



TECHNISCHE UNIVERSITÄT BERLIN

FACHGEBIET BAHNBETRIEB UND INFRASTRUKTUR

BACHELORARBEIT

**Realitätsnahe Fahrzeugsteuerung  
für die Eisenbahnbetriebssimulation  
im Eisenbahn-Betriebs- und  
Experimentierfeld**

*Friedrich Kasper Völkers*

391529

betreut von

Dr.-Ing. Christian BLOME

Berlin, 22. September 2021



## Aufgabenstellung

Im Eisenbahn-Betriebs- und Experimentierfeld (EBuEf) des Fachgebietes Bahnbetrieb und Infrastruktur der Technischen Universität Berlin können Prozesse des Bahnbetriebs unter realitätsnahen Bedingungen simuliert werden. Den Mittelpunkt der Anlagen bilden originale Stellwerke unterschiedlicher Entwicklungsstufen der Eisenbahnsicherungstechnik vom mechanischen Stellwerk bis zu aus einer Betriebszentrale gesteuerten Elektronischen Stellwerken.

Das „Ausgabemedium“ ist eine Modellbahnanlage, die in verkleinertem Maßstab die Abläufe darstellt. Das Betriebsfeld wird in der Lehre im Rahmen der Bachelor- und Masterstudiengänge am Fachgebiet sowie darüber hinaus zur Ausbildung von Fahrdienstleitern, für Schulungen und Weiterbildungen Externer sowie bei öffentlichen Veranstaltungen wie beispielsweise der Langen Nacht der Wissenschaften eingesetzt.

Neben den Stellwerken ist auch bei den Fahrzeugen ein möglichst realitätsnaher Betrieb Teil der umfassenden Eisenbahnbetriebssimulation.

Ziel dieser Arbeit ist die Entwicklung einer Steuerungssoftware, die auf dem (modellseitig nur) punktförmig überwachten Netz die Fahrzeuge kontinuierlich überwacht, um die Fahrzeuge realitätsnäher zu steuern (beispielsweise durch maßstäbliche Beschleunigung oder punktgenaues Anhalten an Bahnsteigen gemäß der aktuellen Zuglängen) und zukünftig auch andere und neue Betriebsverfahren wie Moving Block im EBuEf simulieren zu können.

Teil der kontinuierlichen Überwachung ist die exakte Positionsbestimmung der Fahrzeuge im Netz sowie die Übermittlung der aktuellen Geschwindigkeit.

Beschleunigungs- und Bremsvorgänge sowie Ausrollphasen für optional energieoptimales Fahren sind ebenso zu berücksichtigen. Zur Kalibrierung sind die schon vorhandenen Ortungsmöglichkeiten (Belegung von Gleisabschnitten) zu verwenden.

Weitere zu berücksichtigende Eingangsgrößen aus der vorhandenen Softwarelandschaft im EBuEf sind die Netztopologie (z.B. Streckenlängen, Signalstandorte), die Fahrzeugdaten, die aktuelle Zugbildung sowie die Prüfung (vorhandene API), ob ein Zug an einer Station anhalten muss und ob er abfahren darf. Damit sind in der Simulation Fahrplanteue, Verspätungen sowie Personalausfälle darstellbar.

Die Erkenntnisse sind in einem umfassenden Bericht und einer zusammenfassenden Textdatei darzustellen. Darüber hinaus sind die Ergebnisse der Arbeit ggf. im Rahmen einer Vortragsveranstaltung des Fachgebiets zu präsentieren.

Der Bericht soll in gedruckter Form als gebundenes Dokument sowie in elektronischer Form als ungeschütztes PDF-Dokument eingereicht werden. Methodik und Vorge-

hen bei der Arbeit sind explizit zu beschreiben und auf eine entsprechende Zitierweise ist zu achten. Alle genutzten bzw. verarbeiteten zugrundeliegenden Rohdaten sowie nicht-veröffentlichte Quellen müssen der Arbeit (ggf. in elektronischer Form) beiliegen.

In dem Bericht ist hinter dem Deckblatt der originale Wortlaut der Aufgabenstellung der Arbeit einzuordnen. Weiterhin muss der Bericht eine einseitige Zusammenfassung der Arbeit enthalten. Diese Zusammenfassung der Arbeit ist zusätzlich noch einmal als eigene, unformatierte Textdatei einzureichen.

Für die Bearbeitung der Aufgabenstellung sind die Hinweise zu beachten, die auf der Webseite mit der Adresse [www.railways.tu-berlin.de/?id=66923](http://www.railways.tu-berlin.de/?id=66923) gegeben werden.

Der Fortgang der Abarbeitung ist in engem Kontakt mit dem Betreuer regelmäßig abzustimmen. Hierzu zählen insbesondere mindestens alle vier Wochen kurze Statusberichte in mündlicher oder schriftlicher Form.

## Zusammenfassung

Im Rahmen dieser Arbeit wurde eine Fahrzeugsteuerung entwickelt, welche die Fahrzeuge im eingleisigen Netz des Eisenbahn-Betriebs- und Experimentierfelds (EBuEfs) ansteuert. Dazu wurde ein allgemeingültiger Algorithmus entwickelt, der einen möglichst optimalen Fahrtverlauf ermittelt. Die Berechnung des Fahrtverlaufs basiert auf den gegebenen Infrastrukturabschnitten inklusive deren Länge und zulässiger Höchstgeschwindigkeit, der aktuellen Position und Geschwindigkeit, der Zielposition und der Ankunftszeit. Durch den ermittelten Fahrtverlauf ist eine kontinuierliche Fahrzeugüberwachung möglich und die aktuelle Position der Fahrzeuge ist zu jedem Zeitpunkt bekannt.

Todo-Liste (Beim Korrekturlesen bitte nicht beachten...)

- Was funktioniert nicht
- Formeln?!
- linebreak
- glossaries
- acronyms
- leerzeichen zwischen zahl und einheit
- doppelte leerzeichen
- *\$allTrains* beschreiben

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>2</b>	<b>Grundlagen</b>	<b>2</b>
2.1	Aufbau des Eisenbahn-Betriebs- und Experimentierfelds . . . . .	2
2.2	Aufbau der <i>MySQL</i> -Datenbank . . . . .	3
2.3	Ziele und Prioritätssetzung der Fahrzeugsteuerung . . . . .	3
2.4	Fahrdynamik . . . . .	4
2.5	Aufbau des Projekts . . . . .	4
<b>3</b>	<b>Ablauf der Fahrzeugsteuerung</b>	<b>6</b>
3.1	Einlesen von statischen und mehrfach verwendeten Daten aus der <i>MySQL</i> -Datenbank in den Cache . . . . .	6
3.2	Ermittlung der Session-Daten . . . . .	7
3.3	Ermittlung aller Fahrzeuge im eingleisigen Netz und den zugehörigen Daten . . . . .	8
3.4	Berechnung der Fahrtverläufe aller Fahrzeuge . . . . .	10
3.5	Übermittlung der Echtzeitdaten an die Fahrzeuge . . . . .	14
3.6	Überprüfung nach einer Änderung der Fahrstraße . . . . .	18
3.7	Neukalibrierung der Fahrzeugposition . . . . .	18
3.8	Ermittlung von neuen Fahrzeugen im eingleisigen Netz . . . . .	20
3.9	Fehlerbehebung von Fahrzeugen . . . . .	20
<b>4</b>	<b>Berechnung des Fahrtverlaufs</b>	<b>22</b>
4.1	Ermittlung der Start- und Endposition der einzelnen Infrastrukturab- schnitt (Infra-Abschnitt)e unter Berücksichtigung der Zuglänge . . . . .	24
4.2	Berechnung bei einer Beschleunigung auf die maximal mögliche Ge- schwindigkeit . . . . .	27
4.3	Konvertierung der <i>\$keyPoints</i> in Echtzeitdaten . . . . .	29
4.4	Überprüfung des Fahrtverlaufs nach Geschwindigkeitsüberschreitungen	35
4.5	Neuberechnung unter Berücksichtigung der Geschwindigkeitsüber- schreitung . . . . .	35

4.6	Einhaltung der Mindestzeit auf einer Beharrungsfahrt . . . . .	37
4.7	Berücksichtigung der Ankunftszeit bei der Berechnung des Fahrtverlaufs	41
4.8	Berücksichtigung der exakten Ankunftszeit bei der Berechnung des Fahrtverlaufs . . . . .	43
4.9	Einleitung einer Gefahrenbremsung . . . . .	46
<b>5</b>	<b>Beispielrechnung eines Fahrtverlaufs im EBUf</b>	<b>48</b>
<b>6</b>	<b>Visualisierung der Fahrtverläufe</b>	<b>52</b>
<b>7</b>	<b>Formeln</b>	<b>54</b>
7.1	Formeln für gleichmäßig beschleunigte Bewegungen . . . . .	54
7.2	Formeln für gleichförmige Bewegungen . . . . .	56
<b>8</b>	<b>Fazit</b>	<b>58</b>
8.1	Zusammenfassung der Ergebnisse . . . . .	58
8.2	Komplikationen bei dem Betrieb der Fahrzeugsteuerung im EBUf . . .	58
8.2.1	Einhaltung der Zielposition . . . . .	58
8.2.2	Ermittlung der Fahrstraßen . . . . .	59
8.2.3	Kalibrierung der Position . . . . .	59
8.3	Möglichkeiten für eine Weiterentwicklung der Fahrzeugsteuerung . . . .	59
<b>A</b>	<b>Anhang</b>	<b>60</b>
A.1	fahrzeugsteuerung.php . . . . .	60
A.2	functions.php . . . . .	69
A.3	functions_fahrtverlauf.php . . . . .	97
A.4	functions_math.php . . . . .	147
A.5	functions_cache.php . . . . .	148
A.6	functions_db.php . . . . .	154
A.7	global_variables.php . . . . .	160
A.8	speed_over_position.m . . . . .	161

## Abbildungsverzeichnis

1	Schienennetz des EBUfs . . . . .	2
2	Aufbau der Dateistrukturen . . . . .	5
3	Ablauf der Fahrzeugsteuerung . . . . .	6
4	Eigene Darstellung der Positionsbestimmung bei einem Richtungswechsel	16
5	Ablaufplan der Fahrtverlaufsrechnung . . . . .	23
6	Infra-Abschnitte und die zugehörige Höchstgeschwindigkeit . . . . .	26
7	Infra-Abschnitte und die zugehörige Höchstgeschwindigkeit unter Berücksichtigung der Fahrzeuglänge . . . . .	27
8	Fahrtverlaufsrechnung (1. Iterationsschritt) . . . . .	29
9	Fahrtverlaufsrechnung (2. Iterationsschritt) . . . . .	37
10	Fahrtverlaufsrechnung (3. Iterationsschritt) . . . . .	38
11	Fahrtverlaufsrechnung (4. Iterationsschritt) . . . . .	38
12	Einteilung des Fahrtverlaufs in <i>subsections</i> . . . . .	39
13	Fahrtverlauf unter Einhaltung der Mindestzeit . . . . .	41
14	Fahrtverlauf mit reduzierter Geschwindigkeit unter Einhaltung der Ankunftszeit . . . . .	43
15	Fahrtverlauf vor der Anpassung der exakten Ankunftszeit . . . . .	44
16	Fahrtverlauf nach der Anpassung der exakten Ankunftszeit . . . . .	45
17	Ergebnis der Fahrtverlaufs-Ermittlung . . . . .	46
18	Fahrtverlauf für eine Beispielrechnung . . . . .	49



## Tabellenverzeichnis

1	Beschreibung der wichtigsten Tabellen der <i>MySQL</i> -Datenbank . . . . .	3
2	Aufbau eines Arrays in <i>next_betriebsstellen_data</i> . . . . .	10
3	Aufbau des <i>zeiten</i> -Arrays in <i>next_betriebsstellen_data</i> . . . . .	11
4	Aufbau eines Eintrags aus dem <i>\$allTimes</i> -Array . . . . .	15
5	Verhalten eines Fahrzeugs nach dem Erreichen des Ziels . . . . .	17
6	Übersicht der Fehlermeldungen . . . . .	22
7	Beschreibung der verwendeten Variablen für die Fahrtverlaufsberechnung	24
8	Exemplarische Infra-Abschnitte . . . . .	25
9	Exemplarische Zugdaten . . . . .	25
10	Aufbau des <i>\$subsection</i> -Arrays . . . . .	40
11	Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung vor der Anpassung . . . . .	45
12	Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung nach der Anpassung . . . . .	45
13	<i>\$keyPoints</i> am Beispiel von der Fahrt von XAB nach XZO . . . . .	48
14	Fahrtverlauf am Beispiel von der Fahrt von XAB nach XZO . . . . .	50

## Code-Beispiele

1	Initialisierung der Cache Variablen ( <i>fahrzeugsteuerung.php</i> ) . . . . .	7
2	Ermittlung der Real- und Simulationszeit ( <i>fahrzeugsteuerung.php</i> ) . . .	8
3	<i>getCalibratedPosition()</i> ( <i>functions_db.php</i> ) . . . . .	19
4	<i>showErrors()</i> ( <i>functions.php</i> ) . . . . .	21
5	<i>getVMaxBetweenTwoPoints()</i> ( <i>functions_fahrtverlauf.php</i> ) . . . . .	28
6	<i>createTrainChanges()</i> ( <i>functions_fahrtverlauf.php</i> ) . . . . .	30
7	<i>checkIfTrainIsToFastInCertainSections()</i> ( <i>functions_fahrtverlauf.php</i> ) .	36
8	<i>safeTrainChangeToJSONFile()</i> ( <i>functions_fahrtverlauf.php</i> ) . . . . .	52
9	<i>getBrakeDistance()</i> ( <i>functions_math.php</i> ) . . . . .	55
10	<i>getBrakeTime()</i> ( <i>functions_math.php</i> ) . . . . .	56
11	<i>getTargetBrakeSpeedWithDistanceAndStartSpeed()</i> ( <i>functions_math.php</i> )	56
12	<i>distanceWithSpeedToTime()</i> ( <i>functions_math.php</i> ) . . . . .	57
13	<i>calculateDistanceforSpeedFineTuning()</i> ( <i>functions_math.php</i> ) . . . . .	57

## **Abkürzungsverzeichnis**

**EBuEf** Eisenbahn-Betriebs- und Experimentierfeld

**Infra-Abschnitt** Infrastrukturabschnitt

## Glossar

**Beharrungsfahrt** Beschreibt den Teil des Fahrtverlaufs, bei dem die Geschwindigkeit des Fahrzeugs konstant ist.

**Echtzeitdaten** Die Echtzeitdaten beschreiben für jedes Fahrzeug die Position und Geschwindigkeit bei Beschleunigungen und Verzögerungen in  $2km/h$ -Schritten und bei konstanter Geschwindigkeit in in regelmäßigen Distanz-Intervallen in Abhängigkeit von der Simulationszeit.

**Fahrstraße** Beschreibt den Weg, der für das Fahrzeug durch die Stellung der Weichen vorgegeben ist.<sup>1</sup>

**Fahrtverlauf** Der Fahrtverlauf beschreibt die Positionen, Zeiten und Geschwindigkeiten für alle Beschleunigungs- und Bremsvorgänge und den Fahrten auf einer konstanten Geschwindigkeit eines Fahrzeugs von der aktuellen Position bis zum nächsten Halt.

**Realzeit** Die Realzeit beschreibt die Zeit des Rechners/Servers, auf dem die Fahrzeugsteuerung ausgeführt wird.

**Simulationszeit** Die Simulationszeit beschreibt die aktuelle Zeit, an der sich die Fahrzeuge orientieren (Startzeit, Ankunfts- und Abfahrtszeiten etc.). Die Startzeit der Simulation kann in der Session festgelegt werden und beginnt, sobald die Session gestartet wird.

**Unix-Timestamp** Die Unixzeit zählt die Sekunden, die seit dem 1. Januar 1970 00:00 (UTC) vergangen sind und wird im Unix-Timestamp-Format angegeben.<sup>2</sup>

**Zug-ID** Die Zug-ID ordnet den Fahrzeugen die Fahrpläne zu und ist nicht mit der ID des Fahrzeugs, welche dem Eintrag der *id*-Spalte aus der *MySQL*-Tabelle *fahrzeuge* entspricht, zu verwechseln

---

<sup>1</sup> Maschek (2018, S. 114)

<sup>2</sup> The IEEE and The Open Group (2018)

# 1 Einleitung

In dieser Arbeit wird eine Fahrzeugsteuerung für das Eisenbahn-Betriebs- und Experimentierfeld (EBuEf) entwickelt und dokumentiert. Das EBuEf ist eine Einrichtung des Fachgebiets *Bahnbetrieb und Infrastruktur* der Technischen Universität Berlin und bietet die Möglichkeit theoretisch erlerntes Wissen realitätsnah zu vertiefen.<sup>3</sup>

Für die Dokumentierung werden in Kapitel 2 die Grundlagen, die Ausgangssituation, die Herangehensweise und die Ziele beschrieben. Die Funktionsweise der Fahrzeugsteuerung wird in Kapitel 3 in chronologischer Form beschrieben, wobei im Kapitel 4 die Ermittlung des Fahrtverlaufs im Detail beschrieben wird. Damit die Allgemeingültigkeit der Fahrtverlaufsrechnung in Kapitel 4 gezeigt werden kann, wurden Infrastrukturdaten verwendet, die in dieser Form im EBuEf nicht vorkommen. Aus diesem Grund wird in Kapitel 5 die Funktionsweise anhand eines Beispiels im EBuEf gezeigt und mit Hilfe der in Kapitel 7 hergeleiteten Formeln auf die Richtigkeit überprüft.

Der Quellcode der Fahrzeugsteuerung befindet sich im Anhang der Arbeit und wird nur in Ausschnitten innerhalb der Arbeit abgebildet, wenn das der Erläuterung der Funktionsweise dient. Im Quellcode der Fahrzeugsteuerung wird auf Funktionen zugegriffen, welche bereits vorhanden waren und als Grundlage gedient haben. Diese Funktionen werden bei der Erwähnung mit einem Sternchen (\*) gekennzeichnet und nicht näher erläutert.

---

<sup>3</sup> *EBuEf: Eisenbahn-Betriebs- und Experimentierfeld Berlin* (2021)

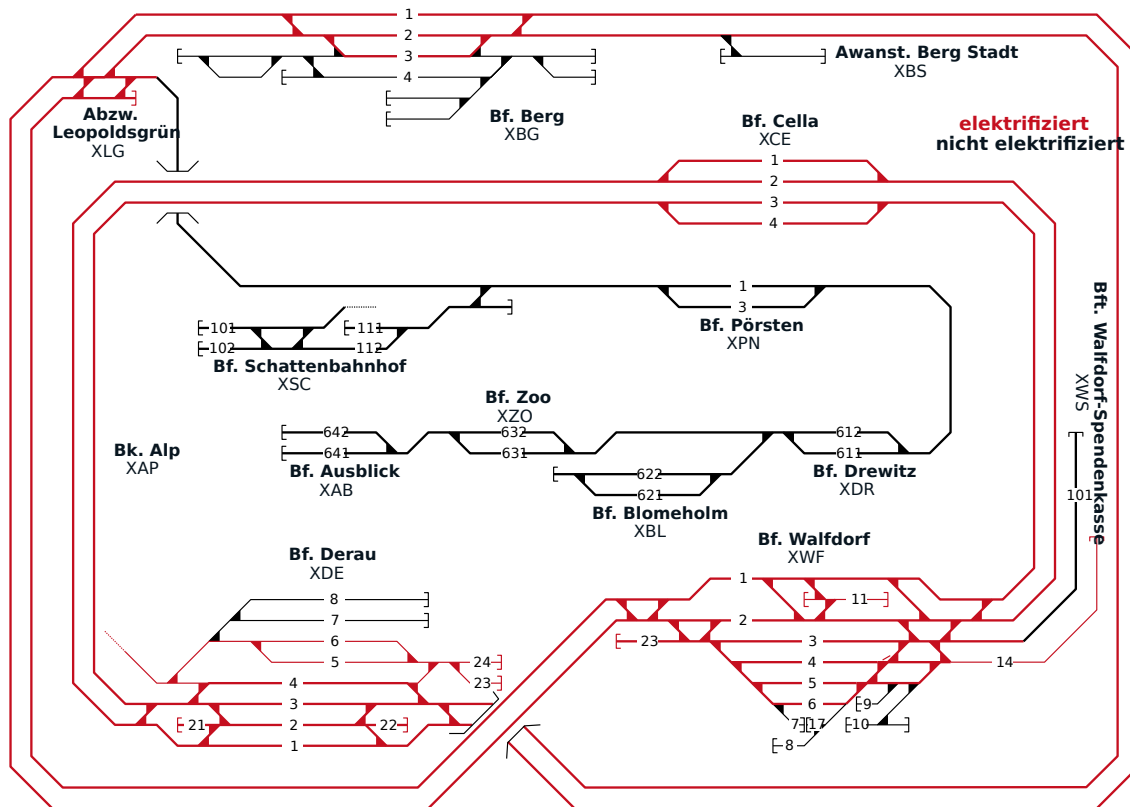


Abbildung 1: Schienennetz des EBUefs (Quelle: [www.ebuef.de/das-betriebsfeld/stellwerke](http://www.ebuef.de/das-betriebsfeld/stellwerke); Letzter Zugriff am: 4. September 2021)

## 2 Grundlagen

### 2.1 Aufbau des Eisenbahn-Betriebs- und Experimentierfelds

Das EBUef ist in ein eingleisiges nicht-elektrifiziertes und ein zweigleisiges elektrifiziertes Streckennetz unterteilt, welche über die Betriebsstelle Leopoldsgrün (XLG) miteinander verbunden sind. In der Abbildung 1 ist das eingleisige Netz in schwarz dargestellt und die zweigleisige Hauptstrecke in rot. Das eingleisige Netz ist in Infrastrukturabschnitte (Infra-Abschnitte) – welche mit Blockstrecken vergleichbar sind und eine Zugfolge im festen Raumabstand ermöglichen – eingeteilt.<sup>4</sup> Die Infra-Abschnitte sind mit der RailCom-Technik ausgestattet, welche über Decoder in den Fahrzeugen den aktuellen Infra-Abschnitt ermittelt und diesen in der *fma*-Tabelle der *MySQL*-Datenbank speichert.<sup>5</sup> Zudem sind in der Datenbank alle Informationen über die Infrastruktur gespeichert. Für die Fahrzeugsteuerung essentiell sind dabei die aktuellen Signalbegriffe

<sup>4</sup> Pachl (2021, S. 7, 42)

<sup>5</sup> RailCom - DCC-Rückmeldeprotokoll (2019)

Name	Beschreibung
<i>fahrplan_sessionfahrplan</i>	Fahrpläne der Session für alle Fahrzeuge
<i>fahrzeuge</i>	Fahrzeuge
<i>fahrzeuge_baureihen</i>	Baureiheninformationen
<i>fahrzeuge_daten</i>	Statische Daten der Fahrzeuge
<i>fma</i>	Freimeldeabschnitte
<i>gbt_fma</i>	Zuordnung der GBT-Abschnitte, FMA-Abschnitte und Infra-Abschnitte
<i>infra_daten</i>	Statische Daten der Infrastruktur
<i>infra_zustand</i>	Zustand der Infrastruktur
<i>signale</i>	Standorte der Signale

Tabelle 1: Beschreibung der wichtigsten Tabellen der *MySQL*-Datenbank

aller Signale und die Längen der Infra-Abschnitte.<sup>6</sup>

Der Betrieb des EBUefs erfordert eine angelegte Session, welche vor dem Start der Fahrzeugsteuerung gestartet werden muss, da die Fahrzeugsteuerung beim Start alle benötigten Informationen der Session einliest.

## 2.2 Aufbau der *MySQL*-Datenbank

Alle Informationen und Daten, die für den Betrieb der Fahrzeugsteuerung benötigt werden, sind in einer *MySQL*-Datenbank gespeichert. In der Tabelle 1 werden die wichtigsten Tabellen der Datenbank aufgelistet und kurz beschrieben.

## 2.3 Ziele und Prioritätssetzung der Fahrzeugsteuerung

Oberste Priorität der Fahrzeugsteuerung hat eine möglichst effiziente Umsetzung und das Einhalten der vorgegebenen Fahrpläne. Für eine effiziente Umsetzung wurden die Zugriffe auf die *MySQL*-Datenbank während des laufenden Betriebs der Fahrzeugsteuerung möglichst gering gehalten und Teile des Quellcodes, welche häufiger verwendet werden, wurden in Funktionen ausgelagert. Die Ermittlung der Fahrtverläufe berücksichtigt für die Einhaltung der Fahrplanzeiten neben den Ankunfts- und Abfahrtszeiten auch die aktuelle Verspätung und versucht diese auszugleichen.

An zweiter Stelle der Prioritätssetzung steht das energieeffiziente Fahren. Damit die Fahrten möglichst energieeffizient sind, fahren die Züge die kleinstmögliche Ge-

<sup>6</sup> *EBUef: Eisenbahn-Betriebs- und Experimentierfeld Berlin* (2021)

geschwindigkeit, bei der das Ziel ohne eine Verspätung erreicht wird. Sollte auch bei der größtmöglichen Geschwindigkeit das Ziel mit einer Verspätung erreicht werden, wird diese Geschwindigkeit gewählt. In dem Fall, dass es für ein Fahrzeug möglich ist mit einer geringeren Geschwindigkeit zu fahren als die maximal zulässige Geschwindigkeit, wird die Geschwindigkeit möglichst am Ende des Fahrtverlaufs reduziert. Dadurch hat das Fahrzeug für den Fall einer Fahrstraßenänderung oder Reduzierung der zulässigen Höchstgeschwindigkeit einen größtmöglichen Zeitpuffer.

## 2.4 Fahrdynamik

In der Realität gibt es vier Bewegungsphasen, in denen sich ein Fahrzeug befinden kann:

- Anfahren
- Beharrungsfahrt
- Auslauf
- Bremsen

Beim Anfahren ist die Antriebskraft größer als die Summe der Widerstandskräfte, wodurch das Fahrzeug beschleunigt und in der Beharrungsfahrt entspricht die Antriebskraft der Summe der Widerstandskräfte, wodurch die Geschwindigkeit des Fahrzeugs konstant bleibt. Für die Reduzierung der Geschwindigkeit kann entweder die Antriebskraft gleich null sein oder eine Bremskraft aufgewendet werden.<sup>7</sup>

Die Widerstandskräfte setzen sich aus dem Streckenwiderstand, dem Fahrzeugwiderstand und dem Anfahrwiderstand zusammen und lassen sich mit den gegebenen Daten des EBUfs nicht vollständig berechnen.<sup>8</sup> Aus diesem Grund werden die Widerstandskräfte bei der Fahrzeugsteuerung nicht berücksichtigt und die Auslaufphase, welche nur von der Widerstandskräften abhängig ist, wird ebenfalls nicht berücksichtigt.

## 2.5 Aufbau des Projekts

In der Darstellung 2 ist der Aufbau des Projekts und die für die Arbeit relevanten Dateien/Ordner dargestellt. Dateien, welche bereits vorhanden waren, sind wie Funktionen

---

<sup>7</sup> Pachl (2021, S. 23 ff.)

<sup>8</sup> Pachl (2021, S. 25 ff.)



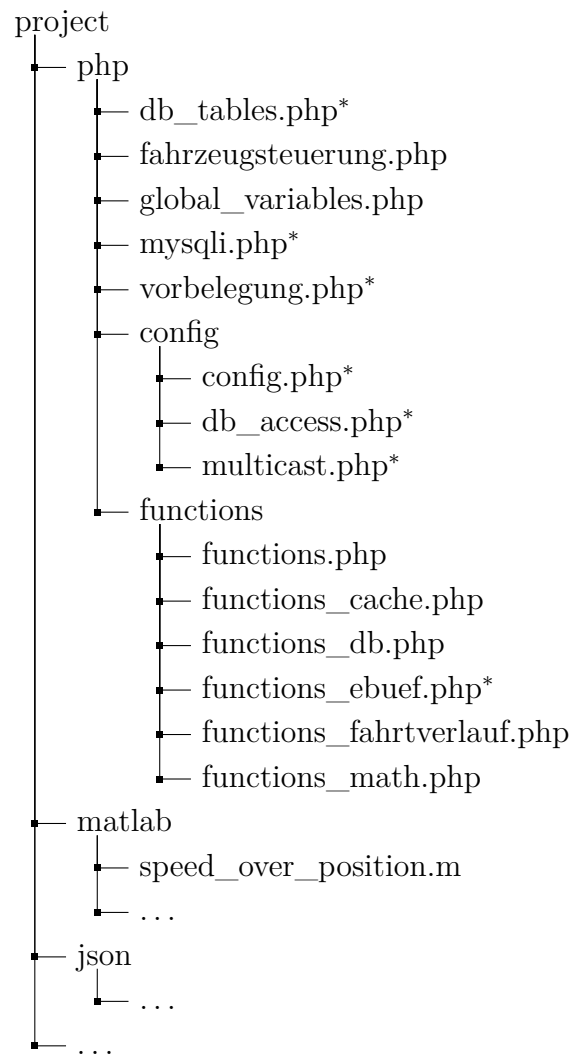


Abbildung 2: Aufbau der Dateistrukturen

mit einem Sternchen (\*) markiert. Die für die Fahrzeugsteuerung essentiellen Dateien befinden sich innerhalb des *php*-Ordners, wobei die Datei *fahrzeugsteuerung.php* die Fahrzeugsteuerung startet und für die Berechnung der Fahrtverläufe auf die Dateien in dem *functions*-Unterordner zugreift. Die benötigten Dateien für den Zugriff auf die *MySQL*-Datenbank befinden sich in dem *config*-Unterordner und global festgelegte Parameter sind in der Datei *global\_variables.php* abgespeichert. Für die Visualisierung (siehe Kapitel 6) der Fahrtverläufe werden die Dateien aus dem *matlab*- und *json*-Ordner benötigt.

