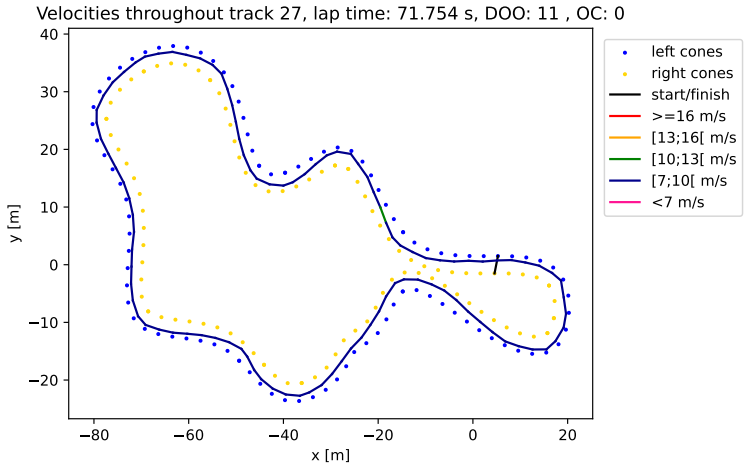


(b) Geschwindigkeitsgraph

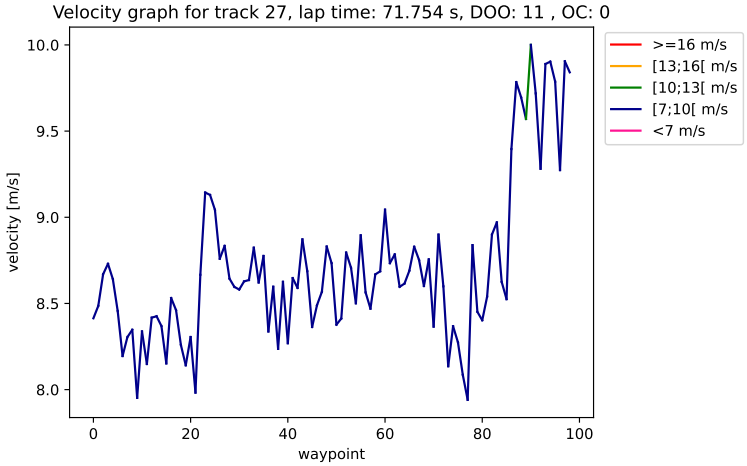


(c) Ist-Trajektorie aus der FSDS

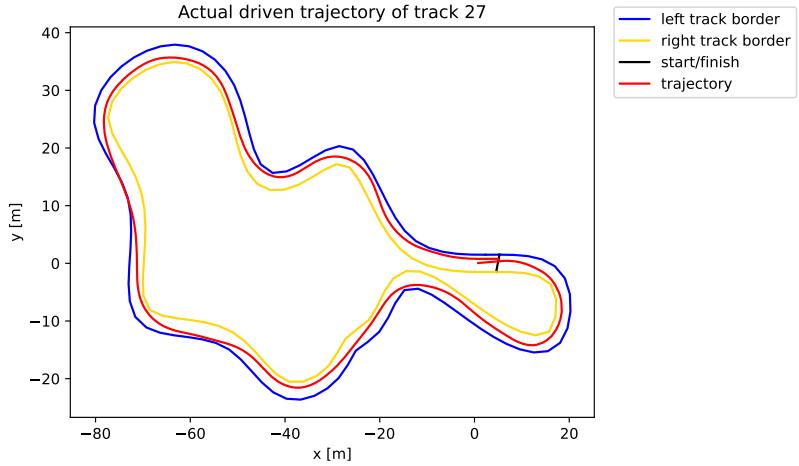
Abbildung A.25: 1-Cumulus, Ergebnis des hybriden Ansatzes