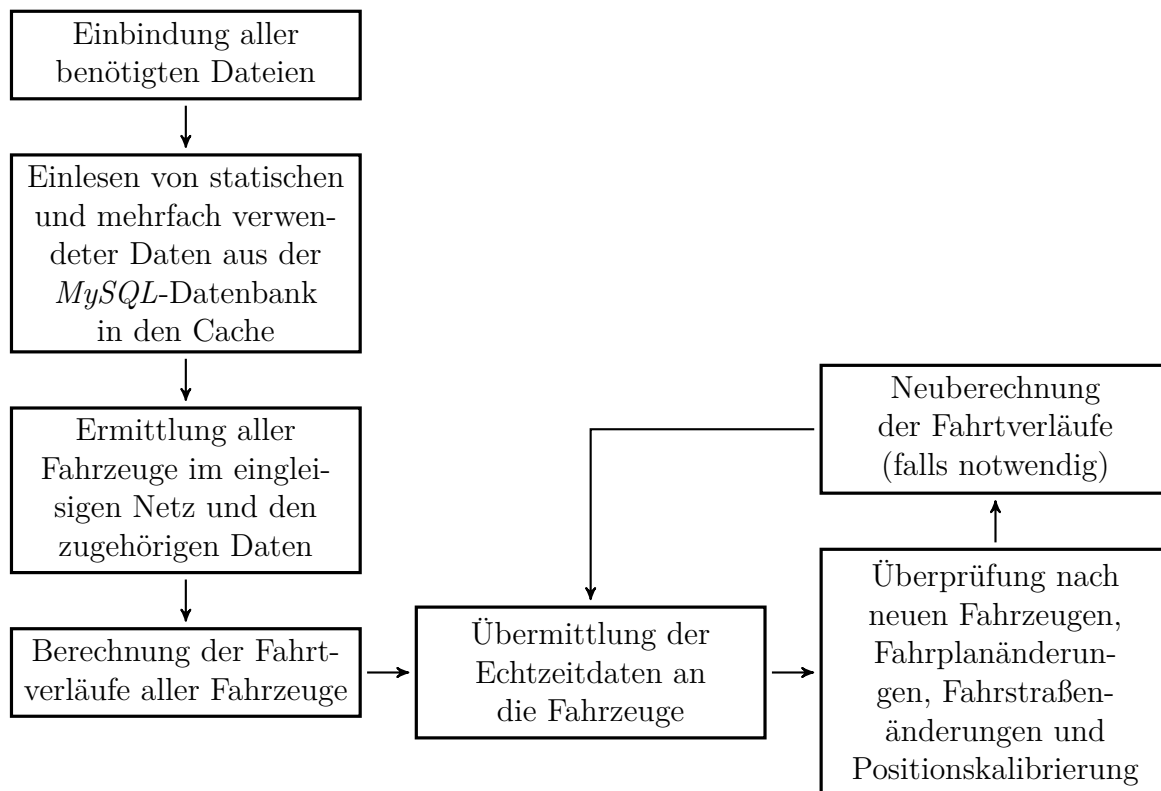


Abbildung 3: Ablauf der Fahrzeugsteuerung

### 3 Ablauf der Fahrzeugsteuerung

Damit die Fahrzeugsteuerung gestartet werden kann, muss die Datei *fahrzeugsteuerung.php* ausgeführt werden. Obligatorisch für die Fahrzeugsteuerung ist die Abschnittsüberwachung (*abschnittueberwachung.php*), welche vor dem Start der Fahrzeugsteuerung ausgeführt werden muss und auf deren Verwendung und Funktionsweise in Kapitel 3.7 eingegangen wird. Der Aufbau dieses Kapitels orientiert sich an dem Ablauf der Fahrzeugsteuerung, welcher in Abbildung 3 schematisch dargestellt wird.

#### 3.1 Einlesen von statischen und mehrfach verwendeten Daten aus der *MySQL*-Datenbank in den Cache

Die Fahrzeugsteuerung benötigt als Grundlage für viele Berechnungen Daten aus der *MySQL*-Datenbank. Damit diese Daten nicht bei jeder Verwendung erneut aus der Datenbank geladen werden müssen und somit die Anzahl an Datenbank-Abfragen möglichst gering gehalten werden kann, werden die wichtigsten Daten beim Programm-

start bzw. bei der ersten Verwendung in den Cache geladen (Code-Beispiel 1). Beispielfähig zu nennen sind hierbei *\$cacheInfraLaenge* (Länge aller Infra-Abschnitte in Metern), *\$cacheHaltepunkte* (zugehörige Infra-Abschnitte für alle Betriebsstellen und Richtung), *\$cacheZwischenhaltepunkte* (zugehörige Infra-Abschnitte für alle Zwischen-Betriebsstellen, die nur einem Infra-Abschnitt zugeordnet sind), *\$cacheGbtToInfra* (Zuordnung der Infra-Abschnitte zu den GBT-Abschnitten) und *\$cacheInfraToGbt* (Zuordnung der GBT-Abschnitte zu den Infra-Abschnitten).

```

1 // Statische Daten einlesen
2 $cacheInfranachbarn = createCacheInfranachbarn();
3 $cacheInfradaten = createCacheInfradaten();
4 $cacheSignaldaten = createCacheSignaldaten();
5 $cacheInfraLaenge = createcacheInfraLaenge();
6 $cacheHaltepunkte = createCacheHaltepunkte();
7 $cacheZwischenhaltepunkte = createChacheZwischenhaltepunkte();
8 $cacheInfraToGbt = createCacheInfraToGbt();
9 $cacheGbtToInfra = createCacheGbtToInfra();
10 $cacheFmaToInfra = createCacheFmaToInfra();
11 $cacheInfraToFma = array_flip($cacheFmaToInfra);
12 $cacheFahrplanSession = createCacheFahrplanSession();
13 $cacheSignalIDToBetriebsstelle = createCacheToBetriebsstelle();
14 $cacheFahrzeugeAbschnitte = createCacheFahrzeugeAbschnitte();
15 $cacheIDTDecoder = createCacheDecoderToAdresse();
16 $cacheDecoderToID = array_flip($cacheIDTDecoder);
17 $cacheAdresseToID = array(); // Filled with data in getAllTrains()
18 $cacheIDToAdresse = array(); // Filled with data in getAllTrains()

```

Code-Beispiel 1: Initialisierung der Cache Variablen (*fahrzeugsteuerung.php*)

### 3.2 Ermittlung der Session-Daten

In der *MySQL*-Tabelle *fahrplan\_session* sind alle Fahrplansessions aufgelistet und der aktuell gültigen wurde der Wert 1 in der *status*-Spalte zugeordnet. Die Daten der gültigen Fahrplansession wurden bei dem Start der Fahrzeugsteuerung in dem Array *\$cacheFahrplanSession* gespeichert und werden benötigt, um die Zeitdifferenz zwischen Real- und Simulationszeit zu ermitteln. Dafür wird im ersten Schritt das Datum der *sim\_startzeit* und *sim\_endzeit*, welche die Start- und Endzeit (Simulationszeit) der Simulation im Unix-Timestamp-Format angeben, auf das aktuelle Datum der Realzeit geändert und im zweiten Schritt mit der Realzeit verglichen (Code-Beispiel 2).

```

1 // Real- und Simulationszeit ermitteln
2 $simulationStartTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->
    ↳ sim_startzeit, "simulationszeit", null, array("outputtyp"=>"h:i:s")), "
    ↳ simulationszeit", null, array("inputtyp"=>"h:i:s"));
3 $simulationEndTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->
    ↳ sim_endzeit, "simulationszeit", null, array("outputtyp"=>"h:i:s")), "
    ↳ simulationszeit", null, array("inputtyp"=>"h:i:s"));
4 $simulationDuration = $cacheFahrplanSession->sim_endzeit -
    ↳ $cacheFahrplanSession->sim_startzeit;
5 $realStartTime = time();
6 $realEndTime = $realStartTime + $simulationDuration;
7 $timeDifference = $simulationStartTimeToday - $realStartTime;

```

Code-Beispiel 2: Ermittlung der Real- und Simulationszeit (*fahrzeugsteuerung.php*)

Das Datum der Simulationszeit wird angepasst, damit auch Fahrplansessions ausgeführt werden können, die nicht dem aktuellen Datum der Realzeit entsprechen. Für die Umwandlung des Datums werden die Zeiten mittels der Funktion *getUhrzeit()*\* in das *hh:mm:ss*-Format umgewandelt und mit derselben Funktion wieder in das Unix-Timestamp-Format zurück umgewandelt.

Für die Ermittlung der Realzeit und der Zeitdifferenz zwischen Real- und Simulationszeit wird die Funktion *time()* aufgerufen, mit der Start-Simulationszeit verglichen und unter der Variable *\$timeDifference* abgespeichert.

Die Differenz zwischen Real- und Simulationszeit ist essenziell, damit die Fahrzeuge zur richtigen Zeit die Echtzeitdaten übermittelt bekommen und die Realzeit in die Simulationszeit umgewandelt werden kann, ohne bei jeder Umrechnung die Funktion *getUhrzeit()*\* aufzurufen.

### 3.3 Ermittlung aller Fahrzeuge im eingleisigen Netz und den zugehörigen Daten

Das eingleisige Netz des EBUfs kann mittels der RailCom-Technik und den Decodern in den Fahrzeugen ermitteln, welches Fahrzeug aktuell welche Infra-Abschnitte belegt. Belegt ein Fahrzeug einen Infra-Abschnitt, wird in der Tabelle *fma* in der Spalte *decoder\_adresse* die Adresse des Fahrzeugs hinterlegt und in der *infra\_zustand*-Tabelle in der Spalte *dir* der Wert 1 hinterlegt. Durch diese Informationen werden alle Fahrzeuge, die sich beim Start des Programms im eingleisigen Netz befinden, mit der Funktion *findTrainsOnTheTracks()* (*functions.php*) eingelesen und die zugehörige Adresse wird der Funktion *prepareTrainForRide()* (*functions.php*) übergeben. Für jedes Fahrzeug, welches dieser Funktion übergeben wird, wird in dem Array *\$allUsedTrains* ein neuer

Eintrag erstellt, für jedes Fahrzeug die exakte Position bestimmt und der Fahrplan geladen. Das Array *\$allUsedTrains* beinhaltet alle Fahrzeuge, die aktuell von der Fahrzeugsteuerung berücksichtigt werden und deren zugehörige Informationen, wobei der Index der ID des Fahrzeugs entspricht.

Bei der Positionsbestimmung wird davon ausgegangen, dass die Fahrzeuge direkt vor dem zugehörigen Signal stehen, da ansonsten die Position nicht exakt ermittelt werden kann. Belegt ein Fahrzeug mehrere Infra-Abschnitte, wird mittels der Fahrtrichtung der Züge der Infra-Abschnitt ermittelt, in dem sich der Zugkopf befindet. Die aktuelle Position wird daraufhin mit dem Infra-Abschnitt und der relativen Position (in Metern) innerhalb des Abschnitts angegeben. Es wird davon ausgegangen, dass das Fahrzeug sich direkt vor dem Signal befindet, wodurch die relative Position der Infra-Abschnittslänge entspricht.

Für die Überprüfung, ob ein Fahrzeug nach Fahrplan fährt, wird die Funktion *getFahrzeugZugIds()*\* (*functions\_ebuef.php*) aufgerufen. Wenn einem Fahrzeug kein Fahrplan zugewiesen wurde (Rückgabewert der Funktion *getFahrzeugZugIds()*\* (*functions\_ebuef.php*) ist ein leeres Array), wird in dem *\$allUsedTrains*-Array dem Fahrzeug unter dem Eintrag *operates\_on\_timetable* der Wert *false* zugewiesen. In dem Fall, dass für das Fahrzeug ein Fahrplan hinterlegt ist (Rückgabewert der Funktion *getFahrzeugZugIds()*\* (*functions\_ebuef.php*) ist ein Array mit allen Zug-IDs), wird mittels der Funktion *getNextBetriebsstellen()* (*functions.php*) der Fahrplan für den ersten Eintrag des Zug-ID Arrays aus der Datenbank geladen. Der Fahrplan wird in dem *\$allUsedTrains*-Array in dem *next\_betriebsstellen\_data*-Array hinterlegt, welches für jede Betriebsstelle ein Array mit den benötigten Daten enthält. Die Indizierung dieser Einträge entspricht dabei den natürlichen Zahlen in aufsteigender Reihenfolge angefangen bei der 0 ( $\mathbb{N}_0$ ). Hierbei werden alle Betriebsstellen hinzugefügt, bei denen ein fahrplanmäßiger Halt vorgesehen ist. Damit ein Fahrzeug nicht erst losfahren kann, wenn die Fahrstraße bis zur nächsten Betriebsstelle mit fahrplanmäßigem Halt gestellt ist, werden auch alle Betriebsstellen ohne fahrplanmäßigem Halt hinzugefügt, welche eindeutig einem Infra-Abschnitt zugeordnet sind (*\$cacheZwischenhaltepunkte*). Das hat den Vorteil, dass Fahrzeuge losfahren können, auch wenn die Fahrstraße noch nicht bis zum nächsten fahrplanmäßigen Halt gestellt ist, das aber nur machen, wenn sichergestellt werden kann, dass die Zwischen-Betriebsstelle auf der Strecke zum nächsten fahrplanmäßigen Halt liegt. In Tabelle 2 ist für eine bessere Übersicht der Aufbau eines Betriebsstellen-Eintrags abgebildet. Für die Ermittlung der Ankunfts- und Abfahrzeiten wird die Funktion *getFahrplanzeiten()*\* (*functions\_ebuef.php*) aufgerufen, welche als Parameter den Namen der Betriebsstelle und die Zug-ID übergeben bekommt. Die zurückgegebenen Daten werden unter dem Eintrag *zeiten* abgespeichert und um

Bezeichnung	Funktion
<i>is_on_fahrstrasse</i> (Boolescher Wert)	Befindet sich die Betriebsstelle auf der Fahrstraße
<i>betriebsstelle</i> (String)	Name der Betriebsstelle
<i>zeiten</i> (Array)	Verspätung und Ankunfts- und Abfahrtszeiten (siehe Tabelle 3)
<i>haltepunkte</i> (Array)	Alle zugehörigen Infra-Abschnitte
<i>fahrplanhalt</i> (Boolescher Wert)	Ist diese Betriebsstelle ein Fahrplanhalt

Tabelle 2: Aufbau eines Arrays in *next\_betriebsstellen\_data*

den Eintrag *verspaetung* ergänzt. Zudem werden die Ankunfts- und Abfahrtszeiten in das Unix-Timestamp-Format mittels der Funktion *getUhrzeit()*\* (*functions\_ebuef.php*) umgewandelt. Der Aufbau des *zeiten*-Arrays ist in der Tabelle 3 dargestellt. Für die Überprüfung, ob eine Betriebsstelle durch die aktuelle Fahrstraße erreichbar ist, müssen den Betriebsstellen die Infra-Abschnitte zugeordnet werden. Dafür werden mit Hilfe der Arrays *\$cacheZwischenhaltepunkte* und *\$cacheHaltepunkte*, jeder Betriebsstelle mögliche Infra-Abschnitte zugeordnet. Die Arrays sind so aufgebaut, dass jeder Betriebsstelle für jede Richtung alle Infra-Abschnitte zugeteilt sind, welchen ein Ausfahrtsignal zugeordnet ist.

Nach der Zuordnung der Infra-Abschnitte zu den Betriebsstellen, wird anhand der aktuellen Positionen der Fahrzeuge überprüft, ob die Fahrzeuge an einer Betriebsstelle des Fahrplans stehen. Stimmt der aktuelle Infra-Abschnitt eines Fahrzeugs mit dem einer Betriebsstelle überein, wird dieser und allen vorherigen der Wert *true* unter der Variablen *angekommen* zugewiesen. Dadurch können Fahrzeuge auch nach Fahrplan fahren, wenn diese nicht an der ersten Betriebsstelle des Fahrplans stehen.

### 3.4 Berechnung der Fahrtverläufe aller Fahrzeuge

Nachdem für alle Fahrzeuge die Fahrplandaten (falls vorhanden) hinterlegt wurden, wird für jedes Fahrzeug die aktuelle Fahrstraße ermittelt. Dafür wird die Funktion *calculateNextSections()* (*functions.php*) aufgerufen und das Array *\$allUsedTrains* für jedes Fahrzeug um die Einträge *next\_sections*, *next\_lenghts* und *next\_v\_max* als Array ergänzt. Diese Arrays speichern die IDs, Längen und zulässigen Höchstgeschwindigkeiten der nächsten Infra-Abschnitte ab, welche auf der Fahrstraße liegen.

Im ersten Schritt wird überprüft, ob das Fahrzeug aktuell in einem Infra-Abschnitt steht, welchem ein auf Halt stehendes Signal zugeordnet ist. Wenn das der Fall ist, wird

Bezeichnung	Funktion
<i>ankunft_soll</i> (String)	Ankunftszeit (hh:mm:ss)
<i>abfahrt_soll</i> (String)	Abfahrtszeit (hh:mm:ss)
<i>ankunft_soll_timestamp</i> (Integer)	Ankunftszeit (Unixtimestamp)
<i>abfahrt_soll_timestamp</i> (Integer)	Abfahrtszeit (Unixtimestamp)
<i>fahrtrichtung</i> (Array)	Fahrtrichtung (Eintrag aus der Tabelle <i>fahrplan_sessionfahrplan</i> )
<i>ist_durchfahrt</i> (Integer)	Fahrplanhalt (Eintrag aus der Tabelle <i>fahrplan_sessionfahrplan</i> )
<i>used_haltepunkt</i> (Integer)	Infra-Abschnitt der Betriebsstelle, welcher auf der Fahrstraße liegt
<i>wendet</i> (Integer)	Wendeauftrag nach Erreichen der Betriebsstelle
<i>verspaetung</i> (Integer)	Verspätung, mit der das Fahrzeug diese Betriebsstelle erreicht hat

Tabelle 3: Aufbau des *zeiten*-Arrays in *next\_betriebsstellen\_data*

den Arrays *next\_sections*, *next\_lenghts* und *next\_v\_max* ein leeres Array zugewiesen. Wenn das Fahrzeug aktuell nicht in einem Abschnitt steht, welchem ein auf Halt stehendes Signal zugeordnet ist, wird über die Funktion *getNaechsteAbschnitte()*\* (*functions\_ebuef.php*) die aktuelle Fahrstraße ermittelt und der Rückgabewert der Funktion *getNaechsteAbschnitte()*\* (*functions\_ebuef.php*) in dem *\$allUsedTrains*-Array unter dem Eintrag *last\_get\_naechste\_abschnitte* gespeichert. Diese Speicherung ist notwendig, um zu überprüfen, ob sich die Fahrstraße geändert hat.

Nach der Ermittlung der Fahrstraße und der Zuordnung der Infra-Abschnitte zu den Betriebsstellen wird im nächsten Schritt überprüft, welche Betriebsstellen des Fahrplans auf der aktuellen Fahrstraße liegen. Dafür iteriert die Funktion *checkIfFahrstrasseIsCorrect()* (*functions.php*) in aufsteigender Reihenfolge über alle Betriebsstellen der Fahrzeuge und die *haltepunkte* der Betriebsstellen werden mit den Werten aus dem Array *next\_sections* verglichen. Bei jedem Aufruf der Funktion wird dem Fahrzeug anfangs (falls das Fahrzeug nach Fahrplan fährt) in dem Array *\$allUsedTrains* der Eintrag *fahrstrasse\_is\_correct* der Wert *false* zugewiesen und erst auf *true* gesetzt, wenn eine Betriebsstelle auf der Fahrstraße liegt. Bei dem Iterieren über die Betriebsstellen wird jeder Betriebsstelle anfangs der Wert *false* für den Eintrag *is\_on\_fahrstrasse* zugeordnet und sobald ein Infra-Abschnitt einer Betriebsstelle in dem Array *next\_sections* ebenfalls vorhanden ist, wird dem Eintrag *is\_on\_fahrstrasse* der Wert *true* zugewiesen und unter dem Eintrag *used\_haltepunkt* der Infra-Abschnitt gespeichert, welcher auf

der Fahrstraße liegt. Bei dem Iterieren über alle Betriebsstellen werden nur die Betriebsstellen beachtet, welche das Fahrzeug noch nicht erreicht hat (*angekommen* == *false*). Für Fahrzeuge ohne Fahrplan wird der Eintrag *fahrstrasse\_is\_correct* direkt auf *true* gesetzt.

Durch die Ermittlung der Fahrstraße kann für jedes Fahrzeug der Fahrtverlauf berechnet werden. Für die Berechnung der Fahrtverläufe wird für jedes Fahrzeug die Funktion *calculateFahrtverlauf()* (*functions.php*) aufgerufen und innerhalb der Funktion überprüft, ob die Fahrstraße richtig eingestellt ist (*fahrstrasse\_is\_correct* == *true*). Wenn die Fahrstraße richtig eingestellt ist, wird zwischen Fahrzeugen unterschieden, die nach Fahrplan fahren und Fahrzeugen, die keinen Fahrplan haben.

Für Fahrzeuge mit Fahrplan muss im ersten Schritt die nächste Betriebsstelle ermittelt werden, an der das Fahrzeug anhalten muss. Dafür wird mit einer *for*-Schleife über alle in *next\_betriebsstellen\_data* hinterlegten Betriebsstellen iteriert, die das Fahrzeug noch nicht angefahren hat (*angekommen* == *false*), die auf der Fahrstraße liegen (*is\_on\_fahrstrasse* == *true*) und die ein fahrplanmäßiger Halt sind (*fahrplanhalt* == *true*). Sobald eine Betriebsstelle gefunden wurde, wird die *for*-Schleife abgebrochen und der Index der Betriebsstelle als *\$nextBetriebsstelleIndex* abgespeichert. Sollte unter den nächsten Betriebsstellen keine dabei sein, auf die diese Kriterien zutreffen, wird in einer zweiten *for*-Schleife nach den selben Kriterien (außer dem des fahrplanmäßigen Halts) nach einer Betriebsstelle gesucht und sobald eine Betriebsstelle gefunden wurde, wird die Schleife abgebrochen und der Index der Betriebsstelle unter der Variablen *\$nextBetriebsstelleIndex* abgespeichert. Sollte eine nächste Betriebsstelle für das Fahrzeug existieren wird in einer dritten *for*-Schleife überprüft, ob zwischen der aktuellen Position und der nächsten Betriebsstelle eine Betriebsstelle ist, bei der das Fahrzeug einen Wendeauftrag bekommt. Sollte eine solche Betriebsstelle existieren, wird diese unter der Variablen *\$nextBetriebsstelleIndex* abgespeichert. In dem Fall, dass keine nächste Betriebsstelle ermittelt werden konnte und das Fahrzeug aktuell eine Geschwindigkeit hat, für die gilt:  $v > 0 \text{ km/h}$ , wird eine Gefahrenbremsung eingeleitet (siehe Kapitel 4.9).

Für alle Fahrzeuge, für die eine nächste Betriebsstelle ermittelt werden konnte, werden im Folgenden alle notwendigen Daten ermittelt. Dazu zählt, ob die Fahrzeuge nach dem Erreichen der Betriebsstelle einen Wendeauftrag erhalten sollen (*wendet*-Eintrag der nächsten Betriebsstelle), in welchen Infra-Abschnitt das Fahrzeug zum Stehen kommen soll (*used\_haltepunkt*-Eintrag der nächsten Betriebsstelle) und an welcher relativen Position innerhalb des Abschnitts das Fahrzeug angehalten soll (Länge des Infra-Abschnitts). Neben den Informationen zur Position müssen die Informationen



zur Zeit ermittelt werden.

Für die Ermittlung der Ankunftszeit muss neben dem zugehörigen Eintrag *ankunft\_soll\_timestamp* der Betriebsstelle die Verspätung berücksichtigt werden. Aus diesem Grund wird im ersten Schritt die zuletzt angefahren Betriebsstelle unter der Variablen *\$prevBetriebsstelle* abgespeichert. Sollte die nächste Betriebsstelle der erste fahrplanmäßige Halt sein (Ankunftszeit nicht definiert), so wird als Start- und Zielzeit (*\$startTime* und *\$endTime*) die aktuelle Simulationszeit verwendet. Wenn die nächste Betriebsstelle nicht dem ersten fahrplanmäßigen Halt entspricht, wird als Zielzeit die Ankunftszeit der Betriebsstelle festgelegt und als Startzeit die Abfahrtszeit der vorherigen Betriebsstelle (*\$prevBetriebsstelle*) plus die eingetragene Verspätung der vorherigen Betriebsstelle. Sollte es zu dem Zeitpunkt der Berechnung keine vorherige Betriebsstelle geben (*\$prevBetriebsstelle == null*), so wird als Startzeit die aktuelle Simulationszeit gewählt. Im zweiten Schritt wird überprüft, ob die Startzeit kleiner als die aktuelle Simulationszeit ist und wenn das der Fall ist, wird die Startzeit gleich der Simulationszeit gesetzt. Im dritten Schritt wird die Startzeit gleich der frühestmöglichen Startzeit des Fahrzeugs (*earliest\_possible\_start\_time*-Eintrag des Fahrzeugs) gesetzt, falls die Startzeit kleiner ist. Der Eintrag *earliest\_possible\_start\_time* der Züge gibt die frühestmögliche Abfahrtszeit der Züge an und wird zum Beispiel bei einem Wendeauftrag auf die aktuelle Simulationszeit gesetzt und um 30 s erhöht.

Für alle Fahrzeuge, die ohne Fahrplan unterwegs sind, wird als Ziel-Infra-Abschnitt der letzte Infra-Abschnitt aus dem Array *last\_get\_naechste\_abschnitte* verwendet, welchem ein Signal zugeordnet ist. Die Ziel-Position innerhalb des Infra-Abschnitts entspricht dabei ebenfalls der Länge des Abschnitts und die Überprüfung, ob ein Wendeauftrag nach dem Erreichen des Ziel-Infra-Abschnitt dem Fahrzeug übermittelt werden soll, wird von dem Signalbegriff abgeleitet. Die Start- und Zielzeit entsprechen der aktuellen Simulationszeit, bzw. der *earliest\_possible\_start\_time*. Sollte keinem der nächsten Infra-Abschnitte aus dem *last\_get\_naechste\_abschnitte*-Array ein Signal zugeordnet sein und die aktuelle Geschwindigkeit des Fahrzeugs ist größer als 0 km/h sein, so wird eine Gefahrenbremsung eingeleitet. Andernfalls wird die Funktion an dieser Stelle abgebrochen und es wird wieder versucht einen Fahrtverlauf zu berechnen, wenn sich die Fahrstraße geändert hat.

Nach der Ermittlung aller notwendigen Daten für die Berechnung des Fahrtverlaufs, wird für jedes Fahrzeug die Funktion *updateNextSpeed()* (*functions\_fahrtverlauf.php*) aufgerufen, welche den Fahrtverlauf berechnet und in Kapitel 4 im Detail beschrieben wird. Wichtig an dieser Stelle ist der Rückgabewert der Funktion, welcher für Fahrzeuge mit Fahrplan die Verspätung in Sekunden angibt, mit der das Fahrzeug die

Ziel-Betriebsstelle erreicht, und wird unter dem Eintrag *verspaetung* der zugehörigen Betriebsstelle gespeichert. Ob ein Fahrzeug eine Betriebsstelle mit einer Verspätung erreicht, kann nur ermittelt werden, wenn die Ankunftszeit definiert ist. Für den Fall, dass für ein Fahrzeug ein Fahrplan hinterlegt ist, das Fahrzeug in einem Infra-Abschnitt steht, welchem keine Betriebsstelle des Fahrplans zugeordnet ist und die Fahrstraße so eingestellt ist, dass das Fahrzeug den ersten fahrplanmäßigen Halt anfahren könnte, kann nicht ermittelt werden, ob das Fahrzeug diese Betriebsstelle mit einer Verspätung erreicht, da für den ersten fahrplanmäßigen Halt in der *MySQL*-Tabelle *fahrplan\_sessionfahrplan* keine Ankunftszeit hinterlegt ist. Aus diesem Grund, wurde in der Datei *global\_variables.php* die Variable *\$globalFirstHaltMinTime* definiert, welche angibt, wie lange ein Fahrzeug an der ersten Betriebsstelle des Fahrplans halten soll. Wenn diese Zeit eingehalten werden kann, wird das Fahrzeug (sofern die Fahrstraße richtig eingestellt ist) zur Abfahrtszeit die Betriebsstelle verlassen. Andernfalls gilt für die Verspätung der ersten Betriebsstelle:

$$\text{Verspätung} = \text{Ankunftszeit} + \$globalFirstHaltMinTime - \text{Abfahrtszeit}$$

### 3.5 Übermittlung der Echtzeitdaten an die Fahrzeuge

Nach dem Aufruf der Funktion *updateNextSpeed()* (*functions\_fahrtverlauf.php*) sind für alle Fahrzeuge – für die ein Fahrtverlauf berechnet wurde – in dem Array *\$allTimes* alle Echtzeitdaten enthalten. Das Array beinhaltet für jedes Fahrzeug wiederum ein Array, welches unter der Adresse des Fahrzeugs abgespeichert ist, und beinhaltet alle Echtzeitdaten eines Fahrzeugs. Der Aufbau eines Array mit Echtzeitdaten ist in Tabelle 4 dargestellt. In einer *while*-Schleife wird über alle Einträge des *\$allTimes*-Arrays iteriert und überprüft, ob der erste Eintrag eines Fahrzeugs Echtzeitdaten enthält, welche an das Fahrzeug übermittelt werden müssen. Dafür wird der Eintrag *live\_time* mit der aktuellen Simulationszeit verglichen und die zugehörigen Echtzeitdaten an das Fahrzeug übermittelt, wenn der Eintrag *live\_time* kleiner als die aktuelle Simulationszeit ist. Nach jedem Durchlauf der *while*-Schleife wird diese mit der Funktion *sleep()* für 0,03 s pausiert. An dieser Stelle wurde sich für einen Wert von 0,03 s entschieden, da so die Position auf einen Meter genau bestimmt werden kann, wenn das Fahrzeug eine Geschwindigkeit von 120 km/h hat.

Wenn für ein Fahrzeug neue Echtzeitdaten vorliegen, wird im ersten Schritt überprüft, ob eine Geschwindigkeitsveränderung vorliegt (*live\_is\_speed\_change == true*) und die neue Geschwindigkeit (falls vorhanden) über die Funktion *sendFahrzeugbefehl()*\* (*functions\_ebuef.php*) dem Fahrzeug übergeben und mittels einer Terminal-Ausgabe

Bezeichnung	Funktion
<i>live_position</i> (Float)	absolute Position (kann weg...)
<i>live_speed</i> (Integer)	Geschwindigkeit des Fahrzeugs
<i>live_time</i> (Float)	Zeit der Übermittlung an das Fahrzeug
<i>live_relative_position</i> (Integer)	relative Position im Infra-Abschnitt
<i>live_section</i> (Integer)	Infra-Abschnitt
<i>live_is_speed_change</i> (Boolescher Wert)	Angabe, ob bei diesen Echtzeitdaten die Geschwindigkeit verändert wird
<i>live_target_reached</i> (Boolescher Wert)	Das Fahrzeug hat sein Ziel erreicht
<i>id</i> (String)	ID des Zugs
<i>wendet</i> (Boolescher Wert)	Angabe, ob ein Wendeauftrag durchgeführt werden soll
<i>betriebsstelle</i> (String)	Name der Betriebsstelle des nächsten Halts
<i>live_all_targets_reached</i> (Integer)	Index der Betriebsstelle, die erreicht wurde

Tabelle 4: Aufbau eines Eintrags aus dem *\$allTimes*-Array

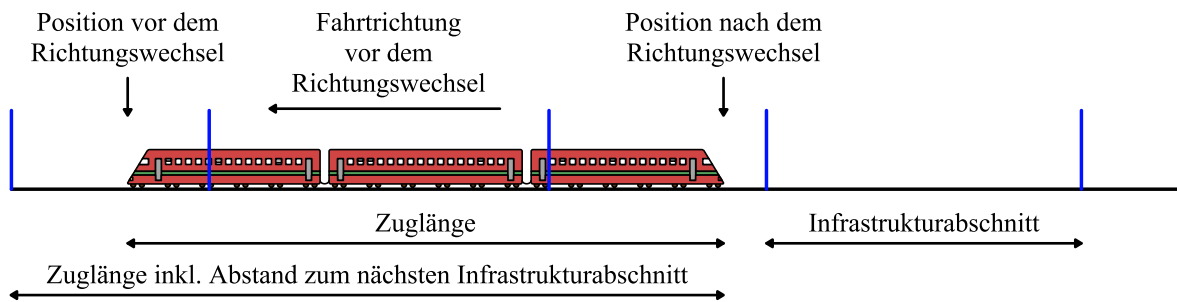


Abbildung 4: Eigene Darstellung der Positionsbestimmung bei einem Richtungswechsel

angezeigt. Im zweiten Schritt wird der aktuelle Infra-Abschnitt, die aktuelle Position innerhalb des Abschnitts und die Geschwindigkeit in dem Array *\$allUsedTrains* abgespeichert.

Sollte das Fahrzeug nach dem Ausführen der Echtzeitdaten einen Wendeauftrag bekommen und dementsprechend der Eintrag *wendet true* sein, so wird die Funktion *changeDirection()* (*functions.php*) aufgerufen. In der Funktion wird neben der Fahrtrichtungsänderung die neue Position ermittelt (die Position eines Fahrzeugs wird durch den Zugkopf beschrieben) und überprüft, ob die Fahrtrichtung geändert werden kann. Damit die Fahrtrichtungsänderung ebenfalls funktioniert, wenn das Fahrzeug nicht am Ende eines Infra-Abschnitts steht, wird für die Ermittlung der neuen Position auf die Fahrzeuglänge der Abstand bis zum Ende Infra-Abschnitts addiert (siehe Abbildung 4). Über den aktuellen und die folgenden Infra-Abschnitte (ermittelt durch die Funktion *getNaechsteAbschnitte()*\* (*functions\_ebuef.php*), des aktuellen Infra-Abschnitts und der neuen Fahrtrichtung) wird iteriert und die Summe der Längen gebildet, bis die Fahrzeuglänge (zuzüglich des Abstands bis zum Ende des Infra-Abschnitts) überschritten wird. Der Infra-Abschnitt, in dem die Fahrzeuglänge inkl. des Abstands zum ersten Mal überschritten wird, entspricht dem Infra-Abschnitt der neuen Position.

Sollte die Länge aller nächsten Abschnitte inklusive des aktuellen Abschnitts in der Summe kleiner sein, als die Zuglänge inkl. dem Abstands bis zum Ende des Infra-Abschnitts, kann die neue Position nicht ermittelt werden und dem Fahrzeug wird eine Fehlermeldung übergeben, sodass das Fahrzeug nicht weiter fahren wird. Andernfalls wird die Richtung des Fahrzeugs in der Datenbank geändert und dem Fahrzeug mit der Funktion *sendFahrzeugbefehl()*\* (*functions\_ebuef.php*) die Geschwindigkeit *-4 km/h* (entspricht einem Wendeauftrag) übergeben.

Bei einem Fahrtverlauf kann es vorkommen, dass Fahrzeuge mit Fahrplan auf der

	Fährt jetzt ohne Fahrplan	Fährt jetzt nach Fahrplan
Fuhr davor ohne Fahrplan	1. Fall	2. Fall
Fuhr davor nach Fahrplan	3. Fall	4. Fall

Tabelle 5: Verhalten eines Fahrzeugs nach dem Erreichen des Ziels

Fahrt mehrere Betriebsstellen passieren. Damit dem Eintrag *angekommen* dieser Betriebsstellen auch der Wert *true* zugewiesen werden kann, wird überprüft, ob in den Echtzeitdaten dem Eintrag *live\_all\_targets\_reached* ein Wert zugewiesen ist. Dieser Eintrag enthält – falls das Fahrzeug eine Betriebsstelle erreicht hat – den Index der Betriebsstelle und weist der Betriebsstelle unter dem Eintrag *angekommen* den Wert *true* zu.

Wenn die letzten Echtzeitdaten eines Fahrzeugs übermittelt wurden (*live\_target\_reached == true*) und das Fahrzeug dementsprechend zum Stehen gekommen ist, wird überprüft, wie sich das Fahrzeug als nächstes verhalten soll. Dafür wird zwischen vier Fällen (siehe Tabelle 5) unterschieden. Für die Überprüfung, ob sich der Fahrplan eines Fahrzeugs geändert hat, wird über die Funktion *getFahrzeugZugIds()* (*functions\_ebuef.php*) die aktuelle Zug-ID abgefragt und mit der vorherigen verglichen. In dem **1. Fall** (alte und neue Zug-ID haben beide den Wert *null*) werden dem Fahrzeug keine neue Daten übergeben und ein neuer Fahrtverlauf wird versucht zu berechnen, sobald die Fahrstraße sich verändert hat. In dem **2. und 4. Fall** wird die neue Zug-ID dem Fahrzeug übergeben, der Eintrag *operates\_on\_timetable* auf *true* gesetzt und die Funktionen *getFahrplanAndPositionForOneTrain()* (*functions.php*), *addStopsectionsForTimetable()* (*functions.php*), *calculateNextSections()* (*functions\_fahrtverlauf.php*), *checkIfFahrstrasseIsCorrect()* (*functions.php*) und *calculateFahrtverlauf()* (*functions.php*) aufgerufen. Abgesehen von der ersten Funktion, werden diese Funktionen auch beim Start des Programms ausgeführt, welcher in Kapitel 3.3 beschrieben wird. Die Funktion *getFahrplanAndPositionForOneTrain()* (*functions.php*) ähnelt der in Kapitel 3.3 beschriebenen Funktion *prepareTrainForRide()* (*functions.php*), fügt aber nur die Position und den Fahrplan hinzu, da alle anderen Daten schon eingelesen wurden. In dem **3. Fall** (die neu ermittelte Zug-ID hat den Wert *null*) wird der Eintrag *operates\_on\_timetable* auf *false* gesetzt und die Funktionen *calculateNextSections()* (*functions.php*) und *calculateFahrtverlauf()* (*functions.php*) aufgerufen.

### 3.6 Überprüfung nach einer Änderung der Fahrstraße

Für die Überprüfung, ob sich die Fahrstraße der Züge verändert hat, wird in regelmäßigen Abständen die Fahrstraße der Fahrzeuge ermittelt und mit der aktuell hinterlegten Fahrstraße verglichen. Das Intervall, in dem diese Überprüfung stattfindet kann über die Variable *\$timeCheckFahrstrasseInterval* (*fahrzeugsteuerung.php*) festgelegt werden und ist standardgemäß auf 3 Sekunden festgelegt. Bei der Ermittlung und dem Vergleich der Fahrstraße wird für jedes Fahrzeug die Funktion *compareTwo-NaechsteAbschnitte()* (*functions.php*) aufgerufen. Innerhalb dieser Funktion wird die in Kapitel 3.3 erläuterte Funktion *calculateNextSections()* (*functions.php*) aufgerufen, mit dem Unterschied, dass die ermittelten nächsten Infra-Abschnitte inkl. der Längen und zulässigen Höchstgeschwindigkeiten nicht dem Fahrzeug hinterlegt werden, sondern lokal in der Funktion gespeichert. Damit die ermittelten Daten für ein Fahrzeug berechnet werden, aber nicht dem Fahrzeug hinterlegt werden, kann der Parameter *\$writeResultToTrain* der Funktion *calculateNextSections()* (*functions.php*) (standardgemäß auf *true* gesetzt) auf *false* gesetzt werden. Sollte sich die Fahrstraße geändert haben, wird mit der Funktion *checkIfFahrstrasseIsCorrect()* (*functions.php*) überprüft, ob die Fahrstraße dem Fahrplan (falls vorhanden) entspricht und im Anschluss die Funktion *calculateFahrtverlauf()* (*functions.php*) aufgerufen.

### 3.7 Neukalibrierung der Fahrzeugposition

Für eine genau Fahrzeugsteuerung ist die aktuelle Position der Züge essenziell und muss während der Fahrt kalibriert werden, damit Ungenauigkeiten ausgeglichen werden können. Dafür werden die Daten aus der *MySQL*-Tabelle *fahrzeuge\_abschnitte* benötigt, welche durch die Abschnittsüberwachung ermittelt werden. Die Abschnittsüberwachung schreibt für jedes Fahrzeug den aktuellen Infra-Abschnitt in die Datenbank, sobald der Zugkopf den Abschnitt befährt inklusive der aktuellen Zeit (Realzeit). Für jedes Fahrzeug, welches durch die Übermittlung der Echtzeitdaten in einen neuen Infra-Abschnitt einfährt und seit der Einfahrt in den Abschnitt die Geschwindigkeit nicht verändert hat, wird die aktuelle Position neu ermittelt. Würde sich das Fahrzeug in einem Infra-Abschnitt befinden und hätte seit der Einfahrt die Geschwindigkeit angepasst, könnte mit der Fahrzeugsteuerung die Position nicht neu berechnet werden, da nicht bekannt ist, welche Strecke das Fahrzeug seit der Einfahrt zurückgelegt hat. Aus diesem Grund wird, sobald das Fahrzeug nach den Echtzeitdaten einen neuen Abschnitt befährt und aktuell nicht die Geschwindigkeit anpasst (*live\_is\_speed\_change == false*) dem Eintrag *calibrate\_section\_one* der aktuelle Infra-Abschnitt hinzugefügt und dem Eintrag *calibrate\_section\_two* wird ebenfalls der aktuelle Infra-Abschnitt

```

1 // Kalibriert die Position des Fahrzeugs neu anhand der Daten in der Tabelle
2 // 'fahrzeuge_abschnitte'
3 function getCalibratedPosition ($id, $speed) {
4     global $cacheFahrzeugeAbschnitte;
5     $DB = new DB_MySQL();
6     $positionReturn = $DB->select("SELECT '".DB_TABLE_FAHRZEUGE_ABSCHNITTE."'.'.
        ↳ infra_id','".DB_TABLE_FAHRZEUGE_ABSCHNITTE."'.'.unixtimestamp' FROM '".
        ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE."' WHERE '".
        ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE."'.'.fahrzeug_id' = $id")[0];
7     unset($DB);
8     if (in_array($id, array_keys($cacheFahrzeugeAbschnitte))) {
9         if ($positionReturn->unixtimestamp == $cacheFahrzeugeAbschnitte[$id]["
            ↳ unixtimestamp"]) {
10             return array("possible" => false);
11         }
12     }
13     $timeDiff = time() - $positionReturn->unixtimestamp;
14     $position = ($speed / 3.6) * $timeDiff;
15     return array("section" => $positionReturn->infra_id, "position" => $position)
        ↳ ;
16 }

```

Code-Beispiel 3: *getCalibratedPosition()* (*functions\_db.php*)

hinzugefügt, wenn *calibrate\_section\_one* ein Wert zugewiesen ist und dieser nicht dem aktuellen Infra-Abschnitt der Echtzeitdaten entspricht. Sobald das Fahrzeug seine Geschwindigkeit anpasst (*live\_is\_speed\_change == true*), wird beiden Einträgen der Wert *null* zugewiesen. Dadurch ist dem Eintrag *calibrate\_section\_two* nur dann ein Infra-Abschnitt zugewiesen, wenn das Fahrzeug in diesem seit der Einfahrt die Geschwindigkeit nicht verändert hat. Wenn dem Eintrag *\$useRecalibration* aus der Datei *global\_variables.php* der Wert *true* zugewiesen ist, wird in regelmäßigen Abständen überprüft, ob eine Neukalibrierung möglich ist. Das Zeitintervall, in dem die Überprüfung stattfindet ist standardmäßig auf 3 Sekunden eingestellt, kann aber mittels der Variable *\$timeCheckCalibrationInterval* (*fahrzeugsteuerung.php*) angepasst werden.

Für die Neukalibrierung wird die Funktion *getCalibratedPosition()* (*functions.php*) (Code-Beispiel 3) aufgerufen, welche als Rückgabewert die aktuelle relative Position und den aktuellen Infra-Abschnitt zurückgibt.

Sollte die ermittelte Position innerhalb des Infra-Abschnitts größer als die Länge des Infra-Abschnitts sein, welche in dem Array *\$cacheInfraLaenge* abgespeichert ist, wird die Neukalibrierung nicht durchgeführt. Der aktuelle Infra-Abschnitt wird aus der Tabelle *fahrzeuge\_abschnitte* der *MySQL*-Datenbank geladen und durch die aktuelle

Geschwindigkeit des Fahrzeugs und die Differenz der Zeit zwischen dem Einfahren in den Infra-Abschnitt und der aktuellen Zeit wird die relative Position innerhalb des Infra-Abschnitts berechnet.

$\text{relative Position} = \text{Geschwindigkeit} \cdot \text{Zeitdifferenz (aktuelle Zeit} - \text{Zeit des Einfahrens)}$

### 3.8 Ermittlung von neuen Fahrzeugen im eingleisigen Netz

Die Fahrzeugsteuerung betrachtet neben den Fahrzeugen, welche sich schon zu Beginn des Programmstarts im eingleisigen Netz befinden auch alle Fahrzeuge, die nach dem Programmstart hinzugefügt werden. Für alle Fahrzeuge, die beim Start des Programms erkannt werden, wird in dem Array *\$allTrainsOnTheTrack* die zugehörige Adresse gespeichert (*findTrainsOnTheTracks()*) (*functions.php*). Für die Überprüfung, ob Fahrzeuge entfernt wurden oder neu hinzugekommen sind, wird die Funktion *updateAllTrainsOnTheTrack()* (*functions.php*) verwendet. Diese Funktion wird – wie die Neukalibrierung in Kapitel 3.7 – alle 3 Sekunden ausgeführt. Bei dem Aufruf der Funktion werden alle Fahrzeuge geladen, denen in der *fma*-Tabelle aus der Datenbank ein Infra-Abschnitt zugeordnet ist und mit dem Array *\$allTrainsOnTheTrack* verglichen. Fahrzeugadressen, die nicht in dem Array hinterlegt sind, werden in dem Rückgabe-Array unter dem Eintrag *new* zurückgegeben und alle Fahrzeugadressen, die in dem Array enthalten sind, aber bei dem Aufruf der Funktion keinem Infra-Abschnitt zugeordnet sind, werden in dem Rückgabe-Array unter dem Eintrag *removed* zurückgegeben. Nach dem Aufruf der Funktion, werden für alle neuen Fahrzeuge die Funktion *prepareTrainForRide()* (*functions.php*), *addStopsectionsForTimetable()* (*functions.php*), *calculateNextSections()* (*functions.php*), *checkIfTrainReachedHaltepunkt()* (*functions.php*), *checkIfFahrstrasseIsCorrect()* (*functions.php*) und *calculateFahrtverlauf()* (*functions.php*) aufgerufen (siehe Kapitel 3.3 und 3.4). Alle entfernten Fahrzeuge werden aus dem Array *\$allUsedTrains* entfernt und somit nicht mehr von der Fahrzeugsteuerung beachtet.

### 3.9 Fehlerbehebung von Fahrzeugen

Wenn es bei einem Fahrzeug zu einem Konflikt kommt, der eine Steuerung des Fahrzeugs verhindert, wird dem Fahrzeug eine Fehlermeldungs-ID unter dem Eintrag *error* in dem *\$allUsedTrains* zugewiesen. Fahrzeuge, denen eine Fehlermeldung zugeordnet wurde, werden ab diesem Zeitpunkt nicht weiter von der Fahrzeugsteuerung berücksichtigt. Für das Erkennen von Fahrzeugen mit Fehlermeldungen, wird in regelmäßigen Zeitintervallen (*\$timeCheckAllTrainStatusInterval*) die Funktion *showErrors()*



```

1 // Gibt für alle Fahrzeuge die vorhanden Fehlermeldungen an.
2 function showErrors() {
3
4     global $allUsedTrains;
5     global $trainErrors;
6
7     $foundError = false;
8     echo "Hier werden für alle Züge mögliche Fehler angezeigt:\n\n";
9
10    foreach ($allUsedTrains as $trainIndex => $trainValue) {
11        if (sizeof($trainValue["error"]) != 0) {
12            $foundError = true;
13            echo "Zug ID: ", $trainValue["id"], "\n";
14            $index = 1;
15
16            foreach ($trainValue["error"] as $error) {
17                echo "\t", $index, ". Fehler:\t", $trainErrors[$error], "\n";
18                $index++;
19            }
20
21            echo "\n";
22        }
23    }
24
25    if (!$foundError) {
26        echo "Keiner der Züge hat eine Fehlermeldung.\n";
27    }
28 }

```

Code-Beispiel 4: *showErrors()* (*functions.php*)

(*functions.php*) aufgerufen, welche die Fehlermeldungen aller Fahrzeuge ausgibt (Code-Beispiel 4). Für das Beheben einer Fehlermeldung muss das Fahrzeug händisch vom Schienennetz genommen werden, gewartet werden, bis die Fahrzeugsteuerung das Entfernen registriert hat und das Fahrzeug wieder händisch auf das Schienennetz gesetzt werden.

Die möglichen Fehlermeldungen sind in dem Array *\$trainErrors* gespeichert und können um beliebig viele weitere Fehlermeldungen ergänzt werden. Für die Implementierung einer neuen Fehlermeldung muss lediglich die Fehlermeldungs-ID (Index der Fehlermeldung in dem *\$trainErrors*-Array) dem Eintrag *error* aus dem *\$allUsedTrains*-Array hinzugefügt werden, sobald der Konflikt in der Fahrzeugsteuerung auftritt. In Tabelle 6 sind alle Fehlermeldungen aufgelistet, welche aktuell in der Fahrzeugsteuerung implementiert sind.

Fehlermeldungs-ID	Beschreibung
0	Fahrtrichtung des Fahrzeugs musste geändert werden und die Positionsbestimmung war nicht möglich
1	In der Datenbank ist für das Fahrzeug keine Zuglänge angegeben
2	In der Datenbank ist für das Fahrzeug keine $v_{\max}$ angegeben
3	Das Fahrzeug musste eine Gefahrenbremsung durchführen

Tabelle 6: Übersicht der Fehlermeldungen

## 4 Berechnung des Fahrtverlaufs

Der Fahrtverlauf eines Fahrzeuges wird bei der Berechnung auf zwei verschiedenen Arten gespeichert. Einmal in so genannten *keyPoints*, welche in einem Array die Start- und Zielgeschwindigkeit (*speed\_0* und *speed\_1*), die Start- und Endposition (*position\_0* und *position\_1*) und die Start- und Endzeit (*time\_0* und *time\_1*) der einzelnen Beschleunigungen bzw. Verzögerungen abspeichern. Für die Überprüfung, ob ein Fahrzeug die zulässige Höchstgeschwindigkeit in einem Infra-Abschnitt überschreitet, und die exakte Positionsbestimmung, werden die *keyPoints* in Echtzeitdaten umgewandelt. Die Konvertierung der *keyPoints* in die Echtzeitdaten wird in Kapitel 4.3 im Detail beschrieben.

Als Grundlage für die Berechnung des Fahrtverlaufs werden zudem die Variablen *\$indexCurrentSection* und *\$indexTargetSection* benötigt, welche die Indexe der Start- und Ziel-Infra-Abschnitte in Bezug auf das Array *\$next\_sections* beschreiben und die Arrays *\$cumulativeSectionLengthStart* und *\$cumulativeSectionLengthEnd*, welche für jeden Infra-Abschnitt den Abstand zur aktuellen Position von dem Anfang und dem Ende des Infra-Abschnitts angeben.

Der Fahrtverlauf wird mit der Funktion *updateNextSpeed()* (*functions\_fahrtverlauf.php*) berechnet, welche als Parameter unter anderem die Zugdaten aus dem *\$all-UsedTrains*-Array, Start- und Endzeit der Fahrt (*\$startTime* und *\$endTime*), den Ziel-Infra-Abschnitt (*\$targetSection*) und die relative Position in dem Ziel-Infra-Abschnitt (*\$targetPosition*) übergeben bekommt.

In dem folgenden Abschnitt werden die einzelnen Schritte beschrieben, die durchlaufen werden, um den optimalen Fahrtverlauf zu berechnen. In der Darstellung 5 wird der Ablauf grob schematisch dargestellt.

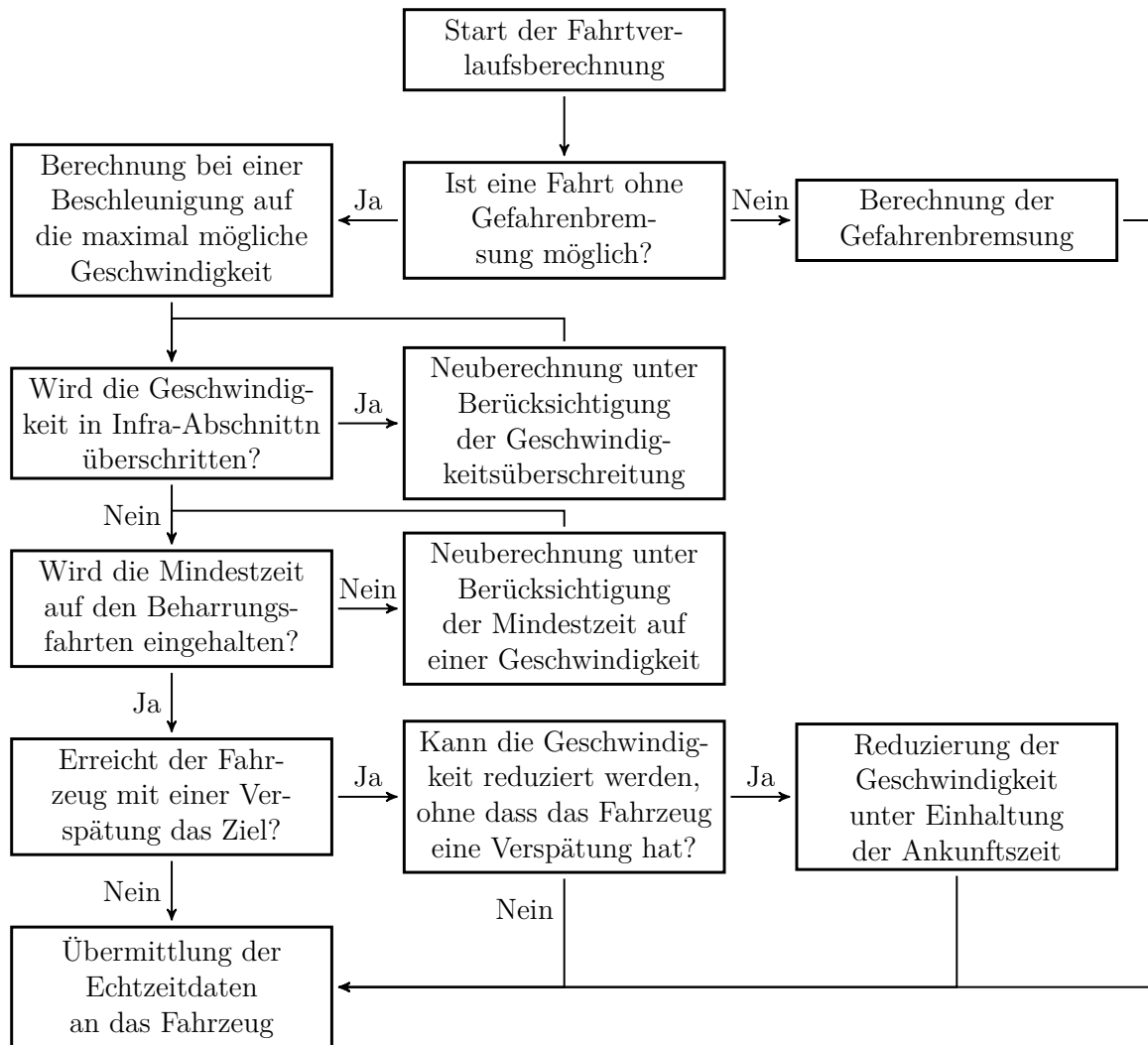


Abbildung 5: Ablaufplan der Fahrungsverlaufsrechnung

Bezeichnung	Funktion
$\$keyPoint$ (Array)	Beschreibt eine Beschleunigung bzw. Verzögerung ( $position\_0$ , $position\_1$ , $time\_0$ , $time\_1$ , $speed\_0$ , $speed\_1$ )
$\$next\_section$ (Array)	IDs aller Infra-Abschnitte
$\$next\_lengths$ (Array)	Längen aller Infra-Abschnitte
$\$next\_v\_max$ (Array)	Höchstgeschwindigkeit aller Infra-Abschnitte
$\$indexCurrentSection$ (Integer)	Index des aktuellen Infra-Abschnitts
$\$indexTargetSection$ (Integer)	Index des Ziel-Infra-Abschnitts
$\$cumulativeSectionLengthStart$ (Array)	Absolute Startposition aller Infra-Abschnitte
$\$cumulativeSectionLengthEnd$ (Array)	Absolute Endposition aller Infra-Abschnitte
$\$strainPositionChange$ (Array)	Alle absoluten Positionen des Fahrtverlaufs
$\$strainSpeedChange$ (Array)	Alle Geschwindigkeiten des Fahrtverlaufs

Tabelle 7: Beschreibung der verwendeten Variablen für die Fahrtverlaufsrechnung

#### 4.1 Ermittlung der Start- und Endposition der einzelnen Infra-Abschnitte unter Berücksichtigung der Zuglängen

Für die Berechnung eines exemplarischen Fahrtverlaufs wurden die in Tabelle 8 definierten Infra-Abschnitte verwendet. Diese Infra-Abschnitte wurden so gewählt, dass alle Funktionen und die Allgemeingültigkeit des Algorithmus gezeigt werden können und existieren in dieser Form im EBUf nicht. Als exemplarisch gewählte Zugdaten wurden die in Tabelle 9 definierten Daten verwendet.