(a) Soll-Trajektorie mit Geschwindigkeiten

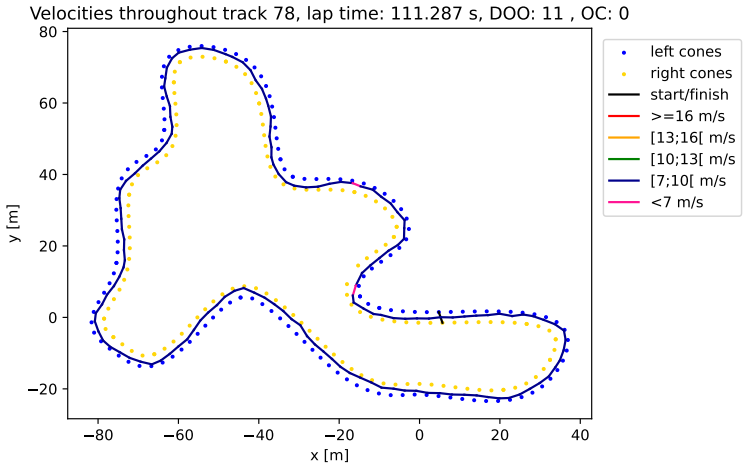


(b) Geschwindigkeitsgraph

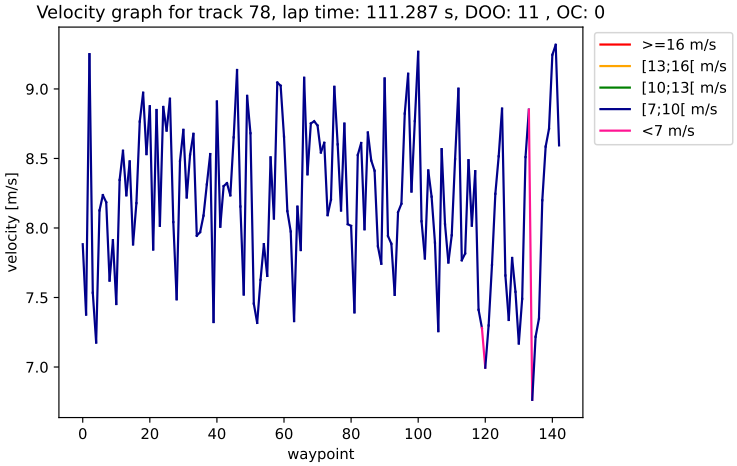


(c) Ist-Trajektorie aus der FSDS

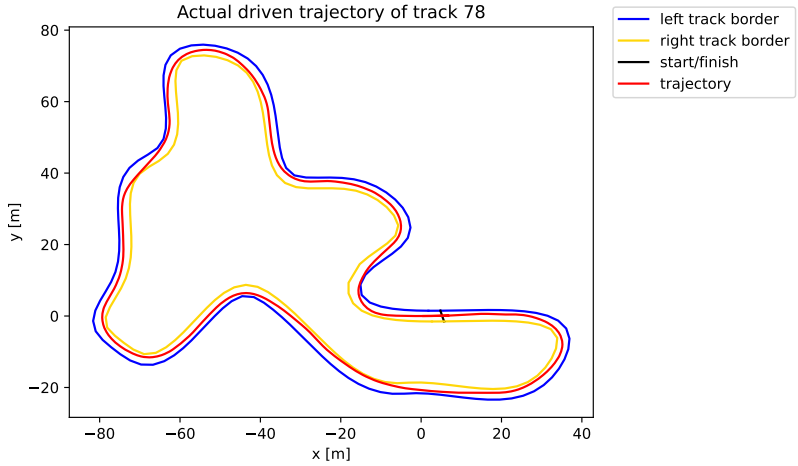
Abbildung A.26: 27-Nessie, Ergebnis des hybriden Ansatzes



(a) Soll-Trajektorie mit Geschwindigkeiten

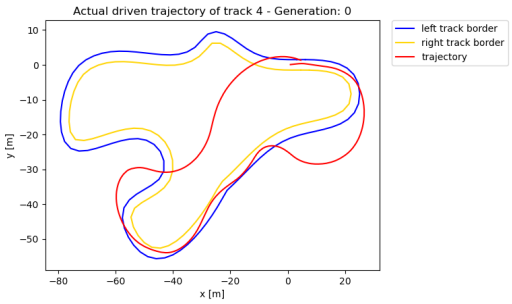


(b) Geschwindigkeitsgraph

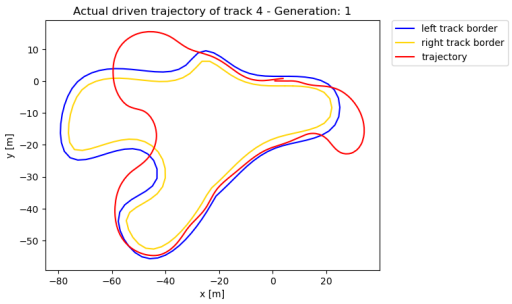


(c) Ist-Trajektorie aus der FSDS

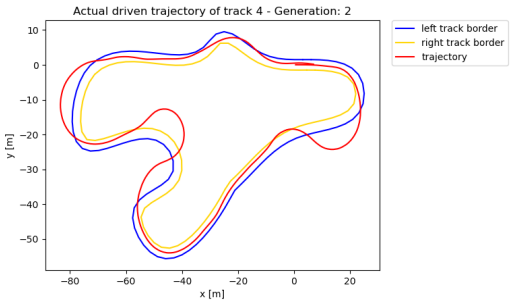
Abbildung A.27: 78-Godzilla, Ergebnis des hybriden Ansatzes



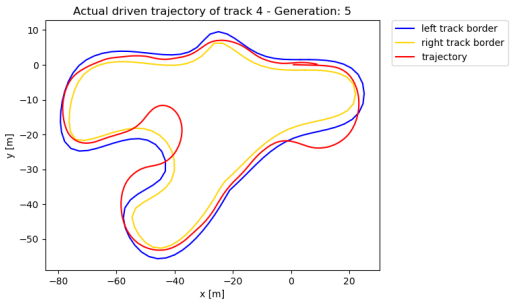
(a) Erste ganze Runde



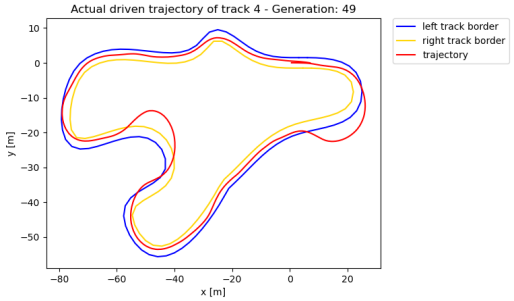
(b) Annäherung an die Strecke



(c) Streckengeometrie erkennbar



(d) Verbesserung von 4 auf 3 OC



(e) Aus 3 werden 2 OC

Abbildung A.28: Meilensteine im Evolutionsprozess von Strecke 4

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original