Die zuvor ermittelten nächsten Infra-Abschnitte inklusive derer Längen und zulässigen Höchstgeschwindigkeit müssen für die Berechnung des Fahrtverlaufs angepasst werden, da ein Fahrzeug erst beschleunigen darf, wenn das komplette Fahrzeug in den Infra-Abschnitt eingefahren ist. In Darstellung 6 sind die Infra-Abschnitte dargestellt, wie sie von der Fahrzeugsteuerung ermittelt wurden. Dabei werden alle Infra-Abschnitte, die das Fahrzeug bereits durchfahren hat oder hinter dem Ziel-Infra-Abschnitt liegen nicht dargestellt. Zudem wird in dem aktuellen Infra-Abschnitt die relative Position von der Länge abgezogen und der Ziel-Infra-Abschnitt wird nur bis zur relativen Zielposition abgebildet. Dementsprechend ist der erste Infra-Abschnitt in der Darstellung 6 der Infra-Abschnitt mit der ID 1001. Dieser hat aufgrund der aktuellen relativen Position des Fahrzeugs eine Länge von 290 m. Und der letzte Infra-Abschnitt ist der Infra-Abschnitt mit der ID 1010 und einer Länge von eben-

Infra-Abschnitts-ID	Länge	zulässige Höchstgeschwindigkeit
1000	300 <i>m</i>	120 <i>km/h</i>
1001	400 <i>m</i>	120 <i>km/h</i>
1002	300 <i>m</i>	120 <i>km/h</i>
1003	400 <i>m</i>	90 <i>km/h</i>
1004	300 <i>m</i>	60 <i>km/h</i>
1005	200 <i>m</i>	60 <i>km/h</i>
1006	400 <i>m</i>	90 <i>km/h</i>
1007	500 <i>m</i>	120 <i>km/h</i>
1008	300 <i>m</i>	120 <i>km/h</i>
1009	400 <i>m</i>	100 <i>km/h</i>
1010	300 <i>m</i>	60 <i>km/h</i>
1011	300 <i>m</i>	40 <i>km/h</i>

Tabelle 8: Exemplarische Infra-Abschnitte

relative Startposition	10 <i>m</i>
relative Zielposition	290 <i>m</i>
aktueller Infra-Abschnitt	1001
Ziel-Infra-Abschnitt	1010
Startgeschwindigkeit	0 <i>km/h</i>
Zielgeschwindigkeit	0 <i>km/h</i>
Zuglänge	50 <i>m</i>
Bremsverzögerung	0,8 <i>m/s</i> <sup>2</sup>
Fahrplan vorhanden	ja
Zeit bis zur nächsten Betriebsstelle	210 <i>s</i>

Tabelle 9: Exemplarische Zugdaten

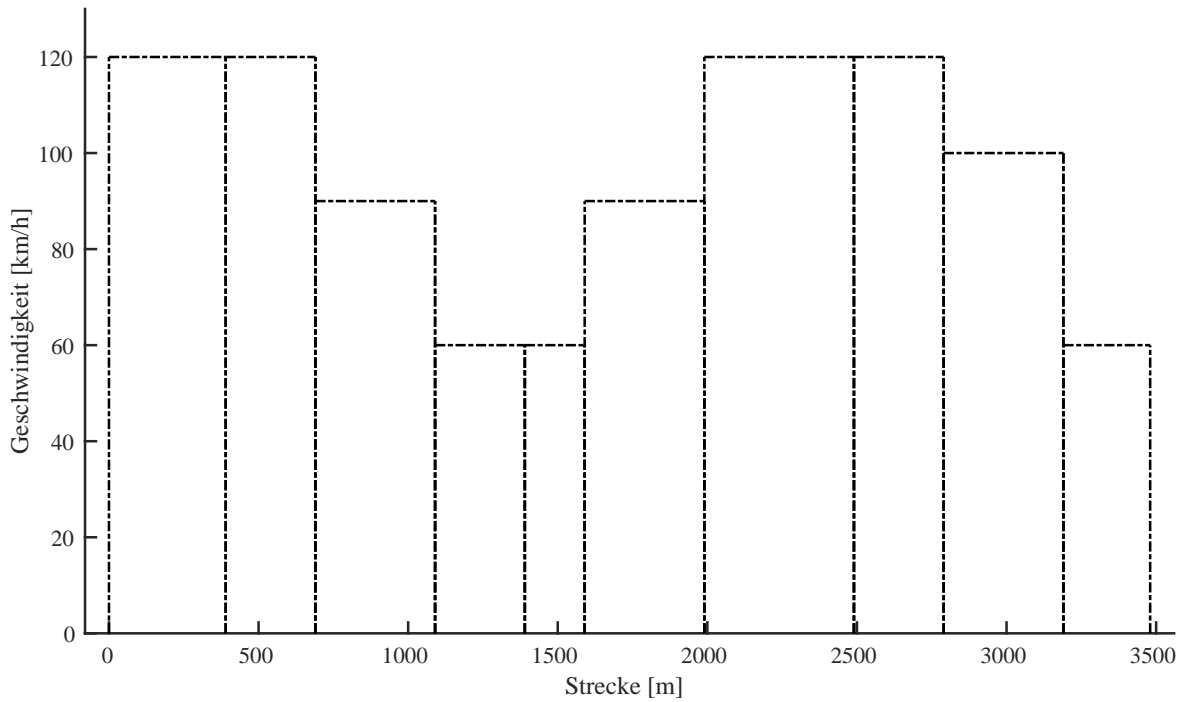


Abbildung 6: Infra-Abschnitte und die zugehörige Höchstgeschwindigkeit

falls 290 m.

Bei der Berücksichtigung der Fahrzeuglänge wird mit einer *for*-Schleife über alle Infra-Abschnitte iteriert und die Zuglänge auf die Länge des Infra-Abschnitts addiert. Von dieser neu ermittelten Endposition des Infra-Abschnitts wird überprüft, ob zwischen der vorherigen Endposition und der neu ermittelten Endposition ein Infra-Abschnitt liegt, dessen zulässige Höchstgeschwindigkeit geringer ist, als die des ursprünglichen Infra-Abschnitts. Wenn dieser Fall eintritt, wird der Infra-Abschnitt nur so weit verlängert, dass keine Höchstgeschwindigkeit der folgenden Infra-Abschnitte überschritten wird. Nach der Ermittlung der neuen Endposition, startet die *for*-Schleife mit dem Infra-Abschnitt, in dem sich die Endposition befindet. Sobald der Ziel-Infra-Abschnitt erreicht wurde, wird die Schleife abgebrochen. Die neu ermittelten Infra-Abschnitte werden in den Arrays *\$next\_lengths\_mod* und *\$next\_v\_max\_mod* abgespeichert (analog zu den Arrays *\$next\_lengths* und *\$next\_v\_max*).

Durch diesen Algorithmus kann es dazu kommen, dass sich die Anzahl der Infra-Abschnitte verändert hat, wodurch die Infra-Abschnitte nicht mehr eindeutig mit der Infrastruktur-ID bezeichnet werden können. Mittels *\$next\_lengths\_mod* und *\$next\_v\_max\_mod* werden mit der Funktion *createCumulativeSections()* (*functions\_fahrtverlauf.php*) für jeden Infra-Abschnitt die absolute Start- und Endposition in den Arrays *\$cumulativeSectionLengthStartMod* und *\$cumulativeSectionLengthEndMod* gespeichert. Diese Um-

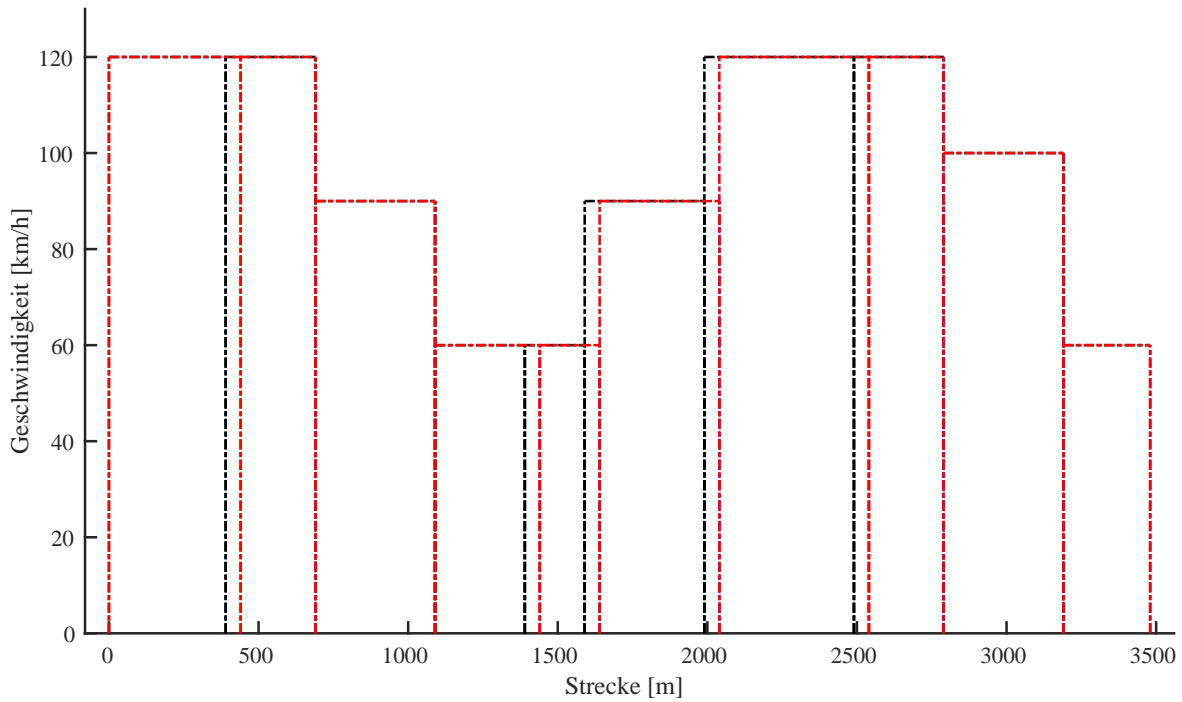


Abbildung 7: Infra-Abschnitte und die zugehörige Höchstgeschwindigkeit unter Berücksichtigung der Fahrzeuglänge

wandlung ist essentiell für die Überprüfung, in welchem Infra-Abschnitt ein Fahrzeug sich aktuell befindet. Die neu berechneten Infra-Abschnitte sind in der Darstellung 7 in rot abgebildet und beschreiben die maximale Geschwindigkeit, die ein Fahrzeug an der jeweiligen Position fahren darf.

## 4.2 Berechnung bei einer Beschleunigung auf die maximal mögliche Geschwindigkeit

Im ersten Schritt der Fahrtverlaufsberechnung wird die Distanz zwischen der aktuellen Position und der Ziel-Position mittels *\$cumulativeSectionLengthStart*, *\$cumulativeSectionLengthEnd*, *\$indexCurrentSection* und *\$indexTargetSection* berechnet. Für die Distanz und die Startgeschwindigkeit wird mit Hilfe der Funktion *getVMaxBetweenTwoPoints()* (*functions\_fahrtverlauf.php*) (Code-Beispiel 5) die maximale Geschwindigkeit ermittelt, auf die das Fahrzeug beschleunigen kann, um bis zum Ziel rechtzeitig bremsen zu können. Dabei wird in 10 *km/h*-Schritten iteriert und der maximale Wert zurückgegeben. Innerhalb der Funktion wird die Funktion *getBrakeDistance()* (*functions\_math.php*) (Code-Beispiel 9) aufgerufen, welche die benötigte Distanz für eine Beschleunigung bzw. Verzögerung berechnet und auf der Gleichung 9 aus Kapitel 7.1 basiert.

```

1 // Ermittelt die maximale Geschwindigkeit zwischen zwei Punkten
2 function getVMaxBetweenTwoPoints(float $distance, int $v_0, int $v_1) {
3
4     global $verzoeigerung;
5     global $globalFloatingPointNumbersRoundingError;
6
7     $v_max = array();
8
9     for ($i = 0; $i <= 120; $i = $i + 10) {
10         if ((getBrakeDistance($v_0, $i, $verzoeigerung) + getBrakeDistance($i, $v_1,
11             ↳ $verzoeigerung)) < ($distance +
12             ↳ $globalFloatingPointNumbersRoundingError)) {
13             array_push($v_max, $i);
14         }
15     }
16
17     if (sizeof($v_max) == 0) {
18         if ($v_0 == 0 && $v_1 == 0 && $distance > 0) {
19             echo "Der zug müsste langsamer als 10 km/h fahren, um das Ziel zu
20                 ↳ erreichen.";
21         } else {
22             //emergencyBreak($id);
23         }
24     } else {
25         if ($v_0 == $v_1 && max($v_max) < $v_0) {
26             $v_max = array($v_0);
27         }
28     }
29
30     return max($v_max);
31 }

```

Code-Beispiel 5: *getVMaxBetweenTwoPoints()* (*functions\_fahrtverlauf.php*)



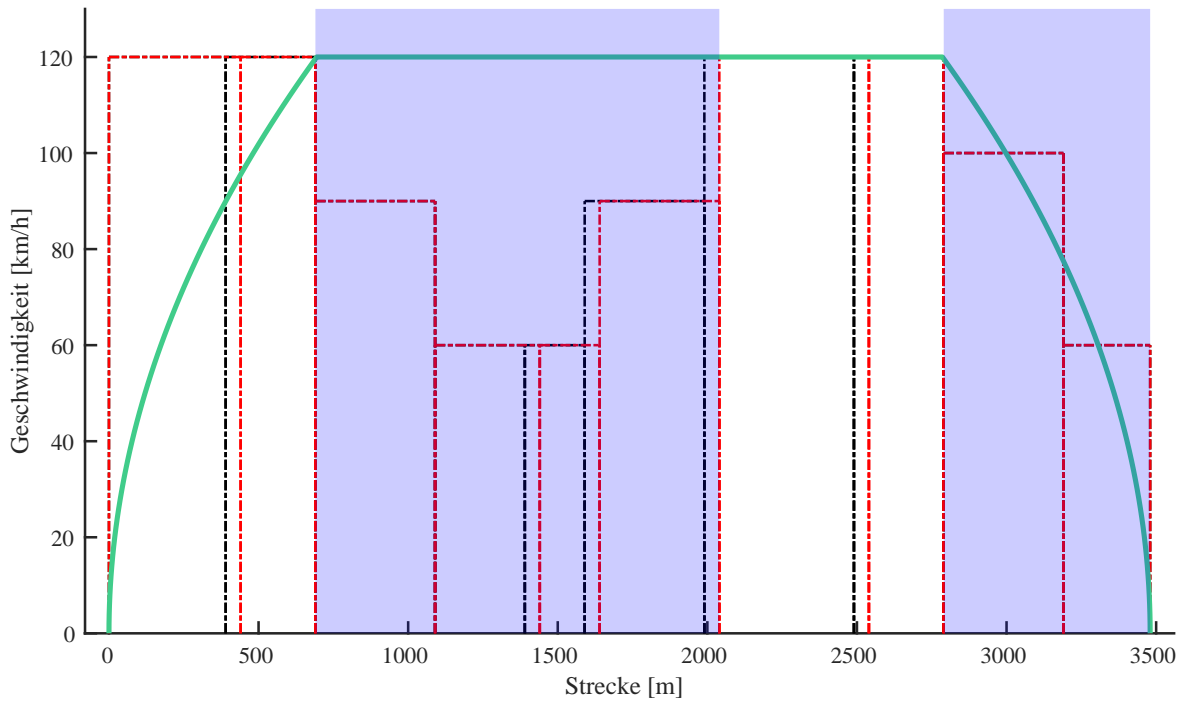


Abbildung 8: Fahrtverlaufs berechnung (1. Iterationsschritt)

Durch die gegebene Startgeschwindigkeit und die größtmögliche Geschwindigkeit wird ein erster Fahrtverlauf berechnet, wobei zwei *\$keyPoints* erzeugt werden. Mithilfe der Funktion *createTrainChanges()* (*functions\_fahrtverlauf.php*) wird aus diesen beiden *\$keyPoints* für jede Geschwindigkeitsveränderung die aktuelle absolute Position und Geschwindigkeit ermittelt. An den Positionen, an denen das Fahrzeug eine konstante Geschwindigkeit hat, wird in 1 Meter Abständen die absolute Position und die Geschwindigkeit gespeichert. Die ermittelten Daten werden in den Arrays *\$trainPositionChange* und *\$trainSpeedChange* gespeichert und sind in der Darstellung 8 abgebildet.

### 4.3 Konvertierung der *\$keyPoints* in Echtzeitdaten

Die Echtzeitdaten geben in mehreren Arrays den Fahrtverlauf bei Beschleunigungen und Verzögerungen in 2 *km/h*-Schritten und bei Beharrungsfahrten in 1 Meter Abständen an. Die Echtzeitdaten werden mit der Funktion *createTrainChanges()* (*functions\_fahrtverlauf.php*) (Code-Beispiel 6) erzeugt und geben die absolute Position des Fahrzeugs zur aktuellen Position (*\$returnTrainPositionChange*), die Geschwindigkeit (*\$returnTrainSpeedChange*), die Simulationszeit (*\$returnTrainTimeChange*), die relative Position im Infra-Abschnitt (*\$returnTrainRelativePosition*), den Infra-Abschnitt (*\$returnTrainSection*) und ob eine Geschwindigkeitsveränderung vorliegt (*\$returnIs-*

*SpeedChange*) an.

Im ersten Schritt wird mit einer *for*-Schleife durch alle *\$keyPoints* (exklusive des letzten *\$keyPoints*) iteriert (Code-Beispiel 6; Zeile 24 – 60). Für jeden *\$keyPoint* wird in einer zweiten *for*-Schleife von der Startgeschwindigkeit (*speed\_0*) bis Zielgeschwindigkeit (*speed\_1*) in 2 *km/h*-Schritten iteriert und für jede Geschwindigkeitsveränderung die benötigte Zeit und Strecke errechnet, die Geschwindigkeit gespeichert und dass es sich um eine Geschwindigkeitsanpassung handelt (Code-Beispiel 6; Zeile 32 – 37 bzw. 39 – 44).

Im Anschluss werden die Echtzeitdaten für die Beharrungsfahrten berechnet. Dafür kann über die globale Variable *\$globalTimeUpdateInterval* (*global\_variables.php*) festgelegt werden, in welchem Zeitintervall die Berechnung stattfinden soll. Der Standardwert entspricht 0,03, MUSS GEÄNDERT WERDEN AUF FEST EINEM METER!!!! (Code-Beispiel 6; Zeile 47 – 59).

```
1 // Echtzeitdatenermittlung eines Fahrtverlaufs auf Grundlage der $keyPoints.
2 // Mit dem Parameter $onlyPositionAndSpeed kann festgelegt werden, ob nur die
3 // Position und Geschwindigkeit berechnet werden soll.
4 function createTrainChanges(bool $onlyPositionAndSpeed) {
5
6     global $keyPoints;
7     global $verzoegerung;
8     global $cumulativeSectionLengthStart;
9     global $cumulativeSectionLengthEnd;
10    global $next_sections;
11    global $indexCurrentSection;
12    global $indexTargetSection;
13    global $currentPosition;
14    global $globalFloatingPointNumbersRoundingError;
15    global $globalTimeUpdateInterval;
16
17    $returnTrainSpeedChange = array();
18    $returnTrainTimeChange = array();
19    $returnTrainPositionChange = array();
20    $returnTrainRelativePosition = array();
21    $returnTrainSection = array();
22    $returnIsSpeedChange = array();
23
24    // Ermittelt für alle bis auf den letzten $keyPoint die Echtzeitdaten der
```

```

25 // Zeit, Geschwindigkeit und Position
26 for ($i = 0; $i < (sizeof($keyPoints) - 1); $i++) {
27     array_push($returnTrainTimeChange, $keyPoints[$i]["time_0"]);
28     array_push($returnTrainSpeedChange, $keyPoints[$i]["speed_0"]);
29     array_push($returnTrainPositionChange, $keyPoints[$i]["position_0"]);
30     array_push($returnIsSpeedChange, true);
31     if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
32         for ($j = ($keyPoints[$i]["speed_0"] + 2); $j <= $keyPoints[$i]["speed_1"]
33             ↪ ]; $j = $j + 2) {
34             array_push($returnTrainPositionChange, (end($returnTrainPositionChange)
35                 ↪ + getBrakeDistance(($j - 2), $j, $verzoeigerung)));
36             array_push($returnTrainSpeedChange, $j);
37             array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
38                 ↪ getBrakeTime(($j - 2), $j, $verzoeigerung)));
39             array_push($returnIsSpeedChange, true);
40         }
41     } else {
42         for ($j = ($keyPoints[$i]["speed_0"] - 2); $j >= $keyPoints[$i]["speed_1"]
43             ↪ ]; $j = $j - 2) {
44             array_push($returnTrainPositionChange, (end($returnTrainPositionChange)
45                 ↪ + getBrakeDistance(($j + 2), $j, $verzoeigerung)));
46             array_push($returnTrainSpeedChange, $j);
47             array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
48                 ↪ getBrakeTime(($j + 2), $j, $verzoeigerung)));
49             array_push($returnIsSpeedChange, true);
50         }
51     }
52 }
53
54 // Ermittelt für die Strecke zwischen zwei $keyPoints die Echtzeitdaten
55 // der Zeit, Geschwindigkeit und Position
56 $startPosition = $keyPoints[$i]["position_1"];
57 $endPosition = $keyPoints[$i + 1]["position_0"];
58 $speedToNextKeyPoint = $keyPoints[$i]["speed_1"];
59 $distanceForOneTimeInterval = $speedToNextKeyPoint / 3.6;
60
61 for ($position = $startPosition + $distanceForOneTimeInterval; $position <
62     ↪ $endPosition; $position = $position + $distanceForOneTimeInterval) {
63     array_push($returnTrainPositionChange, $position);
64     array_push($returnTrainSpeedChange, $speedToNextKeyPoint);

```

```

57     array_push($returnTrainTimeChange, end($returnTrainTimeChange) +
        ↳ $globalTimeUpdateInterval);
58     array_push($returnIsSpeedChange, false);
59 }
60 }
61 array_push($returnTrainPositionChange, $keyPoints[array_key_last($keyPoints)
        ↳ ][ "position_1" ] - getBrakeDistance($keyPoints[array_key_last(
        ↳ ↳ $keyPoints)][ "speed_0" ], $keyPoints[array_key_last($keyPoints)][ "
        ↳ ↳ speed_1" ], $verzoeigerung));
62 array_push($returnTrainSpeedChange, $keyPoints[array_key_last($keyPoints)][ "
        ↳ ↳ speed_0" ]);
63 array_push($returnTrainTimeChange, $keyPoints[array_key_last($keyPoints)][ "
        ↳ ↳ time_0" ]);
64 array_push($returnIsSpeedChange, true);
65
66 // Ermittelt für den letzten $keyPoint die Echtzeitdaten der Zeit,
67 // Geschwindigkeit und Position
68 if ($keyPoints[array_key_last($keyPoints)][ "speed_0" ] < $keyPoints[
        ↳ ↳ array_key_last($keyPoints)][ "speed_1" ]) {
69     for ($j = ($keyPoints[array_key_last($keyPoints)][ "speed_0" ] + 2); $j <=
        ↳ ↳ $keyPoints[array_key_last($keyPoints)][ "speed_1" ]; $j = $j + 2) {
70         array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
            ↳ ↳ getBrakeDistance(($j - 2), $j, $verzoeigerung)));
71         array_push($returnTrainSpeedChange, $j);
72         array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
            ↳ ↳ getBrakeTime(($j - 2), $j, $verzoeigerung))));
73         array_push($returnIsSpeedChange, true);
74     }
75 } else {
76     for ($j = ($keyPoints[array_key_last($keyPoints)][ "speed_0" ] - 2); $j >=
        ↳ ↳ $keyPoints[array_key_last($keyPoints)][ "speed_1" ]; $j = $j - 2) {
77         array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
            ↳ ↳ getBrakeDistance(($j + 2), $j, $verzoeigerung)));
78         array_push($returnTrainSpeedChange, $j);
79         array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
            ↳ ↳ getBrakeTime(($j + 2), $j, $verzoeigerung))));
80         array_push($returnIsSpeedChange, true);
81     }
82 }

```

```

83
84 if ($onlyPositionAndSpeed) {
85     return array($returnTrainPositionChange, $returnTrainSpeedChange);
86 } else {
87     // Ermittelt die relativen Positionen innerhalb der Infra-Abschnitte
88     // zu den absoluten Positionen
89     foreach ($returnTrainPositionChange as $absolutPositionKey =>
90         ↳ $absolutPositionValue) {
91         foreach ($cumulativeSectionLengthStart as $sectionStartKey =>
92             ↳ $sectionStartValue) {
93             if ($absolutPositionValue >= $sectionStartValue && $absolutPositionValue
94                 ↳ < $cumulativeSectionLengthEnd[$sectionStartKey]) {
95                 if ($sectionStartKey == $indexCurrentSection && $sectionStartKey ==
96                     ↳ $indexTargetSection) {
97                     $returnTrainRelativePosition[$absolutPositionKey] =
98                         ↳ $absolutPositionValue + $currentPosition;
99                     $returnTrainSection[$absolutPositionKey] = $next_sections[
100                         ↳ $sectionStartKey];
101                 } else if ($sectionStartKey == $indexCurrentSection) {
102                     $returnTrainRelativePosition[$absolutPositionKey] =
103                         ↳ $absolutPositionValue + $currentPosition;
104                     $returnTrainSection[$absolutPositionKey] = $next_sections[
105                         ↳ $sectionStartKey];
106                 } else if ($sectionStartKey == $indexTargetSection) {
107                     $returnTrainRelativePosition[$absolutPositionKey] =
108                         ↳ $absolutPositionValue - $sectionStartValue;
109                     $returnTrainSection[$absolutPositionKey] = $next_sections[
110                         ↳ $sectionStartKey];
111                 } else {
112                     $returnTrainRelativePosition[$absolutPositionKey] =
113                         ↳ $absolutPositionValue - $sectionStartValue;
114                     $returnTrainSection[$absolutPositionKey] = $next_sections[
115                         ↳ $sectionStartKey];
116                 }
117             }
118             break;
119         } else if ($absolutPositionKey == array_key_last(
120             ↳ $returnTrainPositionChange) && abs($absolutPositionValue -
121             ↳ floatval($cumulativeSectionLengthEnd[$sectionStartKey])) <
122             ↳ $globalFloatingPointNumbersRoundingError) {

```

```

107         $returnTrainRelativePosition[$absolutPositionKey] =
            ↳ $cumulativeSectionLengthEnd[$sectionStartKey] -
            ↳ $sectionStartValue;
108         $returnTrainSection[$absolutPositionKey] = $next_sections[
            ↳ $sectionStartKey];
109         break;
110     } else {
111         debugMessage("Einer absoluten Position konnte kein Infra-Abschnitt und
            ↳ keine relative Position in einem Infra-Abschnitt zugeordnet
            ↳ werden.");
112     }
113 }
114 }
115
116 return array($returnTrainPositionChange, $returnTrainSpeedChange,
            ↳ $returnTrainTimeChange, $returnTrainRelativePosition,
            ↳ $returnTrainSection, $returnIsSpeedChange);
117 }
118 }

```

Code-Beispiel 6: *createTrainChanges()* (*functions\_fahrtverlauf.php*)

#### 4.4 Überprüfung des Fahrtverlaufs nach Geschwindigkeitsüberschreitungen

Für die Überprüfung, ob es bei einem Fahrtverlauf zu einer Überschreitung der zulässigen Höchstgeschwindigkeit kommt, wird nach jeder Berechnung die Funktion *checkIfTrainIsToFastInCertainSections()* (*functions\_fahrtverlauf.php*) (Code-Beispiel 7) aufgerufen. In dieser Funktion wird über alle absoluten Positionen (*\$trainPositionChange*) iteriert, überprüft in welchem Infra-Abschnitt sich diese Position befindet und überprüft, ob die zugehörige Geschwindigkeit aus dem *\$trainSpeedChange*-Array die zulässige Höchstgeschwindigkeit überschreitet. Sobald in einem Infra-Abschnitt eine Geschwindigkeitsüberschreitung vorliegt, wird der zugehörige Index des Infra-Abschnitts in dem *\$failedSections*-Array gespeichert. Diese Infra-Abschnitte sind in der Darstellung 8 Lila hinterlegt.

Als Rückgabewert der Funktion wird ein Array zurückgegeben, welches abspeichert, ob und in welchen Infra-Abschnitten es zu einer Geschwindigkeitsüberschreitung gekommen ist (*failed* und *failed\_sections*).

```

1 // Überprüft, ob das Fahrzeug in Infra-Abschnitten die zulässige
2 // Höchstgeschwindigkeit überschreitet
3 function checkIfTrainIsToFastInCertainSections() {
4
5     global $trainPositionChange;
6     global $trainSpeedChange;
7     global $cumulativeSectionLengthStartMod;
8     global $next_v_max_mod;
9     global $indexTargetSectionMod;
10
11     $failedSections = array();
12
13     foreach ($trainPositionChange as $trainPositionChangeKey =>
14         ↪ $trainPositionChangeValue) {
15         foreach ($cumulativeSectionLengthStartMod as
16             ↪ $cumulativeSectionLengthStartKey =>
17             ↪ $cumulativeSectionLengthStartValue) {
18             if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
19                 if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
20                     ↪ $cumulativeSectionLengthStartKey - 1]) {
21                     array_push($failedSections, ($cumulativeSectionLengthStartKey - 1));
22                 }
23             }
24             break;
25         } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
26             if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
27                 if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
28                     ↪ $cumulativeSectionLengthStartKey]) {
29                     array_push($failedSections, $cumulativeSectionLengthStartKey);
30                 }
31             }
32             break;
33         }
34     }
35 }
36
37 if (sizeof($failedSections) == 0) {
38     return array("failed" => false);
39 } else {
40     return array("failed" => true, "failed_sections" => array_unique(
41         ↪ $failedSections));
42 }
43 }

```

Code-Beispiel 7: *checkIfTrainIsToFastInCertainSections()* (*functions\_fahrtverlauf.php*)

## 4.5 Neuberechnung unter Berücksichtigung der Geschwindigkeitsüberschreitung

In dem Fall, dass es zu einer Geschwindigkeitsüberschreitung gekommen ist, wird der Fahrtverlauf neu berechnet. Als Grundlage dafür dienen die *failed\_sections* aus der *checkIfTrainIsToFastInCertainSections()*-Funktion (*functions\_fahrtverlauf.php*) (Code-Beispiel 7). Die Funktion *recalculateKeyPoints()* (*functions\_fahrtverlauf.php*) vergleicht dabei immer zwei benachbarte *\$keyPoints* und berechnet in dem Fall einer Geschwindigkeitsüberschreitung mit der Funktion *checkBetweenTwoKeyPoints()* (*functions\_fahrtverlauf.php*) diese neu. In dem Fall, dass zwischen zwei benachbarten *\$keyPoints* die zulässige Höchstgeschwindigkeit überschritten wird, wird die absolute Start- und End-Position dieser Geschwindigkeitsüberschreitung gespeichert.

In dem folgenden Schritt wird wie in dem Abschnitt 4.2 zwischen den Start-Werten des ersten *\$keyPoints* und der ersten Geschwindigkeitsüberschreitung die maximale Geschwindigkeit berechnet und zwei neue *\$keyPoints* erzeugt. Das gleiche passiert zwischen der Position der letzten Geschwindigkeitsüberschreitung und den End-Werten des zweiten *\$keyPoints*. Dadurch wird sichergestellt, dass immer eine gerade Anzahl an *\$keyPoints* existiert und somit in jedem Iterationsschritt zwei benachbarte *\$keyPoints* verglichen werden können. Nachdem alle *\$keyPoint*-Paare überprüft wurden, werden mit Hilfe der *createTrainChanges()*-Funktion (*functions\_fahrtverlauf.php*) die Arrays *\$trainPositionChange* und *\$trainSpeedChange* erzeugt. Der neu berechnete Fahrtverlauf wird erneut der Funktion *checkIfTrainIsToFastInCertainSections()* (*functions\_fahrtverlauf.php*) (Code-Beispiel 7) übergeben. Dieser Prozess wird solange durchlaufen, bis es zu keiner Geschwindigkeitsüberschreitung mehr kommt. In den folgenden Abbildungen (Darstellung 9, 10 und 11) werden die Ergebnisse der einzelnen Iterationsschritte visuell abgebildet, wobei die grau gepunkteten Linien die Ergebnisse der vorherigen Iterationsschritte darstellen.

## 4.6 Einhaltung der Mindestzeit auf einer Beharrungsfahrt

Für eine möglichst realitätsnahe Simulation kann über die Variable *\$globalTimeOnOneSpeed* in der Datei *global\_variables.php* eine Mindestzeit festgelegt werden, die ein Fahrzeug auf einer Geschwindigkeit mindestens einhalten muss (Beharrungsfahrt). Ebenfalls kann über die Variablen *\$useMinTimeOnSpeed* und *\$errorMinTimeOnSpeed* festgelegt werden, ob die Funktion aktiviert sein soll und ob es in dem Fall, dass diese Zeit nicht eingehalten werden kann, zu einer Fehlermeldung kommen soll. Im Falle einer Fehlermeldung würde das Fahrzeug nicht losfahren bzw. eine Gefahrenbremsung



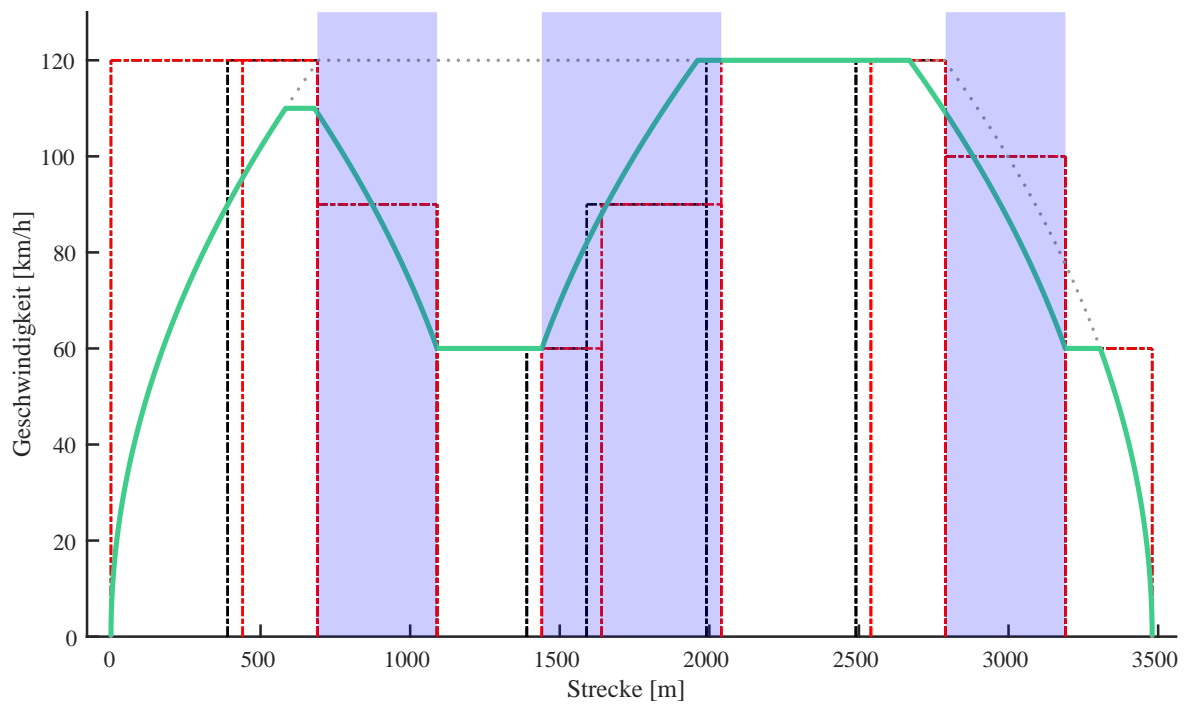


Abbildung 9: Fahrtverlaufsberechnung (2. Iterationsschritt)

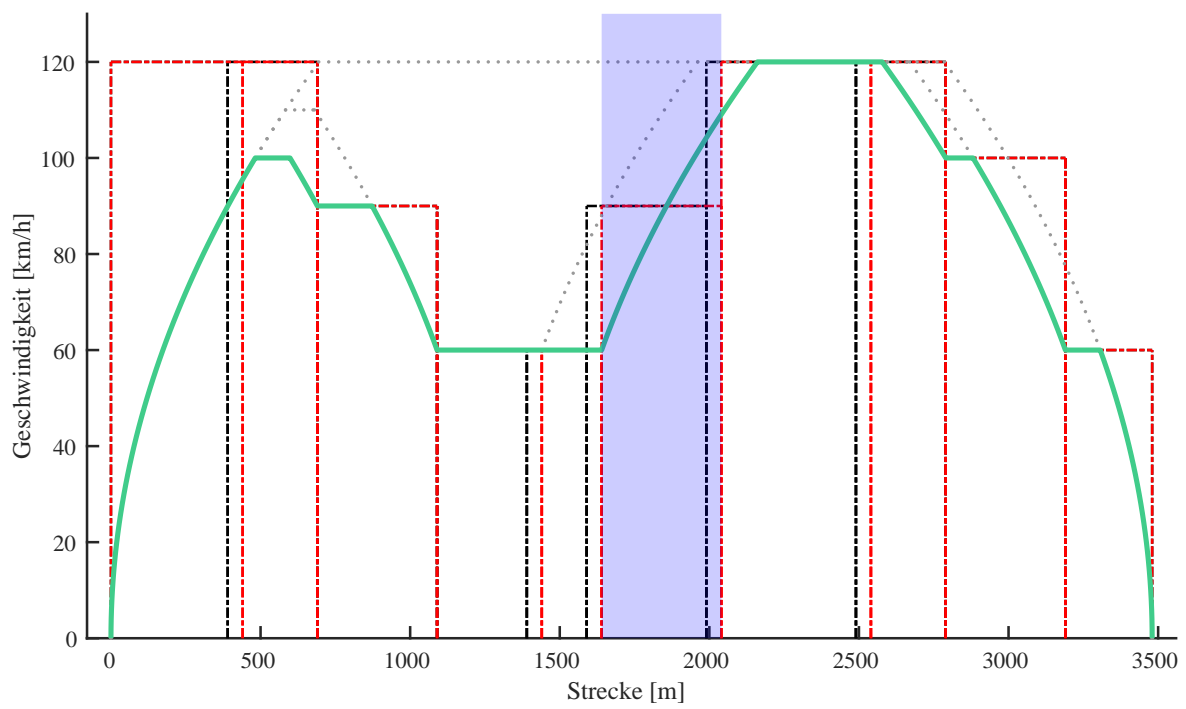


Abbildung 10: Fahrtverlaufsberechnung (3. Iterationsschritt)

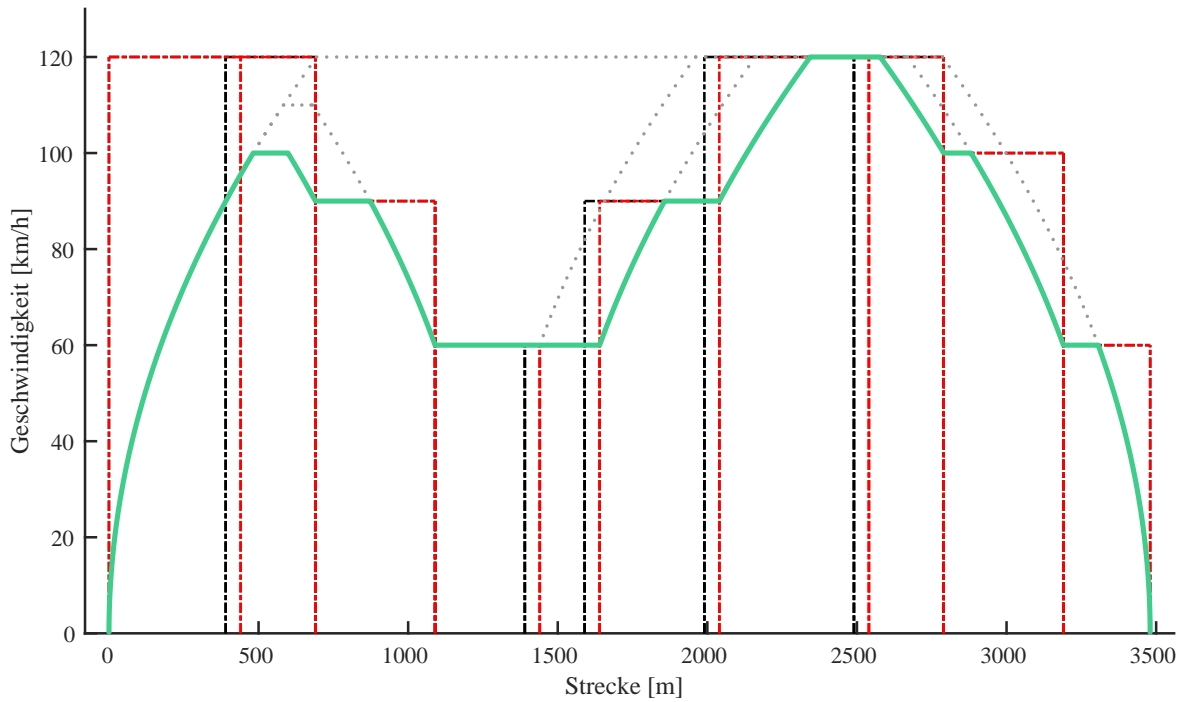


Abbildung 11: Fahrtverlaufs berechnung (4. Iterationsschritt)

einleiten, falls das Fahrzeug aktuell eine Geschwindigkeit  $v > 0 \text{ km/h}$  hat.

Wenn auf einem Abschnitt die Mindestzeit nicht eingehalten werden kann, kann eine Beschleunigung später eingeleitet werden, eine Verzögerung vorzeitig eingeleitet werden oder auf eine kleinere Geschwindigkeit beschleunigt werden. Dadurch, dass sich eine Verschiebung einer Beschleunigung bzw. Verzögerung auf die nächsten Abschnitte auswirken kann, wird der Fahrtverlauf in *Subsections* unterteilt. Eine *Subsection* beschreibt dabei den Bereich des Fahrtverlaufs, in dem das Fahrzeug zum ersten Mal beschleunigt und zum letzten Mal abbremst. In der Darstellung 12 wurde der exemplarische Fahrtverlauf somit in zwei *Subsection* unterteilt, welche Lila bzw. Gelb hinterlegt sind. Durch diese Einteilung kann verhindert werden, dass es zu Konflikten kommt. Falls die Beschleunigungen bzw. Verzögerungen soweit nach hinten bzw. nach vorne verschoben werden müssen, kann die maximale Geschwindigkeit auf dieser *Subsection* reduziert werden und die zur Verfügung stehende Strecke vergrößert werden. Wie in Darstellung 12 zu erkennen wird hierbei im ersten Schritt der Abschnitt zwischen zwei *Subsections* ausgelassen. Nach der Ermittlung der *Subsections* wird überprüft, ob auf den Abschnitten zwischen den *Subsections* die Mindestzeit eingehalten wird. Wenn das nicht der Fall ist, wird der Abschnitt automatisch der in Fahrtrichtung hinteren *Subsection* zugeordnet. Dadurch wird sichergestellt, dass das Fahrzeug, wenn es an einer Stelle des Fahrtverlaufs die Geschwindigkeit reduziert, dies möglichst spät tut.

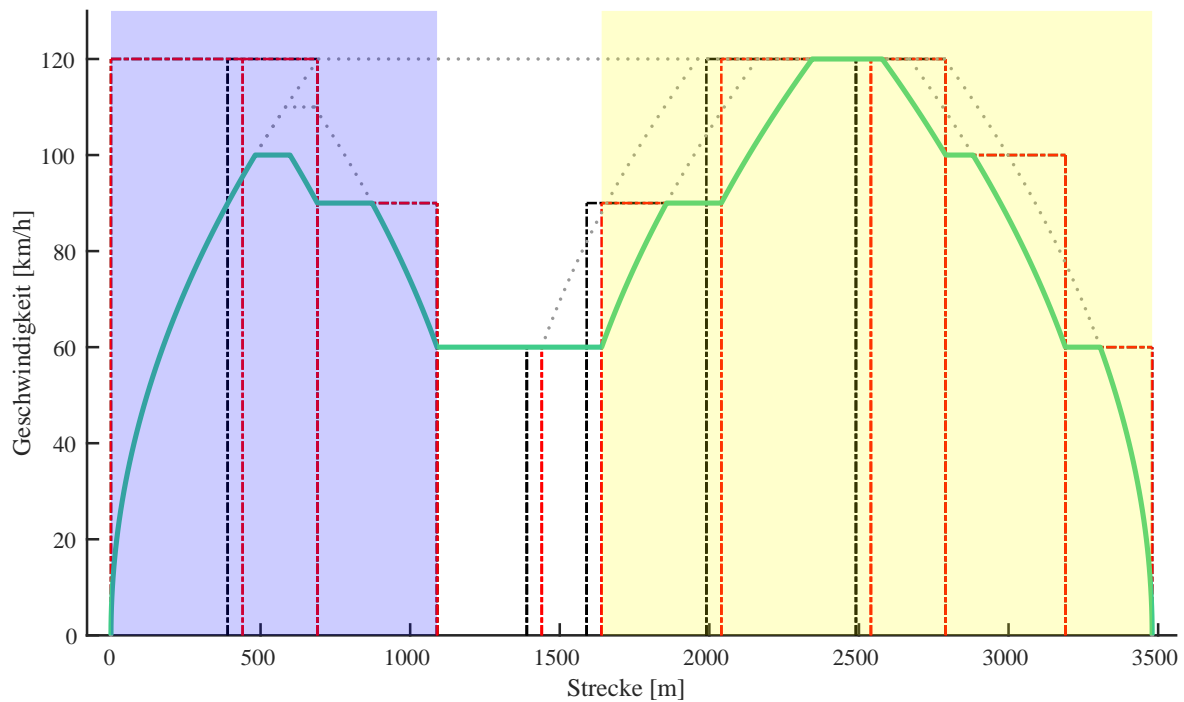


Abbildung 12: Einteilung des Fahrtverlaufs in *subsections*

Nachdem die *subsections* mittels der Funktion *createSubsections()* (*func\_tions\_fahrtverlauf.php*) erstellt wurden und mit der Funktion *array\_reverse()* in umgekehrter Reihenfolge in dem Array *subsection\_list* gesammelt wurden, wurde für jede *subsection* ein Array erzeugt, welches die Variablen aus Tabelle 10 beinhaltet und jede *subsection* eindeutig beschreibt.

Bei den *subsections*, bei denen die Mindestzeit für die Beharrungsfahrten nicht eingehalten wird (*failed == true*), wird überprüft, ob eine Verschiebung der Beschleunigungen bzw. Verzögerungen möglich ist. Bei der Verschiebung einer Beschleunigung

Index	Funktion
<i>max_index</i>	Index des <i>keyPoints</i> mit der Beschleunigung auf die maximale Geschwindigkeit in der <i>subsection</i>
<i>indexes</i> (Array)	Indexe aller beinhalteten <i>keyPoints</i>
<i>is_prev_section</i> (Boolescher Wert)	Berücksichtigung des Abschnitts vor der <i>subsection</i>
<i>is_next_section</i> (Boolescher Wert)	Berücksichtigung des Abschnitts nach der <i>subsection</i>
<i>failed</i> (Boolescher Wert)	Unterschreitung der Mindestzeit auf der <i>subsection</i>

Tabelle 10: Aufbau des *subsection*-Arrays

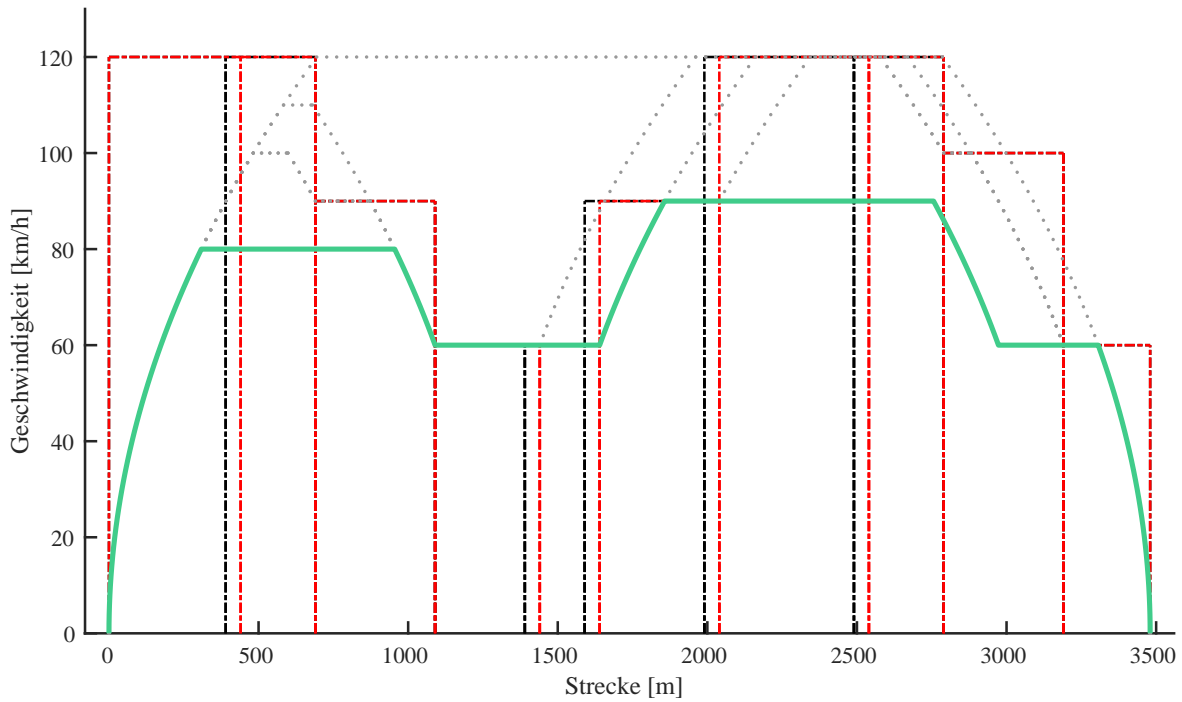


Abbildung 13: Fahrtverlauf unter Einhaltung der Mindestzeit

bzw. Verzögerung wird die Differenz zwischen der Mindestzeit einer Beharrungsfahrt ( $\$globalTimeOnOneSpeed$ ) und der Zeit der vorherigen bzw. folgenden Beharrungsfahrt berechnet und die Beschleunigung bzw. Verzögerung um diese Differenz verschoben.

Sollte bei einer Verschiebung die  $position\_1$  eines  $\$keyPoints$  hinter  $position\_0$  des folgenden  $\$keyPoints$  liegen (bei einer Beschleunigung), wird der zweite  $\$keyPoint$  gelöscht und die Zielgeschwindigkeit des zweiten  $\$keyPoints$  wird der Zielgeschwindigkeit des ersten  $\$keyPoints$  zugewiesen. Gleiches geschieht bei der Verzögerung in umgekehrter Reihenfolge.

Nach der Verschiebung wird überprüft, ob auf allen konstanten Geschwindigkeiten die Mindestzeit eingehalten wird. Wenn das der Fall ist, wird die nächste  $\$subsection$  überprüft. In dem Fall, dass durch die Verschiebung die Mindestzeit nicht eingehalten werden kann, wird die maximale Geschwindigkeit auf dieser  $\$subsection$  um  $10\text{ km/h}$  reduziert, die  $\$subsections$  neu berechnet und erneut über alle  $\$subsection$  iteriert. Die Neuberechnung ist notwendig, da durch die Reduzierung der Geschwindigkeit die  $\$subsections$  anders aufgeteilt sein können.

Wenn alle  $\$subsections$  die Mindestzeit einhalten, wird der Algorithmus beendet. In der Darstellung 13 ist der Fahrtverlauf unter Einhaltung der Mindestzeit auf einer Geschwindigkeit abgebildet. Für den Fall, dass das Fahrzeug auf einer Geschwindigkeit die Mindestzeit nicht einhält und als nächstes beschleunigen würde, kann die Beschleu-

nigung später eingeleitet werden.

#### 4.7 Berücksichtigung der Ankunftszeit bei der Berechnung des Fahrtverlaufs

Der berechnete Fahrtverlauf aus den Kapiteln 4.2, 4.4, 4.5 und 4.6 ermittelt die frühestmögliche Ankunftszeit am Ziel. In dem Fall, dass der Zug dadurch mit einer Verspätung am Ziel ankommt, wird der Fahrtverlauf an das Fahrzeug übergeben. Falls das Fahrzeug mit dem Fahrtverlauf zu früh am Ziel ankommen würde, wird überprüft, ob es möglich ist die Geschwindigkeit zu reduzieren, sodass der Zug energieeffizienter fahren kann und ohne Verspätung am Ziel ankommt.

Im ersten Schritt wird mittels der Funktion *checkIfTheSpeedCanBeDecreased()* (*functions\_fahrtverlauf.php*) überprüft, ob die Geschwindigkeit reduziert werden kann. Dabei werden alle *\$keyPoints* ermittelt, bei denen das Fahrzeug beschleunigt und die beim darauffolgenden *\$keyPoint* abbremsen. Für jeden dieser *\$keyPoints* werden die möglichen Geschwindigkeiten ermittelt, welche das Fahrzeug zwischen den beiden *\$keyPoints* fahren könnte. Für die Berechnung dieser Geschwindigkeiten wird als niedrigste Geschwindigkeit die *speed\_0* des ersten *\$keyPoints* bzw. *speed\_1* des zweiten *\$keyPoints* – je nachdem, welche niedriger ist – genommen und in 10 *km/h*-Schritten bis *speed\_1* des ersten *\$keyPoints* abgespeichert. Daraus ergibt sich für jeden *\$keyPoint* eine *range* an möglichen Geschwindigkeiten. Als Rückgabewert der Funktion wird ein Array zurückgegeben, welches die Einträge *possible* und *range* enthält und als *\$returnSpeedDecrease* abgespeichert. Der Eintrag *possible* gibt an, ob das Fahrzeug auf dem gesamten Fahrtverlauf die Geschwindigkeit reduzieren könnte und wird als Boolescher Wert (*true/false*) abgespeichert und in dem Array *range* werden alle Indexe der möglichen *\$keyPoints* inklusive der ermittelten Geschwindigkeiten abgespeichert.

In dem in Abbildung 13 dargestellten Fahrtverlauf wären für den *\$keyPoint* mit dem Index 0 (die Indexe der *\$keyPoints* entsprechen dem Zahlenbereich der  $\mathbb{N}_0$ ) die Geschwindigkeiten 60, 70 und 80 *km/h* ermittelt worden und für den *\$keyPoint* mit dem Index 2 die Geschwindigkeiten 60, 70, 80 und 90 *km/h*.

Wenn eine Reduzierung der Geschwindigkeit möglich ist, wird in einer *while*-Schleife versucht die Geschwindigkeit zu reduzieren, bis das Fahrzeug bei der nächsten Reduzierung mit einer Verspätung am Ziel ankommen würde oder eine weitere Reduzierung nicht möglich ist. Innerhalb der *while*-Schleife ermittelt die Funktion *findMaxSpeed()* (*functions\_fahrtverlauf.php*) aus dem *\$returnSpeedDecrease*-Array den *\$keyPoint* mit der höchsten Geschwindigkeit. Für den Fall, dass mehrere *\$keyPoints* die selbe Höchstgeschwindigkeit haben, wird der letzte dieser *\$keyPoints* ermittelt. Im An-

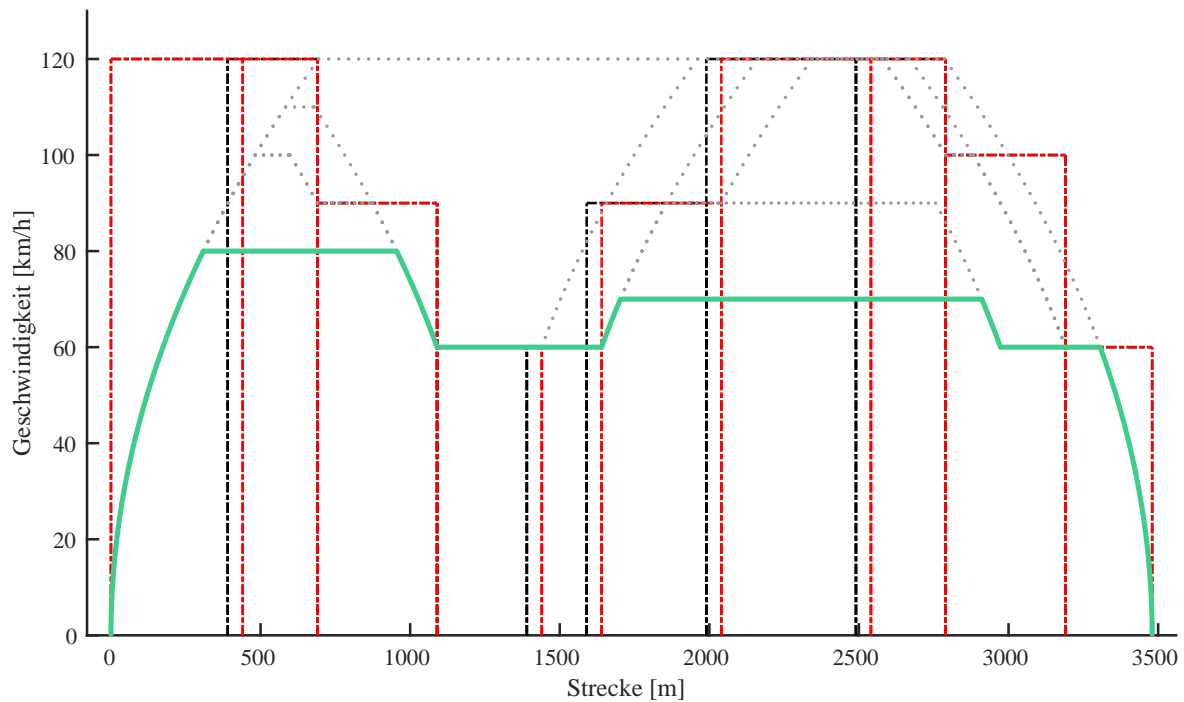


Abbildung 14: Fahrtverlauf mit reduzierter Geschwindigkeit unter Einhaltung der Ankunftszeit

schluss wird mit einer *for*-Schleife in 10 *km/h*-Schritten in absteigender Reihenfolge über die möglichen Geschwindigkeiten iteriert und überprüft, ob durch die Anpassung die Ankunftszeit eingehalten werden kann. Sobald die Ankunftszeit nicht eingehalten werden kann, werden die *\$keyPoints* aus dem vorherigen Iterationsschritt gespeichert und die *while*-Schleife wird abgebrochen. Sollte die *for*-Schleife durchlaufen, ohne dass es zu einer Überschreitung der maximal verfügbaren Zeit kommt, wird die Funktion *checkIfTheSpeedCanBeDecreased()* (*functions\_fahrtverlauf.php*) erneut aufgerufen. Das Ergebnis dieser Berechnung ist in der Abbildung 14 abgebildet.

#### 4.8 Berücksichtigung der exakten Ankunftszeit bei der Berechnung des Fahrtverlaufs

Die in Kapitel 4.6 errechnete Ankunftszeit, beschreibt die spätmöglichste Ankunftszeit am Ziel, ohne dass das Fahrzeug mit einer Verspätung am Ziel ankommt, wenn bei einer Beschleunigung auf eine geringere Zielgeschwindigkeit beschleunigt wird. Dadurch wird das Fahrzeug im Normalfall nicht exakt pünktlich das Ziel erreichen. Über die Variable *\$useSpeedFineTuning* kann festgelegt werden, ob das Fahrzeug eine exakte Ankunftszeit versuchen soll zu erreichen. Wenn diese Funktion aktiviert ist und der Eintrag *possible* aus dem Array *\$returnSpeedDecrease* *true* ist, wird für den letz-

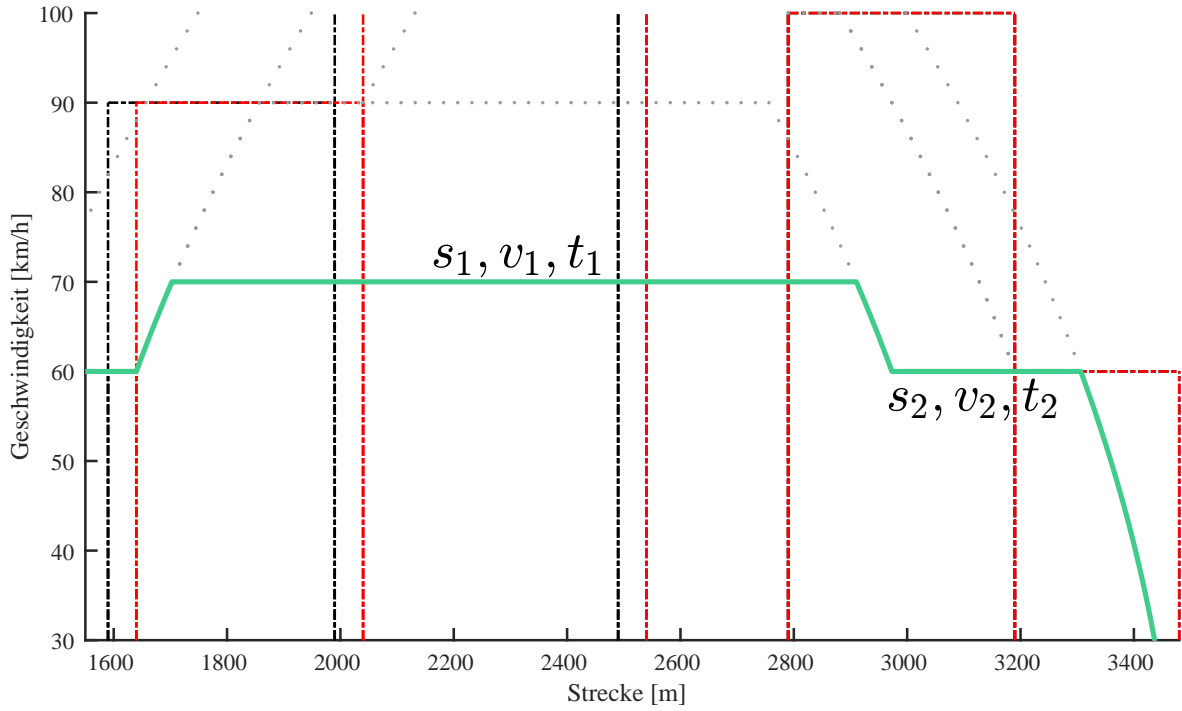


Abbildung 15: Fahrtverlauf vor der Anpassung der exakten Ankunftszeit

ten *\$keyPoint* aus dem *\$returnSpeedDecrease*-Array überprüft, ob die Verzögerung des nächsten *\$keyPoints* vorzeitiger eingeleitet werden kann. Sollte die Zielgeschwindigkeit der Verzögerung 0 *km/h* sein, wird die Verzögerung unterteilt in eine Verzögerung auf 10 *km/h* und eine von 10 *km/h* auf 0 *km/h*. Die Position der vorzeitig eingeleiteten Verzögerung wird mittels der Funktion *speedFineTuning()* (*functions\_fahrtverlauf.php*) berechnet, welche als Parameter den Betrag der Differenz zwischen aktueller Soll- und Ist-Ankunftszeit und den Index des vorherigen *\$keyPoints* übergeben bekommt und auf der Gleichung 19 aus Kapitel 7 basiert. In Abbildung 15 werden die Geschwindigkeiten ( $v_1$ ,  $v_2$ ), Strecken ( $s_1$ ,  $s_2$ ) und Zeiten ( $t_1$ ,  $t_2$ ) vor und nach der Verzögerung – welche vorzeitig eingeleitet werden soll, um eine pünktliche Ankunft am Ziel zu ermöglichen – dargestellt und in Tabelle 11 sind die exakten Werte des exemplarischen Fahrtverlaufs aufgelistet. In diesem konkreten Beispiel würde das Fahrzeug 3,31 s ( $t_\Delta$ ) zu früh an der Betriebsstelle ankommen, wodurch das Fahrzeug für die Zurücklegung der Strecken  $s_1$  und  $s_2$  insgesamt 85,42 s ( $t_{ges} = t_1 + t_2 + t_\Delta$ ) zur Verfügung hat.

Durch das Einsetzen der Werte in die Gleichung 19 aus dem Kapitel 7.2 ergibt sich für  $t_3$  ( $t_3$ ,  $t_4$ ,  $s_3$  und  $s_4$  bezeichnen die Strecken und Zeiten nach der Anpassung) ein Wert von 42,2 s. Dementsprechend muss die Verzögerung 19,85 s ( $t_1 - t_3$ ) früher eingeleitet werden.

$v_1$	70 km/h ( $\approx 19,44$ m/s)
$v_2$	60 km/h ( $\approx 16,67$ m/s)
$s_1$	1207,67 m
$s_2$	333,33 m
$s_{ges}$	1541 m
$t_1$	62,11 s
$t_2$	20 s
$t_{\Delta}$	3,31 s
$t_{ges}$	85,42 s

Tabelle 11: Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung vor der Anpassung

$s_3$	821,91 m
$s_4$	719,1 m
$t_3$	42,26 s
$t_4$	43,16 s

Tabelle 12: Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung nach der Anpassung

$$t_3 = \frac{1541 \text{ m} - 16,67 \text{ m/s} \cdot 85,42 \text{ s}}{19,44 \text{ m/s} - 16,67 \text{ m/s}}$$

$$t_3 = 42,26 \text{ s}$$

Die vorzeitige Einleitung der Verzögerung sorgt dafür, dass das Fahrzeug die nächste Betriebsstelle genau pünktlich erreicht und ist in Abbildung 16 dargestellt, wobei durch die gepunktete Linie der Fahrtverlauf vor der Anpassung zu sehen ist. Die neu berechneten Werte sind in Tabelle 12 aufgelistet. Der finale Fahrtverlauf ist in Abbildung 17 dargestellt und kann so dem Fahrzeug übergeben werden.

#### 4.9 Einleitung einer Gefahrenbremsung

Eine Gefahrenbremsung wird eingeleitet, sobald ein Fahrzeug bei einer sofortigen Verzögerung ein auf Halt stehendes Signal überfahren würde, in einem Infra-Abschnitt die zulässige Höchstgeschwindigkeit überschreiten würde oder an dem nächsten planmäßigen Halt nicht rechtzeitig zum stehen kommen würde. Bei einer Gefahrenbremsung



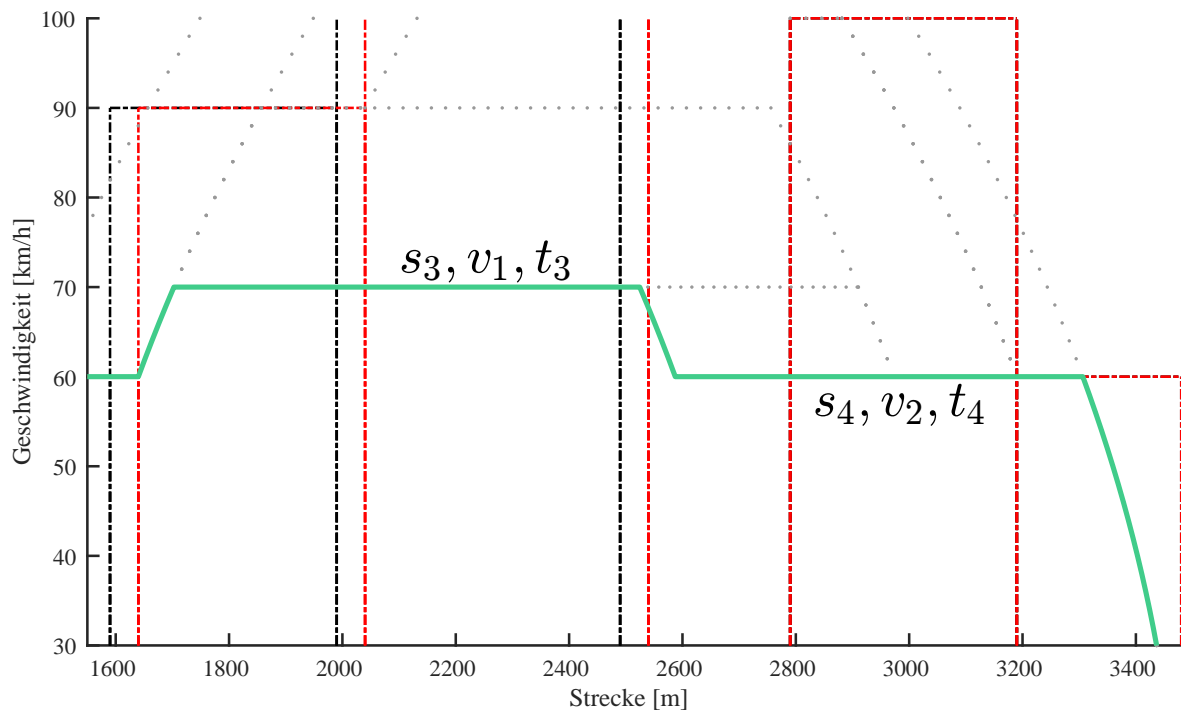


Abbildung 16: Fahrtverlauf nach der Anpassung der exakten Ankunftszeit

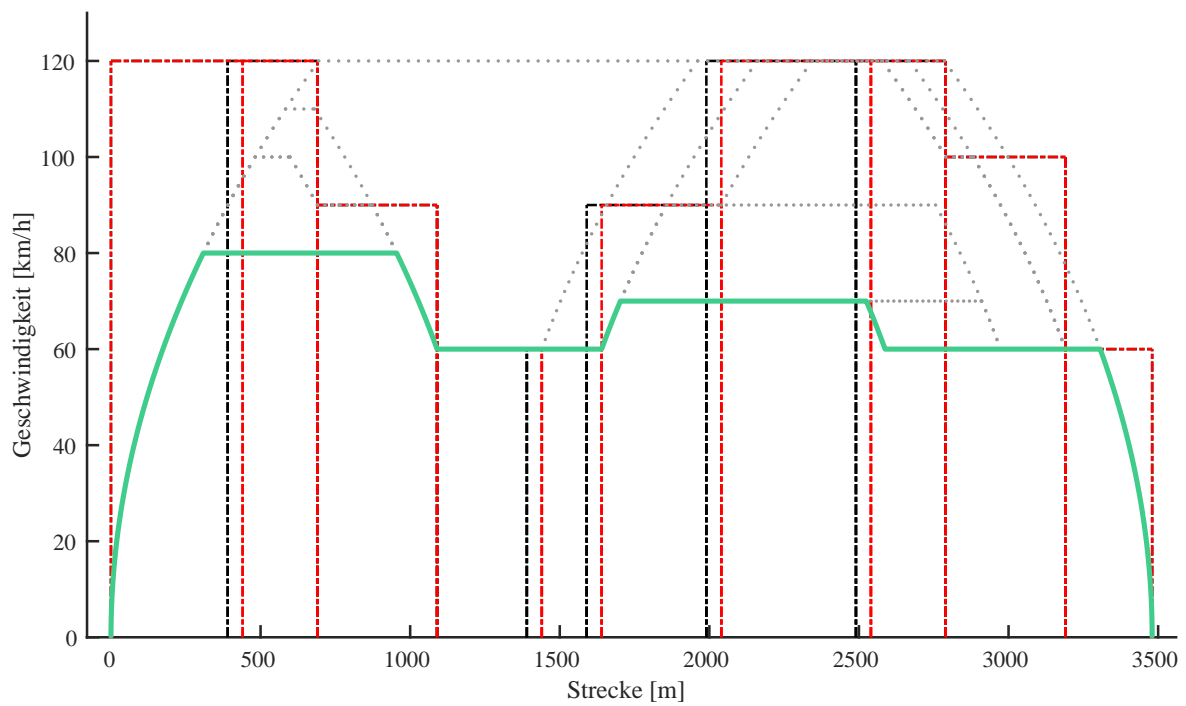


Abbildung 17: Ergebnis der Fahrtverlaufs-Ermittlung

wird mit einer Notbremsverzögerung von  $2 \text{ m/s}^2$  abgebremst. Dieser Wert kann in der Datei *global\_variables.php* über die Variable *\$globalNotverzoeigerung* angepasst werden. Für eine möglichst realitätsnahe Simulation einer Gefahrenbremsung, bei der das Risiko für Fahrzeugschäden möglichst gering ist, wurde sich dafür entschieden, dass die Fahrzeuge – wenn sie an der Gefahrenstelle eine Geschwindigkeit haben, für die gilt:  $v \geq 10 \text{ km/h}$  – nach der Geschwindigkeit von  $10 \text{ km/h}$  direkt die Geschwindigkeit von  $0 \text{ km/h}$  übermittelt bekommen. Dadurch wird bei der Berechnung einer Gefahrenbremsung zwischen drei Fällen unterschieden:

1. Fahrzeug hält mit der Notbremsverzögerung vor der Gefahrenstelle
2. Fahrzeug hat bei der Gefahrenstelle eine Geschwindigkeit von  $v < 10 \text{ km/h}$
3. Fahrzeug hat bei der Gefahrenstelle eine Geschwindigkeit von  $v \geq 10 \text{ km/h}$

Für die Überprüfung, ob das Fahrzeug mit der Notbremsverzögerung vor der Gefahrenstelle zum Stehen kommt, wird mittels der Funktion *getBrakeDistance()* (*functions.php*) der Bremsweg ( $s_{\text{Bremsweg}}$ ) berechnet und mit der Distanz zur Gefahrenstelle ( $s_{\text{Gefahrenstelle}}$ ) verglichen. Sollte für den Bremsweg gelten:  $s_{\text{Bremsweg}} \leq s_{\text{Gefahrenstelle}}$ , wird das Fahrzeug die Gefahrenbremsung einleiten und in  $2 \text{ km/h}$ -Schritten auf  $0 \text{ km/h}$  abbremsen. In dem Fall, dass der Bremsweg länger als die Strecke bis zur Gefahrenstelle ist, wird überprüft, welche Geschwindigkeit das Fahrzeug an der Gefahrenstelle hat. Für diese Berechnung wird die Gleichung 11 aus dem Kapitel 7.1 verwendet.

Sollte das Fahrzeug an der Gefahrenstelle eine Geschwindigkeit von  $v \geq 10 \text{ km/h}$  haben, bremst das Fahrzeug in  $2 \text{ km/h}$ -Schritten auf  $10 \text{ km/h}$  ab und bekommt nach der Übermittlung der  $10 \text{ km/h}$  direkt  $0 \text{ km/h}$  übergeben. In dem Fall, dass das Fahrzeug an der Gefahrenstelle langsamer als  $10 \text{ km/h}$  ist, bremst das Fahrzeug wie im 1. Fall in  $2 \text{ km/h}$ -Schritten auf  $0 \text{ km/h}$  ab. Bei einer Gefahrenbremsung bekommt das jeweilige Fahrzeug eine Fehlermeldung übermittelt und wird nicht weiterfahren, da durch die Gefahrenbremsung keine genaue Positionsbestimmung vorgenommen werden kann. Damit das Fahrzeug wieder seinen Fahrbetrieb aufnehmen kann, muss das Fahrzeug händisch von der Anlage genommen werden, gewartet werden, bis die Fahrzeugsteuerung das Entfernen registriert hat und wieder neu positioniert werden.

<i>\$keyPoint</i> -Index	0	1	2
<i>\$speed_0</i>	0 <i>km/h</i>	30 <i>km/h</i>	10 <i>km/h</i>
<i>\$speed_1</i>	30 <i>km/h</i>	10 <i>km/h</i>	0 <i>km/h</i>
<i>\$position_0</i>	0 <i>m</i>	528.83 <i>m</i>	667.18 <i>m</i>
<i>\$position_1</i>	43.40 <i>m</i>	567.41 <i>m</i>	672 <i>m</i>
<i>\$time_0</i> (Unix-Timestamp)	1631088005	1631088073,67	1631088116,53
<i>\$time_1</i> (Unix-Timestamp)	1631088015,41	1631088080,61	1631088120
<i>\$time_0</i> (hh:mm:ss)	10:00:05	10:01:14	10:01:57
<i>\$time_1</i> (hh:mm:ss)	10:00:15	10:01:21	10:02:00

Tabelle 13: *\$keyPoints* am Beispiel von der Fahrt von XAB nach XZO

## 5 Beispielrechnung eines Fahrtverlaufs im EBUf

Die in Kapitel 4 beschriebene Berechnung des Fahrtverlaufs wird in diesem Kapitel an einer Beispielfahrt von Ausblick (XAB) nach Zoo (XZO) exemplarisch gezeigt. Dafür wurde dem Zug ein Fahrplan zugewiesen, nach dem der Zug nach Simulationszeit um 10:00:05 in Ausblick losfahren soll und um 10:02:00 in dem Bahnhof Zoo ankommen soll. Zu Beginn steht der Zug im Infra-Abschnitt 1189, hat die Fahrtrichtung 1 und die Fahrstraße ist so eingestellt, dass das Fahrzeug bis zum Ausfahrtsignal im Bahnhof Zoo fahren kann und dort im Infra-Abschnitt 1178 zum Stehen kommen kann. Somit beträgt die Strecke bis zum nächsten Halt 672 *m* und das Fahrzeug hat 115 *s* zur Verfügung. Die Bremsverzögerung des Fahrzeugs beträgt 0,8 *m/s*<sup>2</sup>.

Für die Fahrt wurde eine Mindestzeit von 20 *s* für Beharrungsfahrten (*\$globalTimeOnOneSpeed* = 20) festgelegt, den Optionen *\$useSpeedFineTuning*, *\$useMinTimeOnSpeed* und *\$slowDownIfTooEarly* wurde der Wert *true* zugewiesen und der Option *\$errorMinTimeOnSpeed* der Wert *false*.

In der Tabelle 13 sind die berechneten *\$keyPoints* aufgelistet, welche durch die Berechnung des Fahrtverlaufs ermittelt wurden, und in der Darstellung 18 ist der Fahrtverlauf visuell dargestellt. Bei der Berechnung des Fahrtverlaufs wurde laut der Fahrzeugsteuerung die Ankunftszeit exakt eingehalten. Die Zeit-Werte der *\$keyPoints* geben bei der Berechnung die Simulationszeit im Unix-Timestamp-Format an und sind deswegen ebenfalls im Format *hh:mm:ss* angegeben. Durch die *\$keyPoints* und die Darstellung des Fahrtverlaufs (Abbildung 18) lässt sich der Fahrtverlauf in 5 Abschnitte einteilen. Die Start- und Zielgeschwindigkeit, die Strecke und die Zeit der einzelnen Abschnitte sind in der Tabelle 14 aufgelistet und werden mittels der Formeln aus Ka-

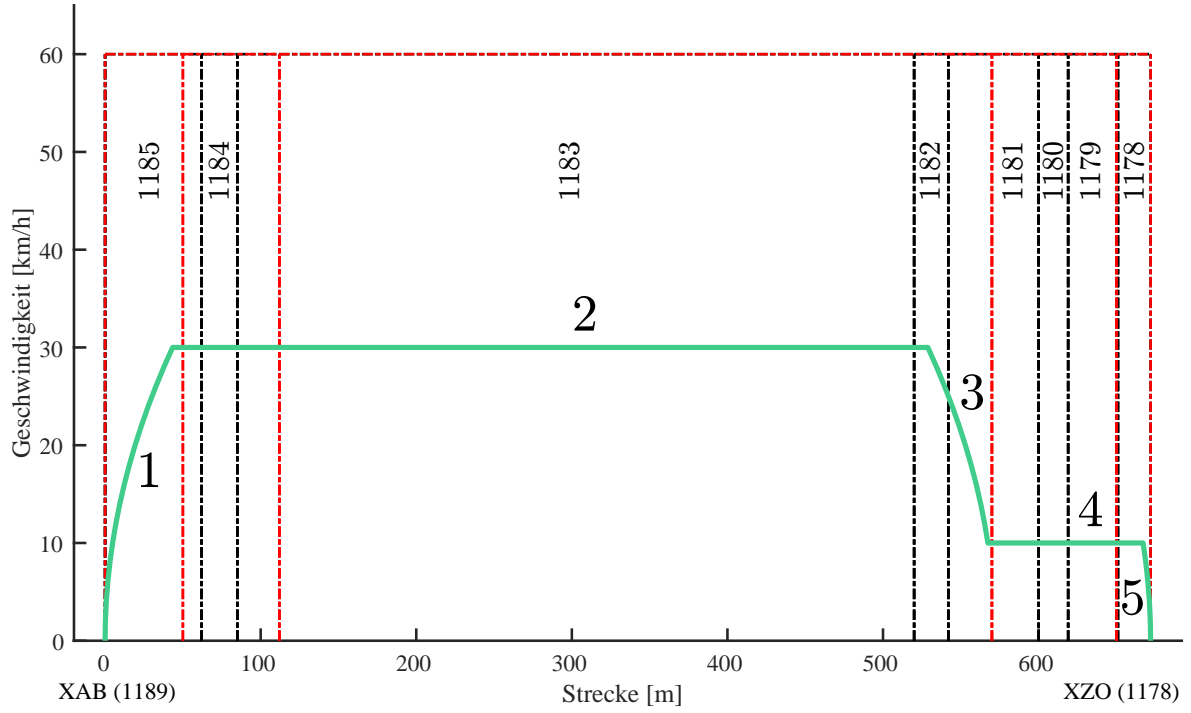


Abbildung 18: Fahrtverlauf für eine Beispielrechnung

pitel 7 überprüft. Bei der Überprüfung werden die Start- und Zielgeschwindigkeiten als Grundlage genommen und untersucht, ob unter Einhaltung der gegebenen Zeit die selben Werte rauskommen. Damit der berechnet Fahrtverlauf den Vorgaben entspricht, muss gelten:

$$t_{ges} = t_1 + t_2 + t_3 + t_4 + t_5 = 115 \text{ s}$$

$$s_{ges} = s_1 + s_2 + s_3 + s_4 + s_5 = 672 \text{ m}$$

Für die Berechnung werden die Strecken und Zeiten in gleichförmige und gleichmäßig beschleunigte Bewegungen unterteilt:

$$t_{ges} = t_{\text{gleichförmigeBewegungen}} + t_{\text{gleichmäßigBeschleunigteBewegungen}}$$

$$s_{ges} = s_{\text{gleichförmigeBewegungen}} + s_{\text{gleichmäßigBeschleunigteBewegungen}}$$

$$t_{\text{gleichförmigeBewegungen}} = t_2 + t_4$$

$$s_{\text{gleichförmigeBewegungen}} = s_2 + s_4$$

$$t_{\text{gleichmäßigBeschleunigteBewegungen}} = t_1 + t_3 + t_5$$

$$s_{\text{gleichmäßigBeschleunigteBewegungen}} = s_1 + s_3 + s_5$$

Abschnitt	Beschleunigung/Verzögerung	$v_0$	$v_1$	Strecke	Zeit
1	ja (Beschleunigung)	0 km/h	30 km/h	43,40 m	10,42 s
2	nein	30 km/h	30 km/h	485,43 m	58,25 s
3	ja (Verzögerung)	30 km/h	10 km/h	38,58 m	6,94 s
4	nein	10 km/h	10 km/h	99,76 m	35,92 s
5	ja (Verzögerung)	10 km/h	0 km/h	4,82 m	3,47 s
$\Sigma$	—	—	—	672 m <sup>1</sup>	115 s

<sup>1</sup> Die Werte in der Strecken-Spalte sind auf zwei Nachkommastellen gerundet und würden durch das Aufsummieren der Strecken von Abschnitt 1 bis 5 eine Gesamtstrecke von 671,99 m ergeben. Die angegebenen 672 m entsprechen der Summe der Abschnitte 1 bis 5, ohne dass die einzelnen Strecken gerundet werden.

Tabelle 14: Fahrtverlauf am Beispiel von der Fahrt von XAB nach XZO

Für die gleichmäßig beschleunigten Bewegungen gilt nach den Gleichungen 9 und 10:

$$t_1 = \left| \frac{\frac{30 \text{ km/h}}{3,6} - \frac{0 \text{ km/h}}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{125}{12} \text{ s} \approx 10,42 \text{ s}$$

$$t_3 = \left| \frac{\frac{10 \text{ km/h}}{3,6} - \frac{30 \text{ km/h}}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{125}{18} \text{ s} \approx 6,94 \text{ s}$$

$$t_5 = \left| \frac{\frac{0 \text{ km/h}}{3,6} - \frac{10 \text{ km/h}}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{125}{36} \text{ s} \approx 3,47 \text{ s}$$

$$s_1 = \frac{1}{2} \cdot \left| \frac{\frac{30 \text{ km/h}^2}{3,6} - \frac{0 \text{ km/h}^2}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{3125}{72} \text{ m} \approx 43,40 \text{ m}$$

$$s_3 = \frac{1}{2} \cdot \left| \frac{\frac{10 \text{ km/h}^2}{3,6} - \frac{30 \text{ km/h}^2}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{3125}{81} \text{ m} \approx 38,58 \text{ m}$$

$$s_5 = \frac{1}{2} \cdot \left| \frac{\frac{0 \text{ km/h}^2}{3,6} - \frac{10 \text{ km/h}^2}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{3125}{648} \text{ m} \approx 4,82 \text{ m}$$

Dadurch ergibt sich für die Beschleunigungen und Verzögerungen insgesamt eine Strecke von:

$$t_{\text{gleichmäßigBeschleunigteBewegungen}} = \frac{125}{6} \text{ s}$$

$$s_{\text{gleichmäßigBeschleunigteBewegungen}} = \frac{3125}{36} \text{ m}$$

Und für die gleichförmigen Bewegungen gilt dementsprechend:

$$t_{\text{gleichförmigeBewegungen}} = \frac{565}{6} \text{ s}$$

$$s_{\text{gleichförmigeBewegungen}} = \frac{21067}{36} \text{ m}$$

Für die Berechnung der Strecke und Zeit von der Beharrungsfahrt auf 30 km/h gilt nach der Gleichung 19:

$$t_2 = \frac{s_{\text{gleichförmigeBewegungen}} - \frac{10 \text{ km/h}}{3.6} \cdot t_{\text{gleichförmigeBewegungen}}}{\frac{30 \text{ km/h}}{3.6} - \frac{10 \text{ km/h}}{3.6}}$$

$$t_2 = \frac{\frac{21067}{36} \text{ m} - \frac{10 \text{ km/h}}{3.6} \cdot \frac{565}{6} \text{ s}}{\frac{30 \text{ km/h}}{3.6} - \frac{10 \text{ km/h}}{3.6}}$$

$$t_2 = \frac{34951}{600} \text{ s} \approx 58,25 \text{ s}$$

Daraus folgt nach der Gleichung 15 für die Abschnitte 2 und 4:

$$t_4 = \frac{7183}{200} \text{ s} \approx 35,91 \text{ s}$$

$$s_2 = \frac{34951}{600} \text{ s} \cdot \frac{30 \text{ km/h}}{3,6}$$

$$s_2 = \frac{34951}{72} \text{ m} \approx 485,43 \text{ m}$$

$$s_4 = \frac{7183}{200} \text{ s} \cdot \frac{10 \text{ km/h}}{3,6}$$

$$s_4 = \frac{7183}{72} \text{ m} \approx 99,76 \text{ m}$$

In Summe ergibt das:

$$t_{\text{ges}} = \frac{125}{12} \text{ s} + \frac{34951}{600} \text{ s} + \frac{125}{18} \text{ s} + \frac{7183}{200} \text{ s} + \frac{125}{36} \text{ s} = 115 \text{ s}$$

$$s_{\text{ges}} = \frac{3125}{72} \text{ m} + \frac{34951}{72} \text{ m} + \frac{3125}{81} \text{ m} + \frac{7183}{72} \text{ m} + \frac{3125}{648} \text{ m} = 672 \text{ m}$$

Wie an den errechneten Werten zu erkennen ist, wurde die Mindestzeit von 20 s auf einer konstanten Geschwindigkeit ( $t_2$  und  $t_4$ ) eingehalten und die Werte stimmen mit den Werten aus der Tabelle 14 überein.

## 6 Visualisierung der Fahrtverläufe

Für die Visualisierung der Fahrtverläufe wurde ein MATLAB-Skript geschrieben, welches aus den Arrays *\$cumulativeSectionLengthStart*, *\$cumulativeSectionLengthEnd*, *\$cumulativeSectionLengthStartMod*, *\$cumulativeSectionLengthEndMod*, *\$trainSpeedChange* und *\$trainPositionChange* eines Fahrtverlaufs den kompletten Fahrtverlauf darstellt. Dieses Skript wurde auch verwendet, um die einzelnen Schritte bei der Kalkulation des Fahrtverlaufs in dieser Arbeit darzustellen (wie z. B. in Abbildung 17).

Damit die Daten aus der Berechnung des Fahrtverlaufs von MATLAB eingelesen werden können, wurde die Funktion *safeTrainChangeToJSONFile()* (*functions\_fahrtverlauf.php*) (Code-Beispiel 8) geschrieben, welche die Daten aus den Arrays als JSON-Datei speichert. Für eine bessere Verdeutlichung des Prozesses bei der Ermittlung des Fahrtverlaufs, werden neben dem Ergebnis auch alle vorherigen Iterationsschritte abgebildet.

Das MATLAB-Skript befindet sich im Anhang dieser Arbeit (siehe A.8) und auf weitere Details bezüglich der Funktionsweise wird im Rahmen dieser Arbeit nicht weiter eingegangen.

```
1 // Wandelt die Daten der Infra-Abschnitte und der Iterationsschritte der
2 // Fahrtverlaufs Berechnung in JSON-Dateien um, damit die Fahrtverläufe
3 // visuell dargestellt werden können.
4 function safeTrainChangeToJSONFile(int $indexCurrentSection, int
    ↳ $indexTargetSection, int $indexCurrentSectionMod, int
    ↳ $indexTargetSectionMod, array $speedOverPositionAllIterations) {
5
6     global $trainPositionChange;
7     global $trainSpeedChange;
8     global $next_v_max;
9     global $cumulativeSectionLengthEnd;
10    global $next_v_max_mod;
11    global $cumulativeSectionLengthEndMod;
12    $speedOverPosition = array_map('toArr', $trainPositionChange,
    ↳ $trainSpeedChange);
13    $speedOverPosition = json_encode($speedOverPosition);
14    $fp = fopen('../json/speedOverPosition.json', 'w');
15    fwrite($fp, $speedOverPosition);
16    fclose($fp);
17    $v_maxFromUsedSections = array();
18
```

```

19 for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
20     array_push($v_maxFromUsedSections, $next_v_max[$i]);
21 }
22
23 $VMaxOverCumulativeSections = array_map('toArr', $cumulativeSectionLengthEnd,
    ↪ $v_maxFromUsedSections);
24 $VMaxOverPositionsJSoN = json_encode($VMaxOverCumulativeSections);
25 $fp = fopen('../json/VMaxOverCumulativeSections.json', 'w');
26 fwrite($fp, $VMaxOverPositionsJSoN);
27 fclose($fp);
28
29 $v_maxFromUsedSections = array();
30
31 for ($i = $indexCurrentSectionMod; $i <= $indexTargetSectionMod; $i++) {
32     array_push($v_maxFromUsedSections, $next_v_max_mod[$i]);
33 }
34
35 $VMaxOverCumulativeSectionsMod = array_map('toArr',
    ↪ $cumulativeSectionLengthEndMod, $v_maxFromUsedSections);
36 $VMaxOverPositionsJSoN = json_encode($VMaxOverCumulativeSectionsMod);
37 $fp = fopen('../json/VMaxOverCumulativeSectionsMod.json', 'w');
38 fwrite($fp, $VMaxOverPositionsJSoN);
39 fclose($fp);
40
41 $jsonReturn = array();
42
43 for ($i = 0; $i < sizeof($speedOverPositionAllIterations); $i++) {
44     $iteration = array_map('toArr', $speedOverPositionAllIterations[$i][0],
    ↪ $speedOverPositionAllIterations[$i][1]);
45     array_push($jsonReturn, $iteration);
46 }
47
48 $speedOverPosition = json_encode($jsonReturn);
49 $fp = fopen('../json/speedOverPosition_prevIterations.json', 'w');
50 fwrite($fp, $speedOverPosition);
51 fclose($fp);
52 }

```

Code-Beispiel 8: *safeTrainChangeToJSONFile()* (*functions\_fahrtverlauf.php*)



## 7 Formeln

Für die im folgenden Kapitel verwendeten Einheiten gilt:

$$\begin{aligned}a &= \text{Bremsverzögerung } [m/s^2] \\v &= \text{Geschwindigkeit } [m/s] \\s &= \text{Strecke } [m] \\t &= \text{Zeit } [s]\end{aligned}$$

### 7.1 Formeln für gleichmäßig beschleunigte Bewegungen

Bei einer gleichmäßig beschleunigten Bewegung gilt:<sup>9</sup>

$$a(t) = a \tag{1}$$

Für die Bestimmung der Geschwindigkeit in Abhängigkeit der Zeit, muss die Beschleunigung  $a(t)$  nach der Zeit  $t$  integriert werden.<sup>10</sup>

$$v(t) = \int a(t) dt \tag{2}$$

Daraus ergibt sich folgende Gleichung für die Geschwindigkeit in Abhängigkeit der Zeit. Die bei der Integration entstehende Integrationskonstante  $v_0$  gibt dabei die Startgeschwindigkeit an.

$$v(t) = a \cdot t + v_0 \tag{3}$$

Für die Bestimmung der benötigten Zeit muss die Geschwindigkeit erneut integriert werden.<sup>11</sup> Die dabei entstehende Integrationskonstante  $s_0$  gibt die bereits zurückgelegte Strecke an.

$$s(t) = \int v(t) dt \tag{4}$$

$$s(t) = \frac{1}{2} \cdot a \cdot t^2 + v_0 \cdot t + s_0 \tag{5}$$

Bei der Verwendung dieser Gleichung werden die Integrationskonstanten  $v_0$  und  $s_0$  gleich 0 gesetzt, damit die Gleichungen allgemein gültig sind. Für die Berechnung des Beschleunigungs- und Abbremsverhalten der Fahrzeuge ist es notwendig zu wissen,

---

<sup>9</sup> Richard & Sander (2011, S. 22)

<sup>10</sup> Richard & Sander (2011, S. 20)

<sup>11</sup> Richard & Sander (2011, S. 20)

welche Strecke ein Fahrzeug zurücklegen muss, um von einer Startgeschwindigkeit  $v_0$  auf eine Zielgeschwindigkeit  $v_1$  zu beschleunigen bzw. abzubremesen. Dafür wird die Gleichung für die Geschwindigkeit  $v(t)$  nach  $t(v)$  umgestellt und in die Gleichung  $s(t)$  eingesetzt. Daraus ergibt sich folgende Gleichung für die Strecke in Abhängigkeit von der Geschwindigkeit:

$$t(v) = \frac{v}{a} \quad (6)$$

$$s(v) = \frac{1}{2} \cdot \frac{v^2}{a} \quad (7)$$

Durch die Festlegung von  $v_0 = 0$  wird so die benötigte Strecke ermittelt, welche ein Fahrzeug bei einer gegebenen Bremsverzögerung  $a$  benötigt, um von  $0 \text{ m/s}$  auf eine gegebenen Zielgeschwindigkeit  $v_1$  zu beschleunigen. Bei der Berechnung des Beschleunigungs- und Abbremsverhalten wird es aber auch zu Situationen kommen, bei denen ein Fahrzeug eine Startgeschwindigkeit hat, für die gilt  $v_0 \neq 0$ . Um eine allgemein gültige Gleichung aufzustellen, wird für die Ermittlung der benötigten Strecke bei einer gegebenen Start- und Zielgeschwindigkeit die Strecke berechnet, die das Fahrzeug benötigt, um von  $0 \text{ m/s}$  auf  $v_1$  und von  $0 \text{ m/s}$  auf  $v_0$  zu beschleunigen. Für die gesuchte Strecke gilt dann:

$$s(v_0, v_1) = |s(v_1) - s(v_0)| \quad (8)$$

$$s(v_0, v_1) = \frac{1}{2} \cdot \left| \frac{v_1^2 - v_0^2}{a} \right| \quad (9)$$

In der Fahrzeugsteuerung übernimmt diese Berechnung die Funktion *getBrakeDistance()* (Code-Beispiel 9).

```

1 // Ermittlung der Strecke für eine Beschleunigung bzw. Verzögerung
2 function getBrakeDistance (float $v_0, float $v_1, float $verzoeigerung) {
3     return abs(0.5 * ((pow($v_0/3.6, 2) - pow($v_1/3.6, 2))/($verzoeigerung)));
4 }

```

Code-Beispiel 9: *getBrakeDistance()* (*functions\_math.php*)

Neben der Berechnung der Strecke ist auch die benötigte Zeit essenziell. Dafür wird mittels  $t(v)$  die Zeit berechnet, die das Fahrzeug benötigt, um von  $0 \text{ km/h}$  auf  $v_0$  bzw.  $v_1$  zu beschleunigen und aus der Differenz wird die benötigte Zeit berechnet.

$$t(v_0, v_1) = \left| \frac{v_1 - v_0}{a} \right| \quad (10)$$

In der Fahrzeugsteuerung übernimmt diese Berechnung die Funktion *getBrakeTime()* (*functions\_math.php*) (Code-Beispiel 10).

```

1 // Ermittelt die Distanz für Brems- und Verzögerungsvorgänge
2 function getBrakeTime (float $v_0, float $v_1, float $verzögerung) {
3     return abs((( $v_1/3.6)/$verzögerung) - (( $v_0/3.6)/$verzögerung));
4 }

```

Code-Beispiel 10: *getBrakeTime()* (*functions\_math.php*)

Für die Berechnung einer Gefahrenbremsung ist es notwendig zu wissen, welche Geschwindigkeit das Fahrzeug an der Position der Gefahrenstelle hat. Dafür wird die Gleichung (9) nach  $v_2$  umgestellt. Umgesetzt wird diese Gleichung mit der Funktion *getTargetBrakeSpeedWithDistanceAndStartSpeed()* (*functions\_math.php*) (Code-Beispiel 11).

$$v_2(v_1, s) = \sqrt{-2 \cdot s \cdot a} + v_1 \quad (11)$$

```

1 // Ermittelt die Geschwindigkeit, die ein Fahrzeug in einem Bremsvorgang
2 // nach einer gegebenen Distanz hat.
3 function getTargetBrakeSpeedWithDistanceAndStartSpeed (float $distance, float
4     ↳ $verzögerung, int $speed) {
5     return sqrt((-2 * $verzögerung * $distance) + (pow(($speed / 3.6), 2)))*3.6;
6 }

```

Code-Beispiel 11: *getTargetBrakeSpeedWithDistanceAndStartSpeed()* (*functions\_math.php*)

## 7.2 Formeln für gleichförmige Bewegungen

Bei einer gleichförmigen Bewegung gilt der Grundsatz:<sup>12</sup>

$$v(t) = v \quad (12)$$

Für die Berechnung der Strecke gilt wie bei der gleichmäßig beschleunigten Bewegung:<sup>13</sup>

$$s(t) = \int v(t) dt \quad (13)$$

$$s(t) = v \cdot t + s_0 \quad (14)$$

<sup>12</sup> Richard & Sander (2011, S. 22)

<sup>13</sup> Richard & Sander (2011, S. 20)

Damit die Gleichung allgemeingültig ist, wird die Integrationskonstante  $s_0$  gleich 0 gesetzt.

$$s(t) = v \cdot t \quad (15)$$

```

1 // Ermittelt die Zeit, die ein Fahrzeug bei einer gegebenen Strecke für
2 // eine gegebene Distanz benötigt
3 function distanceWithSpeedToTime (int $v, float $distance) {
4     return (($distance)/($v / 3.6));
5 }

```

Code-Beispiel 12: *distanceWithSpeedToTime()* (*functions\_math.php*)

Für die Einhaltung der exakten Ankunftszeit, muss errechnet werden, wie lange das Fahrzeug bei zwei gegebenen Geschwindigkeiten ( $v_1$  und  $v_2$ ) auf den jeweiligen Geschwindigkeiten fahren muss, um die Gesamtstrecke ( $s_{ges}$ ) und die Gesamtzeit ( $t_{ges}$ ) einzuhalten. Für die Zeiten und Strecken gilt:

$$t_{ges} = t_1 + t_2 \quad (16)$$

$$s_{ges} = s_1 + s_2 \quad (17)$$

Durch das Einsetzen der Gleichung (15) in die Gleichung (17) erhält man folgende Gleichung:

$$s_{ges} = v_1 \cdot t_1 + v_2 \cdot t_2 \quad (18)$$

Durch das Umstellen der Gleichung (16) nach  $t_2$  und dem Einsetzen in Gleichung (18) gilt für  $t_1$ :

$$t_1 = \frac{s_{ges} - v_2 \cdot t_{ges}}{v_1 - v_2} \quad (19)$$

```

1 // Ermittelt die Distanz, um die eine Verzögerung "verschoben" werden müsste,
2 // damit die exakte Ankunftszeit eingehalten werden kann.
3 function calculateDistanceforSpeedFineTuning(int $v_0, int $v_1, float
4     ↪ $distance, float $time) : float {
5     return $distance - (($distance - $time * $v_1 / 3.6)/($v_0 / 3.6 - $v_1 /
6     ↪ 3.6)) * ($v_0 / 3.6);
7 }

```

Code-Beispiel 13: *calculateDistanceforSpeedFineTuning()* (*functions\_math.php*)

## 8 Fazit

### 8.1 Zusammenfassung der Ergebnisse

Der Entwickelte Algorithmus ist in der Lage, für eine gegebene Position und Geschwindigkeit, Infra-Abschnitte inklusive deren Längen und zulässigen Höchstgeschwindigkeiten und einer Zielposition den optimalen Fahrtverlauf zu ermitteln, sodass das Fahrzeug ohne eine Überschreitung der zulässigen Höchstgeschwindigkeit und unter Berücksichtigung der Fahrzeuglänge frühestmöglich die Zielposition erreicht. Unter Berücksichtigung der Ankunftszeit wurden Ansätze entwickelt, die dafür sorgen, dass das Fahrzeug – durch eine Reduzierung der Geschwindigkeit – pünktlich das Ziel erreicht und durch eine geringere Geschwindigkeit energiesparsamer fährt. Die Ansätze für die Einhaltung der Ankunftszeit ermitteln nicht den optimalsten Fahrtverlauf, da in vereinzelt Fällen die Geschwindigkeit reduziert wird, obwohl eine Verschiebung von Brems- bzw. Verzögerungsvorgängen ausreichen würde.

Bei der Entwicklung und Testung der Fahrzeugsteuerung wurden die Fahrten am Rechner auf Grundlage der *MySQL*-Datenbank simuliert. Die dabei ermittelten Fahrtverläufe haben die erforderlichen Bedingungen (Ankunfts-, Abfahrts- und Mindesthaltezeit und zulässige Höchstgeschwindigkeit) eingehalten und die Fahrzeuge haben richtig auf Fahrstraßen-Änderungen reagiert (Einleitung einer Gefahrenbremsung und Berücksichtigung der Signalbegriffe). Bei der Verwendung der Fahrzeugsteuerung im EBUef ist es zu Fehlern gekommen, welche in dem folgenden Kapitel 8.2 erläutert werden.

### 8.2 Komplikationen bei dem Betrieb der Fahrzeugsteuerung im EBUef

#### 8.2.1 Einhaltung der Zielposition

Bei Zugfahrten ist es dazu gekommen, dass die Fahrzeuge den Bremsvorgang zu spät eingeleitet haben und an dem Ausfahrtsignal bzw. einem Halt zeigenden Signal vorbei gefahren sind. Für die Fehlerbehebung wurde überprüft, ob die eingetragenen Längen der Infra-Abschnitte aus der *MySQL*-Datenbank mit den realen Werten des EBUefs übereinstimmen. Diese mögliche Ursache konnte ausgeschlossen werden und weitere Ursachen konnten nicht ermittelt werden.

### 8.2.2 Ermittlung der Fahrstraßen

Bei der Ermittlung der Fahrstraßen hat die Funktion *getNaechsteAbschnitte()*\* (*functions\_ebuef.php*) in manchen Fällen nicht die eingestellte Fahrstraße und die erwarteten Infra-Abschnitte wiedergegeben, wodurch für Fahrzeuge kein Fahrtverlauf berechnet wurde.

### 8.2.3 Kalibrierung der Position

Bei der Kalibrierung der Fahrzeugposition wurden Positionen ermittelt, welche stark von der realen Position abwichen. Mögliche Ursachen könnten eine falsche Positionsermittlung bei der Berechnung des Fahrtverlaufs oder eine nicht rechtzeitige Eintragung der aktuellen Abschnitte in die *MySQL*-Tabelle *fahrzeuge\_abschnitte* sein. Keine der beiden Ursachen konnte eindeutig widerlegt oder belegt werden.

## 8.3 Möglichkeiten für eine Weiterentwicklung der Fahrzeugsteuerung

Durch die kontinuierliche Positionsbestimmung der Fahrzeuge ist es in zukünftigen Weiterentwicklungen der Fahrzeugsteuerung möglich, anstatt der Zugfolge im festen Raumabstand, einen Zugbetrieb im Bremswegabstand (Moving Block) zu realisieren, wodurch die Zugfolgezeiten zweier aufeinander folgender Züge deutlich reduziert werden können.<sup>14</sup>

---

<sup>14</sup> Büker et al. (2020, S. 37)

## A Anhang

### A.1 fahrzeugsteuerung.php

```
1 <?php
2
3 // Liest alle benötigten Dateien ein
4 require 'config/multicast.php';
5 require 'vorbelegung.php';
6 require 'functions/functions.php';
7 require 'functions/functions_cache.php';
8 require 'functions/functions_db.php';
9 require 'functions/functions_math.php';
10 require 'functions/functions_ebuef.php';
11 require 'functions/functions_fahrtverlauf.php';
12 require 'global_variables.php';
13
14 // Zeitzone setzen
15 date_default_timezone_set("Europe/Berlin");
16
17 // PHP-Fehlermeldungen
18 error_reporting(1);
19
20 // Globale Variablen
21 global $useRecalibration;
22
23 // Fahrzeugfehlermeldungen definieren
24 $trainErrors = array();
25 $trainErrors[0] = "Fahrtrichtung des Fahrzeugs musste geändert werden und die
    ↳ Positionsbestimmung war nicht möglich.";
26 $trainErrors[1] = "In der Datenbank ist für das Fahrzeug keine Zuglänge
    ↳ angegeben.";
27 $trainErrors[2] = "In der Datenbank ist für das Fahrzeug keine v_max angegeben.
    ↳ ";
28 $trainErrors[3] = "Das Fahrzeug musste eine Notbremsung durchführen.";
29
30 // Statische Daten einlesen
31 $cacheInfranachbarn = createCacheInfranachbarn();
32 $cacheInfradaten = createCacheInfradaten();
```

```

33 $cacheSignaldaten = createCacheSignaldaten();
34 $cacheInfraLaenge = createCacheInfraLaenge();
35 $cacheHaltepunkte = createCacheHaltepunkte();
36 $cacheZwischenhaltepunkte = createCacheZwischenhaltepunkte();
37 $cacheInfraToGbt = createCacheInfraToGbt();
38 $cacheGbtToInfra = createCacheGbtToInfra();
39 $cacheFmaToInfra = createCacheFmaToInfra();
40 $cacheInfraToFma = array_flip($cacheFmaToInfra);
41 $cacheFahrplanSession = createCacheFahrplanSession();
42 $cacheSignalIDToBetriebsstelle = createCacheToBetriebsstelle();
43 $cacheFahrzeugeAbschnitte = createCacheFahrzeugeAbschnitte();
44 $cacheIDTDecoder = createCacheDecoderToAdresse();
45 $cacheDecoderToID = array_flip($cacheIDTDecoder);
46 $cacheAdresseToID = array(); // Filled with data in getAllTrains()
47 $cacheIDToAdresse = array(); // Filled with data in getAllTrains()
48
49 // Variablendeklaration
50 $allTrainsOnTheTrack = array();
51 $allTrains = array();
52 $allUsedTrains = array();
53 $allTimes = array();
54 $lastMaxSpeedForInfraAndDir = array();
55
56 // Real- und Simulationszeit ermitteln
57 $simulationStartTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->
    ↳ sim_startzeit, "simulationszeit", null, array("outputtyp"=>"h:i:s")), "
    ↳ simulationszeit", null, array("inputtyp"=>"h:i:s"));
58 $simulationEndTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->
    ↳ sim_endzeit, "simulationszeit", null, array("outputtyp"=>"h:i:s")), "
    ↳ simulationszeit", null, array("inputtyp"=>"h:i:s"));
59 $simulationDuration = $cacheFahrplanSession->sim_endzeit -
    ↳ $cacheFahrplanSession->sim_startzeit;
60 $realStartTime = time();
61 $realEndTime = $realStartTime + $simulationDuration;
62 $timeDifference = $simulationStartTimeToday - $realStartTime;
63
64 // Startmeldung
65 startMessage();
66

```



```

67 // Ermittlung aller Fahrzeuge
68 $allTrains = getAllTrains();
69
70 // Ermittlung der Fahrzeuge im eingleisigen Netz
71 findTrainsOnTheTracks();
72
73 // Ermittlung der Fahrpläne der Fahrzeuge
74 addStopsectionsForTimetable();
75
76 // Überprüfung, ob die Fahrzeuge schon an einer Betriebsstelle des Fahrplans
    ↳ stehen
77 checkIfTrainReachedHaltepunkt();
78
79 // Überprüfung, ob die Fahrtrichtung der Fahrzeuge mit dem
80 // Fahrplan übereinstimmt. Falls die Richtung nicht übereinstimmt,
81 // wird die Fahrtrichtung der Fahrzeuge geändert
82 checkIfStartDirectionIsCorrect();
83 consoleAllTrainsPositionAndFahrplan();
84 showErrors();
85
86 // Ermittlung der Fahrstraßen aller Fahrzeuge
87 calculateNextSections();
88
89 // Überprüfung, ob die Fahrstraße für die Fahrzeuge mit Fahrplan
90 // richtig eingestellt ist
91 checkIfFahrstrasseIsCorrect();
92
93 // Ermittlung der Fahrtverläufe aller Fahrzeuge
94 calculateFahrtverlauf();
95
96 // Übermittlung der Echtzeitdaten an die Fahrzeuge
97 // $timeCheckFahrstrasseInterval => Überprüfung von Fahrstraßenänderungen
98 // $timeCheckAllTrainErrorsInterval => Ausgabe der aktuellen Positionen und
    ↳ Fahrplänen
99 // $timeCheckCalibrationInterval => Neukalibrierung der P0sition
100 $timeCheckFahrstrasseInterval = 3;
101 $timeCheckFahrstrasse = $timeCheckFahrstrasseInterval + microtime(true);
102 $timeCheckAllTrainStatusInterval = 30;
103 $timeCheckAllTrainStatus = $timeCheckAllTrainStatusInterval + microtime(true);

```

```

104 $timeCheckCalibrationInterval = 3;
105 $timeCheckCalibration = $timeCheckCalibrationInterval + microtime(true);
106
107 // Zeitintervall, in dem überprüft wird, ob neue Echtzeitdaten vorliegen
108 $sleepTime = 0.03;
109 while (true) {
110
111     // Iteration über alle Fahrzeuge
112     foreach ($allTimes as $timeIndex => $timeValue) {
113         if (sizeof($timeValue) > 0) {
114             $id = $timeValue[0]["id"];
115
116             // Überprüfung, ob der erste Eintrag der Echtzeitdaten in der
117             // Vergangenheit liegt
118             if ((microtime(true) + $timeDifference) > $timeValue[0]["live_time"]) {
119
120                 // Überprüfung, ob der Eintrag der Echtzeitdaten eine
121                 // Geschwindigkeitsveränderung beinhaltet
122                 if ($timeValue[0]["live_is_speed_change"]) {
123                     $allUsedTrains[$id]["calibrate_section_one"] = null;
124                     $allUsedTrains[$id]["calibrate_section_two"] = null;
125
126                     // Übermittlung der Echtzeitdaten bei einer Gefahrenbremsung
127                     if ($timeValue[0]["betriebsstelle"] == 'Notbremsung') {
128                         sendFahrzeugbefehl($timeValue[0]["id"], intval($timeValue[0]["
129                             ↳ live_speed"]));
130                         $allTrains[$id]["speed"] = intval($timeValue[0]["live_speed"]);
131                         echo "Der Zug mit der Adresse ", $timeIndex, " leitet gerade eine
132                             ↳ Gefahrenbremsung ein und hat seine Geschwindigkeit auf ",
133                             ↳ $timeValue[0]["live_speed"], " km/h angepasst.\n";
134                     } else {
135
136                         // Übermittlung der neuen Geschwindigkeit an das Fahrzeug
137                         sendFahrzeugbefehl($timeValue[0]["id"], intval($timeValue[0]["
138                             ↳ live_speed"]));
139                         $allTrains[$id]["speed"] = intval($timeValue[0]["live_speed"]);
140                         echo "Der Zug mit der Adresse ", $timeIndex, " hat auf der Fahrt
141                             ↳ nach ", $timeValue[0]["betriebsstelle"],

```

```

137         " seine Geschwindigkeit auf ", $timeValue[0]["live_speed"], " km/h
138         ↳ angepasst.\n";
139     }
140 } else {
141     if (isset($allUsedTrains[$id]["calibrate_section_one"])) {
142         if ($allUsedTrains[$id]["calibrate_section_one"] != $timeValue[0]["
143             ↳ live_section"]) {
144             $allUsedTrains[$id]["calibrate_section_two"] = $timeValue[0]["
145                 ↳ live_section"];
146         }
147     }
148     $allUsedTrains[$id]["calibrate_section_one"] = $timeValue[0]["
149         ↳ live_section"];
150 }
151
152 // Aktualisierung der Position im $allUsedTrains-Array
153 $allUsedTrains[$id]["current_position"] = $timeValue[0]["
154     ↳ live_relative_position"];
155 $allUsedTrains[$id]["current_speed"] = $timeValue[0]["live_speed"];
156 $allUsedTrains[$id]["current_section"] = $timeValue[0]["live_section"];
157
158 // Überprüfung, ob die Fahrtrichtung geändert werden muss
159 if ($timeValue[0]["wendet"]) {
160     changeDirection($timeValue[0]["id"]);
161 }
162
163 // Überprüfung, ob das Fahrzeug eine Betriebsstelle erreicht hat
164 if (isset($timeValue[0]["live_all_targets_reached"])) {
165     $allUsedTrains[$id]["next_betriebsstellen_data"][$timeValue[0]["
166         ↳ live_all_targets_reached"]]["angekommen"] = true;
167     echo "Der Zug mit der Adresse ", $timeIndex, " hat den Halt ",
168         ↳ $allUsedTrains[$id]["next_betriebsstellen_data"][$timeValue[0][
169             ↳ "live_all_targets_reached"]]["betriebsstelle"], " erreicht.\n";
170 }
171
172 // Überprüfung, ob ein (neuer) Fahrplan für das Fahrzeug
173 // vorliegt, wenn das ermittelte Ziel erreicht wurde
174 if ($timeValue[0]["live_target_reached"]) {

```

```

168     $currentZugId = $allUsedTrains[$id]["zug_id"];
169     $newZugId = getFahrzeugZugIds(array($id));
170
171     if (sizeof($newZugId) == 0) {
172         $newZugId = null;
173     } else {
174         $newZugId = getFahrzeugZugIds(array($timeValue[0]["id"]));
175         $newZugId = $newZugId[array_key_first($newZugId)]["zug_id"];
176     }
177
178     if (!($currentZugId == $newZugId && $currentZugId != null)) {
179         if ($currentZugId != null && $newZugId != null) {
180             // Das Fahrzeug hat einen neuen Fahrplan
181             $allUsedTrains[$id]["zug_id"] = $newZugId;
182             $allUsedTrains[$id]["operates_on_timetable"] = true;
183             getFahrplanAndPositionForOneTrain($id, $newZugId);
184             addStopsectionsForTimetable($id);
185             checkIfTrainReachedHaltepunkt($id);
186             checkIfStartDirectionIsCorrect($id);
187             calculateNextSections($id);
188             checkIfFahrstrasseIsCorrect($id);
189             calculateFahrtverlauf($id);
190         } else if ($currentZugId == null && $newZugId != null) {
191             // Das Fahrzeug hat jetzt einen Fahrplan und
192             // hatte davor keinen
193             $allUsedTrains[$id]["zug_id"] = $newZugId;
194             $allUsedTrains[$id]["operates_on_timetable"] = true;
195             getFahrplanAndPositionForOneTrain($id);
196             addStopsectionsForTimetable($id);
197             checkIfTrainReachedHaltepunkt($id);
198             checkIfStartDirectionIsCorrect($id);
199             calculateNextSections($id);
200             checkIfFahrstrasseIsCorrect($id);
201             calculateFahrtverlauf($id);
202         } else if ($currentZugId != null && $newZugId == null) {
203             // Das Fahrzeug fährt ab jetzt ohne Fahrplan
204             $allUsedTrains[$id]["operates_on_timetable"] = false;
205             calculateNextSections($id);
206             calculateFahrtverlauf($id);

```

```

207     }
208 }
209 }
210 array_shift($allTimes[$timeIndex]);
211 }
212 }
213 }
214
215 // Neukalibrierung der Position
216 if ($useRecalibration) {
217     if (microtime(true) > $timeCheckCalibration) {
218         foreach ($allUsedTrains as $trainKey => $trainValue) {
219             if (isset($allUsedTrains[$trainKey]["calibrate_section_two"])) {
220                 $newPosition = getCalibratedPosition($trainKey, $allUsedTrains[
221                     ↪ $trainKey]["current_speed"]);
222                 if ($newPosition["possible"]) {
223                     echo "Die Position des Fahrzeugs mit der ID: ", $trainKey, " wird
224                         ↪ neu ermittelt.\n";
225                     $position = $newPosition["position"];
226                     $section = $newPosition["section"];
227                     echo "Die alte Position war Abschnitt: ", $allUsedTrains[$trainKey][
228                         ↪ "current_section"], " (", number_format($allUsedTrains[
229                             ↪ $trainKey]["current_position"], 2), " m) und die neue
230                         ↪ Position ist Abschnitt: ", $section, " (", number_format(
231                             ↪ $position, 2), " m).\n";
232                     if ($position > $cacheInfraLaenge[$section]) {
233                         echo "Die Position konnte nicht neu kalibriert werden, da die
234                             ↪ aktuelle Position im Abschnitt größer ist, als die Länge
235                             ↪ des Abschnitts.\n";
236                     } else {
237                         $allUsedTrains[$trainKey]["current_section"] = $section;
238                         $allUsedTrains[$trainKey]["current_position"] = $position;
239                         calculateNextSections($trainKey);
240                         checkIfFahrstrasseIsCorrect($trainKey);
241                         calculateFahrtverlauf($trainKey, true);
242                         echo "Die Position des Fahrzeugs mit der ID: ", $trainKey, " wurde
243                             ↪ neu ermittelt.\n";
244                     }
245                 }
246             }
247         }
248     }
249 }

```

```

237     }
238 }
239 $timeCheckCalibration = $timeCheckCalibration +
    ↳ $timeCheckCalibrationInterval;
240 }
241 }
242
243 // Überprüfung, ob die Fahrstraße der einzelnen Fahrzeuge sich geändert hat
244 if (microtime(true) > $timeCheckFahrstrasse) {
245     foreach ($allUsedTrains as $trainID => $trainValue) {
246         compareTwoNaechsteAbschnitte($trainID);
247     }
248
249     $returnUpdate = updateAllTrainsOnTheTrack();
250     $newTrains = $returnUpdate["new"];
251     $removeTrains = $returnUpdate["removed"];
252
253     if (sizeof($newTrains) > 0) {
254         echo "Neu hinzugefügte Züge: \n";
255         foreach ($newTrains as $newTrain) {
256             $id = $cacheDecoderToID[$newTrain];
257             echo "\tID:\t", $id, "\tAdresse:\t", $newTrain;
258         }
259         echo "\n";
260     }
261
262     foreach ($newTrains as $newTrain) {
263         $id = $cacheDecoderToID[$newTrain];
264         prepareTrainForRide($newTrain);
265         addStopsectionsForTimetable($id);
266         checkIfTrainReachedHaltepunkt($id);
267         checkIfStartDirectionIsCorrect($id);
268         consoleAllTrainsPositionAndFahrplan($id);
269         calculateNextSections($id);
270         checkIfFahrstrasseIsCorrect($id);
271         calculateFahrtverlauf($id);
272     }
273
274     if (sizeof($removeTrains) > 0) {

```

```

275     echo "Entfernte Züge:\n";
276
277     foreach ($removeTrains as $removeTrain) {
278         $id = $cacheDecoderToID[$removeTrain];
279         unset($allUsedTrains[$id]);
280         echo "\tID:\t", $id, "\tAdresse:\t", $removeTrain;
281     }
282
283     echo "\n";
284 }
285 $timeCheckFahrstrasse = $timeCheckFahrstrasse +
    ↳ $timeCheckFahrstrasseInterval;
286 }
287
288 // Ausgabe der aktuellen Positionen, Fahrplänen und Fehlermeldungen aller
    ↳ Fahrzeuge
289 if (microtime(true) > $timeCheckAllTrainStatus) {
290     consoleAllTrainsPositionAndFahrplan();
291     showFahrplan();
292     showErrors();
293     $timeCheckAllTrainStatus = $timeCheckAllTrainStatus +
        ↳ $timeCheckAllTrainStatusInterval;
294 }
295
296 sleep($sleeptime);
297 }

```

## A.2 functions.php

```

1 <?php
2 ini_set('memory_limit', '1024M');
3
4 // Zeigt beim Starten der Fahrzeugsteuerung eine Startmeldung im Terminal an,
5 //in der Informationen zur Session angezeigt werden.
6 function startMessage() {
7     global $simulationStartTimeToday;
8     global $simulationEndTimeToday;
9     global $simulationDuration;
10    global $realStartTime;
11    global $realEndTime;
12    global $cacheFahrplanSession;
13
14    $realStartTimeAsHHMMSS = getUhrzeit($realStartTime, "simulationszeit", null,
        ↳ array("outputtyp"=>"h:i:s"));
15    $simulationEndTimeAsHHMMSS = getUhrzeit($simulationEndTimeToday, "
        ↳ simulationszeit", null, array("outputtyp"=>"h:i:s"));
16    $simulationDurationAsHHMMSS = toStd($simulationDuration);
17    $realEndTimeAsHHMMSS = getUhrzeit($realEndTime, "simulationszeit", null,
        ↳ array("outputtyp"=>"h:i:s"));
18    $simulationStartTimeAsHHMMSS = getUhrzeit($simulationStartTimeToday, "
        ↳ simulationszeit", null, array("outputtyp"=>"h:i:s"));
19    $hashtagLine = "
        ↳ #####\n";
20    $emptyLine = "#\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t#\\n";
21
22    echo $hashtagLine;
23    echo $emptyLine;
24    echo "#\\t\\t\\t Start der automatischen Zugbeeinflussung\\t\\t\\t\\t#\\n";
25    echo "#\\t\\tim Eisenbahnbetriebs- und Experimentierfeld (EBuEf) \\t\\t#\\n";
26    echo "#\\t\\t\\t\\t\\t\\t\\t der TU Berlin\\t\\t\\t\\t\\t\\t\\t#\\n";
27    echo "#\\t\\t\\t\\t\\t im eingleisigen Netz \\t\\t\\t\\t\\t\\t\\t#\\n";
28    echo $emptyLine;
29    echo "#\\t\\t\\t\\t\\t\\t\\t____\\t\\t\\t\\t\\t\\t\\t\\t\\t#\\n";
30    echo "#\\t\\t\\t\\t\\t\\t\\t|DD|____T_\\t\\t\\t\\t\\t\\t\\t\\t\\t#\\n";
31    echo "#\\t\\t\\t\\t\\t\\t\\t|_ |_____|<\\t\\t\\t\\t\\t\\t\\t\\t\\t#\\n";
32    echo "#\\t\\t\\t\\t\\t\\t\\t @-@-@-oo\\\\\\t\\t\\t\\t\\t\\t\\t\\t\\t#\\n";

```





```

68 }
69
70 if ($sekunden <= 9) {
71     $strSekunden = "0" . $sekunden;
72 } else {
73     $strSekunden = $sekunden;
74 }
75
76 return "$strStunden:$strMinuten:$strSekunden";
77 }
78
79 // Fügt ein Fahrzeug zur Fahrzeugsteuerung ($allUsedTrains) über die Adresse
80 // hinzu und ermittelt die aktuelle Position und die Fahrplandaten
81 function prepareTrainForRide(int $adresse) {
82
83     global $allUsedTrains;
84     global $allTrains;
85     global $cacheAdresseToID;
86     global $cacheFmaToInfra;
87     global $cacheInfraToFma;
88     global $cacheZwischenhaltepunkte;
89     global $cacheInfraLaenge;
90     global $globalNotverzoegerung;
91
92     $trainID = $cacheAdresseToID[$adresse];
93     $zugID = null;
94     $keysZwischenhalte = array_keys($cacheZwischenhaltepunkte);
95     $allUsedTrains[$trainID]["id"] = $allTrains[$trainID]["id"];
96     $allUsedTrains[$trainID]["adresse"] = $allTrains[$trainID]["adresse"];
97     $allUsedTrains[$trainID]["zug_id"] = null;
98     $allUsedTrains[$trainID]["verzoegerung"] = floatval($allTrains[$trainID]['
        ↪ verzoegerung']);
99     $allUsedTrains[$trainID]["notverzoegerung"] = $globalNotverzoegerung;
100     $allUsedTrains[$trainID]["zuglaenge"] = $allTrains[$trainID]["zuglaenge"];
101     $allUsedTrains[$trainID]["v_max"] = $allTrains[$trainID]["v_max"];
102     $allUsedTrains[$trainID]["dir"] = $allTrains[$trainID]["dir"];
103     $allUsedTrains[$trainID]["error"] = array();
104     $allUsedTrains[$trainID]["operates_on_timetable"] = false;
105     $allUsedTrains[$trainID]["fahrstrasse_is_correct"] = false;

```

```

106 $allUsedTrains[$trainID]["current_speed"] = intval($allTrains[$trainID]["
    ↪ speed"]);
107 $allUsedTrains[$trainID]["current_position"] = null;
108 $allUsedTrains[$trainID]["current_section"] = null;
109 $allUsedTrains[$trainID]["next_sections"] = array();
110 $allUsedTrains[$trainID]["next_lenghts"] = array();
111 $allUsedTrains[$trainID]["next_v_max"] = array();
112 $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array();
113 $allUsedTrains[$trainID]["next_bs"] = '';
114 $allUsedTrains[$trainID]["earliest_possible_start_time"] = null;
115 $allUsedTrains[$trainID]["calibrate_section_one"] = null;
116 $allUsedTrains[$trainID]["calibrate_section_two"] = null;
117
118 // Fehlerüberprüfung
119 if (!($allUsedTrains[$trainID]["zuglaenge"] > 0)) {
120     array_push($allUsedTrains[$trainID]["error"], 1);
121 }
122
123 if (!isset($allUsedTrains[$trainID]["v_max"])) {
124     array_push($allUsedTrains[$trainID]["error"], 2);
125 }
126
127 // Positionsermittlung
128 $fma = getPosition($adresse);
129
130 if (sizeof($fma) == 0) {
131     $allUsedTrains[$trainID]["current_fma_section"] = null;
132     $allUsedTrains[$trainID]["current_section"] = null;
133 } elseif (sizeof($fma) == 1) {
134     $allUsedTrains[$trainID]["current_fma_section"] = $fma[0];
135     $allUsedTrains[$trainID]["current_section"] = $cacheFmaToInfra[$fma[0]];
136 } else {
137     $infraArray = array();
138     foreach ($fma as $value) {
139         array_push($infraArray, $cacheFmaToInfra[$value]);
140     }
141     $infra = getFrontPosition($infraArray, $allTrains[$trainID]["dir"]);
142     $allUsedTrains[$trainID]["current_fma_section"] = $cacheInfraToFma[$infra];
143     $allUsedTrains[$trainID]["current_section"] = $infra;

```

```

144 }
145
146 $allUsedTrains[$trainID]["current_position"] = $cacheInfraLaenge[
    ↳ $allUsedTrains[$trainID]["current_section"]];
147 $timetableIDs = getFahrzeugZugIds(array($trainID));
148
149 if (sizeof($timetableIDs) != 0) {
150     $timetableID = $timetableIDs[array_key_first($timetableIDs)];
151     $allUsedTrains[$trainID]["zug_id"] = intval($timetableID["zug_id"]);
152     $zugID = intval($timetableID["zug_id"]);
153     $allUsedTrains[$trainID]["operates_on_timetable"] = true;
154 } else {
155     $allUsedTrains[$trainID]["zug_id"] = null;
156     $allUsedTrains[$trainID]["operates_on_timetable"] = false;
157 }
158
159 // Ermittlung der Fahrplaninformationen
160 if (isset($zugID)) {
161     $nextBetriebsstellen = getNextBetriebsstellen($zugID);
162 }
163
164 if ($zugID != null && sizeof($nextBetriebsstellen) != 0) {
165     for ($i = 0; $i < sizeof($nextBetriebsstellen); $i++) {
166         if (sizeof(explode("_", $nextBetriebsstellen[$i])) != 2) {
167             $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["
                ↳ is_on_fahrstrasse"] = false;
168             $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["betriebsstelle
                ↳ "] = $nextBetriebsstellen[$i];
169             $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["zeiten"] =
                ↳ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);
170             $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["fahrplanhalt"
                ↳ ] = true;
171         } else if (in_array($nextBetriebsstellen[$i], $keysZwischenhalte)) {
172             $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["
                ↳ is_on_fahrstrasse"] = false;
173             $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["betriebsstelle
                ↳ "] = $nextBetriebsstellen[$i];
174             $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["zeiten"] =
                ↳ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);

```

```

175         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["fahrplanhalt"
176             ↳ ] = false;
177     }
178     }
179     $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array_values(
180         ↳ $allUsedTrains[$trainID]["next_betriebsstellen_data"]);
181 } else {
182     $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array();
183 }
184 foreach ($allUsedTrains[$trainID]["next_betriebsstellen_data"] as
185     ↳ $betriebsstelleKey => $betriebsstelleValue) {
186     if ($allUsedTrains[$trainID]["next_betriebsstellen_data"][$
187         ↳ $betriebsstelleKey]["zeiten"]["abfahrt_soll"] != null) {
188         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey
189             ↳ ]["zeiten"]["abfahrt_soll_timestamp"] = getUhrzeit(
190                 ↳ $betriebsstelleValue["zeiten"]["abfahrt_soll"], "simulationszeit",
191                 ↳ null, array("inputtyp" => "h:i:s"));
192     } else {
193         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey
194             ↳ ]["zeiten"]["abfahrt_soll_timestamp"] = null;
195     }
196     if ($allUsedTrains[$trainID]["next_betriebsstellen_data"][$
197         ↳ $betriebsstelleKey]["zeiten"]["ankunft_soll"] != null) {
198         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey
199             ↳ ]["zeiten"]["ankunft_soll_timestamp"] = getUhrzeit(
200                 ↳ $betriebsstelleValue["zeiten"]["ankunft_soll"], "simulationszeit",
201                 ↳ null, array("inputtyp" => "h:i:s"));
202     } else {
203         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey
204             ↳ ]["zeiten"]["ankunft_soll_timestamp"] = null;
205     }
206     $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["
207         ↳ zeiten"]["verspaetung"] = 0;
208 }
209 }
210 // Positionsermittlung einer Zuges, wenn das Fahrzeug mehrere
211 // Infrastrukturabschnitte belegt.

```

```

200 function getFrontPosition(array $infra, int $dir) {
201
202     foreach ($infra as $section) {
203         $nextSections = array();
204         $test = getNaechsteAbschnitte($section, $dir);
205
206         foreach ($test as $value) {
207             array_push($nextSections, $value["infra_id"]);
208         }
209
210         if (sizeof(array_intersect($infra, $nextSections)) == 0) {
211             return $section;
212         }
213     }
214
215     return false;
216 }
217
218 // Ermittelt für ein Fahrzeug und die zugehörige Zug-ID den Fahrplan
219 function getFahrplanAndPositionForOneTrain (int $trainID, int $zugID) {
220
221     global $cacheZwischenhaltepunkte;
222     global $allUsedTrains;
223
224     $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array();
225     $keysZwischenhalte = array_keys($cacheZwischenhaltepunkte);
226
227     // Get timetable data
228     $nextBetriebsstellen = getNextBetriebsstellen($zugID);
229
230     if ($zugID != null && sizeof($nextBetriebsstellen) != 0) {
231         for ($i = 0; $i < sizeof($nextBetriebsstellen); $i++) {
232             if (sizeof(explode("_", $nextBetriebsstellen[$i])) != 2) {
233                 $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["
                    ↳ is_on_fahrstrasse"] = false;
234                 $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["betriebsstelle
                    ↳ "] = $nextBetriebsstellen[$i];
235                 $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["zeiten"] =
                    ↳ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);

```

```

236     $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["fahrplanhalt"
        ↳ ] = true;
237 } else if(in_array($nextBetriebsstellen[$i], $keysZwischenhalte)) {
238     $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["
        ↳ is_on_fahrstrasse"] = false;
239     $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["betriebsstelle
        ↳ "] = $nextBetriebsstellen[$i];
240     $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["zeiten"] =
        ↳ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);
241     $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["fahrplanhalt"
        ↳ ] = false;
242 }
243 }
244 $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array_values(
        ↳ $allUsedTrains[$trainID]["next_betriebsstellen_data"]);
245 } else {
246     $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array();
247 }
248
249 foreach ($allUsedTrains[$trainID]["next_betriebsstellen_data"] as
        ↳ $betriebsstelleKey => $betriebsstelleValue) {
250     if ($allUsedTrains[$trainID]["next_betriebsstellen_data"][$
        ↳ $betriebsstelleKey]["zeiten"]["abfahrt_soll"] != null) {
251         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey
            ↳ ]["zeiten"]["abfahrt_soll_timestamp"] = getUhrzeit(
            ↳ $betriebsstelleValue["zeiten"]["abfahrt_soll"], "simulationszeit",
            ↳ null, array("inputtyp" => "h:i:s"));
252     } else {
253         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey
            ↳ ]["zeiten"]["abfahrt_soll_timestamp"] = null;
254     }
255
256     if ($allUsedTrains[$trainID]["next_betriebsstellen_data"][$
        ↳ $betriebsstelleKey]["zeiten"]["ankunft_soll"] != null) {
257         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey
            ↳ ]["zeiten"]["ankunft_soll_timestamp"] = getUhrzeit(
            ↳ $betriebsstelleValue["zeiten"]["ankunft_soll"], "simulationszeit",
            ↳ null, array("inputtyp" => "h:i:s"));
258     } else {

```

```

259     $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey
        ↳ ]["zeiten"]["ankunft_soll_timestamp"] = null;
260 }
261
262     $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["
        ↳ zeiten"]["verspaetung"] = 0;
263 }
264 }
265
266 // Gibt in der Konsole für alle Züge (oder nur einen, wenn eine ID übergeben
        ↳ wird)
267 // die aktuellen Daten (Adresse, ID, Zug ID, Position, Fahrplan vorhanden,
        ↳ Fehler
268 // vorhanden und die Fahrtrichtung) aus.
269 function consoleAllTrainsPositionAndFahrplan($id = false) {
270
271     global $allUsedTrains;
272
273     $checkAllTrains = true;
274
275     if ($id != false) {
276         $checkAllTrains = false;
277     } else {
278         echo "Alle vorhandenen Züge:\n\n";
279     }
280
281     foreach ($allUsedTrains as $train) {
282         if ($checkAllTrains || $train["id"] == $id) {
283             $fahrplan = null;
284             $error = null;
285             $zugId = null;
286             if ($train["operates_on_timetable"]) {
287                 $fahrplan = "ja";
288             } else {
289                 $fahrplan = "nein";
290             }
291
292             if (sizeof($train["error"]) != 0) {
293                 $error = "ja";

```



```

294     } else {
295         $error = "nein";
296     }
297
298     if (!isset($train["zug_id"])) {
299         $zugId = '-----';
300     } else {
301         $zugId = $train["zug_id"];
302     }
303
304     echo "Zug ID: ", $train["id"], " (Adresse: ", $train["adresse"], ", Zug
        ↳ ID: ", $zugId, ") \t Führt nach Fahrplan: ",
305     $fahrplan, "\t Fahrtrichtung: ", $train["dir"], "\t Infra-Abschnitt: ",
        ↳ $train["current_section"],
306     "\t \t Aktuelle relative Position im Infra-Abschnitt: ", number_format(
        ↳ $train["current_position"], 2), "m \t \t Fehler vorhanden: \t", $error,
        ↳ "\n";
307 }
308 }
309 echo "\n";
310 }
311
312 // Zeigt für alle Züge, die nach Fahrplan fahren (oder nur für einen Zug,
313 // wenn eine ID übergeben wird) die zuletzt erreichte Betriebsstelle und
314 // die nächsten Betriebsstellen an.
315 function showFahrplan ($id = false) {
316
317     global $allUsedTrains;
318
319     $checkAllTrains = true;
320
321     if ($id != false) {
322         $checkAllTrains = false;
323     } else {
324         echo "Alle vorhandenen Fahrpläne:\n\n";
325     }
326
327     foreach ($allUsedTrains as $train) {
328         if ($checkAllTrains || $train["id"] == $id) {

```

```

329     $fahrplan = null;
330     $error = null;
331     $zugId = null;
332     if ($train["operates_on_timetable"]) {
333
334         if (!isset($train["zug_id"])) {
335             $zugId = '-----';
336         } else {
337             $zugId = $train["zug_id"];
338         }
339
340         $nextStations = '';
341         $lastStation = '';
342
343         foreach ($train["next_betriebsstellen_data"] as $bs) {
344             if (!$bs["angekommen"]) {
345                 $nextStations = $nextStations . $bs["betriebsstelle"] . ' ';
346
347             } else {
348                 $lastStation = $bs["betriebsstelle"];
349             }
350         }
351
352         if ($lastStation == '') {
353             $lastStation = '---';
354         }
355
356         echo "Zug ID: ", $train["id"], " (Adresse: ", $train["adresse"], ", Zug
            ↳ ID: ", $zugId, ")\t Letzte Station: ", $lastStation, " \t Nächste
            ↳ Stationen: ", $nextStations, "\n";
357     }
358 }
359 }
360 echo "\n";
361 }
362
363 // Über prüft für alle Fahrzeuge die nach Fahrplan fahren (oder nur für ein
364 // Fahrzeug, wenn eine ID übergeben wird), ob die Fahrtrichtung mit dem
365 // Fahrplan übereinstimmt, und ob diese geändert werden muss. Wenn die

```

```

366 // Fahrtrichtung geändert werden muss, wird die Funktion changeDirection()
367 // aufgerufen
368 function checkIfStartDirectionIsCorrect($id = false) {
369
370     global $allUsedTrains;
371
372     $checkAllTrains = true;
373
374     if ($id != false) {
375         $checkAllTrains = false;
376         echo "Für den Fall, dass die Fahrtrichtung der Züge nicht mit dem Fahrplan
           ↳ übereinstimmt, wird die Richtung verändert:\n\n";
377     } else {
378         echo "Für den Fall, dass die Fahrtrichtung des Zuges nicht mit dem Fahrplan
           ↳ übereinstimmt, wird die Richtung verändert:\n\n";
379     }
380
381     foreach ($allUsedTrains as $train) {
382         if ($checkAllTrains || $train["id"] == $id) {
383             if ($train["operates_on_timetable"]) {
384                 $endLoop = 0;
385                 for ($i = 0; $i < sizeof($train["next_betriebsstellen_data"]); $i++) {
386                     if ($train["next_betriebsstellen_data"][$i]["angekommen"]) {
387                         $endLoop = $i;
388                     }
389                 }
390
391                 if ($train["dir"] != $train["next_betriebsstellen_data"][$endLoop]["
           ↳ zeiten"]["fahrtrichtung"][1]) {
392                     changeDirection($train["id"]);
393                 }
394             }
395         }
396     }
397     echo "\n";
398 }
399
400 // Ändert die Fahrtrichtung eines Zuges, wenn das möglich ist. Sollte
401 // das Fahrzeug seine Richtung ändern müssen und ist dies nicht möglich,

```

```

402 // so wird dem Fahrzeug eine Fehlermeldung (Fehlerstatus = 0) hinzugefügt.
403 function changeDirection (int $id) {
404
405     global $allUsedTrains;
406     global $cacheInfraLaenge;
407     global $timeDifference;
408     global $allTrains;
409
410     $section = $allUsedTrains[$id]["current_section"];
411     $position = $allUsedTrains[$id]["current_position"];
412     $direction = $allUsedTrains[$id]["dir"];
413     $length = $allUsedTrains[$id]["zuglaenge"];
414     $newTrainLength = $length + ($cacheInfraLaenge[$section] - $position);
415     $newDirection = null;
416     $newSection = null;
417     $cumLength = 0;
418
419     if ($direction == 0) {
420         $newDirection = 1;
421     } else {
422         $newDirection = 0;
423     }
424
425     $newPosition = null;
426     $nextSections = getNaechsteAbschnitte($section, $newDirection);
427     $currentData = array(0 => array("laenge" => $cacheInfraLaenge[$section], "
        ↳ infra_id" => $section));
428     $mergedData = array_merge($currentData, $nextSections);
429
430     foreach ($mergedData as $sectionValue) {
431         $cumLength += $sectionValue["laenge"];
432
433         if ($newTrainLength <= $cumLength) {
434             $newSection = $sectionValue["infra_id"];
435             $newPosition = $cacheInfraLaenge[$newSection] - ($cumLength -
                ↳ $newTrainLength);
436             break;
437         }
438     }

```

```

439
440 if ($newPosition == null) {
441     echo "Die Richtung des Zugs mit der ID ", $id, " lässt sich nicht ändern,
        ↳ weil das Zugende auf einem auf Halt stehenden Signal steht.\n";
442     echo "\tDie Zuglänge beträgt:\t", $length, " m\n\tDie Distanz zwischen
        ↳ Zugende und dem auf Halt stehenden Signal beträgt:\t", ($cumLength -
        ↳ ($cacheInfraLaenge[$section] - $position)), " m\n\n";
443     array_push($allUsedTrains[$id]["error"], 0);
444 } else {
445     echo "Die Richtung des Zugs mit der ID: ", $id, " wurde auf ",
        ↳ $newDirection, " geändert.\n";
446     $allUsedTrains[$id]["current_section"] = $newSection;
447     $allUsedTrains[$id]["current_position"] = $newPosition;
448     $allUsedTrains[$id]["dir"] = $newDirection;
449     $allUsedTrains[$id]["earliest_possible_start_time"] = FZS_WARTEZEIT_WENDEN
        ↳ + time() + $timeDifference;
450     $allTrains[$id]["dir"] = $newDirection;
451     $DB = new DB_MySQL();
452     $DB->select("UPDATE '".DB_TABLE_FAHRZEUGE.'" SET '".DB_TABLE_FAHRZEUGE.'"'.
        ↳ 'dir' = $newDirection WHERE '".DB_TABLE_FAHRZEUGE.'"'. 'id' = $id");
453     unset($DB);
454     sendFahrzeugbefehl($id, -4);
455 }
456 }
457
458 // Gibt für alle Fahrzeuge die vorhanden Fehlermeldungen an.
459 function showErrors() {
460
461     global $allUsedTrains;
462     global $trainErrors;
463
464     $foundError = false;
465     echo "Hier werden für alle Züge mögliche Fehler angezeigt:\n\n";
466
467     foreach ($allUsedTrains as $trainIndex => $trainValue) {
468         if (sizeof($trainValue["error"]) != 0) {
469             $foundError = true;
470             echo "Zug ID: ", $trainValue["id"], "\n";
471             $index = 1;

```

```

472
473     foreach ($trainValue["error"] as $error) {
474         echo "\t", $index, ". Fehler:\t", $trainErrors[$error], "\n";
475         $index++;
476     }
477
478     echo "\n";
479 }
480 }
481
482 if (!$foundError) {
483     echo "Keiner der Züge hat eine Fehlermeldung.\n";
484 }
485 }
486
487 // Fügt allen Fahrzeugen (oder nur einem Fahrzeug, wenn eine ID übergeben wird)
488     ↪ ,
489 // die nach Fahrplan fahren, mögliche Halte-Infrastrukturabschnitte hinzu.
490 function addStopsectionsForTimetable($id = false) {
491
492     global $allUsedTrains;
493     global $cacheHaltepunkte;
494     global $cacheZwischenhaltepunkte;
495
496     $checkAllTrains = true;
497
498     if ($id != false) {
499         $checkAllTrains = false;
500     }
501
502     foreach ($allUsedTrains as $trainIndex => $trainValue) {
503         if ($checkAllTrains || $trainValue["id"] == $id) {
504             if (sizeof($trainValue["error"]) == 0) {
505                 if ($trainValue["operates_on_timetable"]) {
506                     foreach ($trainValue["next_betriebsstellen_data"] as
507                         ↪ $betriebsstelleKey => $betriebsstelleValue) {
508                         if (in_array($betriebsstelleValue["betriebsstelle"], array_keys(
509                             ↪ $cacheHaltepunkte))) {

```

```

507     $allUsedTrains[$trainIndex]["next_betriebsstellen_data"] [
        ↳ $betriebsstelleKey["haltepunkte"] = $cacheHaltepunkte[
        ↳ $betriebsstelleValue["betriebsstelle"]][$trainValue["dir"]];
508 } else if (in_array($betriebsstelleValue["betriebsstelle"],
        ↳ array_keys($cacheZwischenhaltepunkte))) {
509     $allUsedTrains[$trainIndex]["next_betriebsstellen_data"] [
        ↳ $betriebsstelleKey["haltepunkte"] = array(
        ↳ $cacheZwischenhaltepunkte[$betriebsstelleValue["
        ↳ betriebsstelle"]]);
510 } else {
511     $allUsedTrains[$trainIndex]["next_betriebsstellen_data"] [
        ↳ $betriebsstelleKey["haltepunkte"] = array();
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519
520 // Ermittelt für alle Fahrzeuge (wenn keine ID übergeben wird) oder für ein
521 // Fahrzeug (wenn eine ID übergeben wird) die Fahrstraße inkl. der Längen,
522 // der zulässigen Höchstgeschwindigkeiten und der IDs der nächsten Abschnitte.
523 //
524 // Die Ergebnisse können direkt im Array $usedTrains gespeichert werden
525 // ($writeResultToTrain = true) oder als return zurückgegeben werden
526 // ($writeResultToTrain = false), so dass sie verglichen werden können
527 // mit den vorherigen Daten verglichen werden können.
528 function calculateNextSections($id = false, $writeResultToTrain = true) {
529
530     global $allUsedTrains;
531     global $cacheInfraLaenge;
532     global $globalSpeedInCurrentSection;
533     global $lastMaxSpeedForInfraAndDir;
534
535     $checkAllTrains = true;
536
537     if ($id != false) {
538         $checkAllTrains = false;

```

```

539 }
540
541 foreach ($allUsedTrains as $trainIndex => $trainValue) {
542     if (($checkAllTrains || $trainValue["id"] == $id) && sizeof($trainValue["
        ↳ error"]) == 0) {
543         $dir = $trainValue["dir"];
544         $currentSectionComp = $trainValue["current_section"];
545         $signal = getSignalForSectionAndDirection($currentSectionComp, $dir);
546         $nextSectionsComp = array();
547         $nextVMaxComp = array();
548         $nextLengthsComp = array();
549         $nextSignalbegriff = null;
550
551         if ($signal != null) {
552             $nextSignalbegriff = getSignalbegriff($signal);
553             $nextSignalbegriff = $nextSignalbegriff[array_key_last(
                ↳ $nextSignalbegriff)]["geschwindigkeit"];
554             if ($nextSignalbegriff == -25) {
555                 $nextSignalbegriff = 25;
556             } else if ($nextSignalbegriff <= 0) {
557                 $nextSignalbegriff = 0;
558             }
559         } else {
560             $nextSignalbegriff = null;
561         }
562
563         $return = getNaechsteAbschnitte($currentSectionComp, $dir);
564         $allUsedTrains[$trainIndex]["last_get_naechste_abschnitte"] = $return;
565
566         if (isset($lastMaxSpeedForInfraAndDir[$trainValue["dir"]][$trainValue["
            ↳ current_section"]])) {
567             $currentVMax = $lastMaxSpeedForInfraAndDir[$trainValue["dir"]][
                ↳ $trainValue["current_section"]];
568         } else {
569             $currentVMax = $globalSpeedInCurrentSection;
570         }
571
572         array_push($nextSectionsComp, $currentSectionComp);
573         array_push($nextVMaxComp, $currentVMax);

```



```

574 array_push($nextLengthsComp, $cacheInfraLaenge[$currentSectionComp]);
575
576 if (isset($nextSignalbegriff)) {
577     $currentVMax = $nextSignalbegriff;
578 }
579
580 if ($currentVMax == 0) {
581     if ($writeResultToTrain) {
582         $allUsedTrains[$trainIndex]["next_sections"] = $nextSectionsComp;
583         $allUsedTrains[$trainIndex]["next_lengths"] = $nextLengthsComp;
584         $allUsedTrains[$trainIndex]["next_v_max"] = $nextVMaxComp;
585     } else {
586         return array($nextSectionsComp, $nextLengthsComp, $nextVMaxComp);
587     }
588 } else {
589     foreach ($return as $section) {
590         array_push($nextSectionsComp, $section["infra_id"]);
591         array_push($nextVMaxComp, $currentVMax);
592         array_push($nextLengthsComp, $cacheInfraLaenge[$section["infra_id"]]);
593         $lastMaxSpeedForInfraAndDir[intval($trainValue["dir"])] [intval(
594             ↳ $section["infra_id"]) ] = intval($currentVMax);
595         if ($section["signal_id"] != null) {
596             $signal = $section["signal_id"];
597             $nextSignalbegriff = getSignalbegriff($signal);
598             $nextSignalbegriff = $nextSignalbegriff[array_key_last(
599                 ↳ $nextSignalbegriff)]["geschwindigkeit"];
600             if ($nextSignalbegriff == -25) {
601                 $currentVMax = 25;
602             } else if ($nextSignalbegriff < 0) {
603                 $currentVMax = 0;
604             } else {
605                 $currentVMax = $nextSignalbegriff;
606             }
607         }
608     }
609     if ($writeResultToTrain) {
610         $allUsedTrains[$trainIndex]["next_sections"] = $nextSectionsComp;
611         $allUsedTrains[$trainIndex]["next_lengths"] = $nextLengthsComp;
612         $allUsedTrains[$trainIndex]["next_v_max"] = $nextVMaxComp;

```

```

611     } else {
612         return array($nextSectionsComp, $nextLengthsComp, $nextVMaxComp);
613     }
614 }
615 }
616 }
617 }
618
619 // Prüft für alle Fahrzeuge (falls keine ID übergeben wird) oder für ein
        ↳ Fahrzeug
620 // (falls eine ID übergeben wird), ob das Fahrzeug bereits am ersten fahrplanmä
        ↳ Bogen
621 // Halt ist oder nicht.
622 function checkIfTrainReachedHaltepunkt ($id = false) {
623
624     global $allUsedTrains;
625     global $cacheInfraToGbt;
626     global $cacheGbtToInfra;
627
628     $checkAllTrains = true;
629
630     if ($id != false) {
631         $checkAllTrains = false;
632     }
633
634     foreach ($allUsedTrains as $trainIndex => $trainValue) {
635         if ($checkAllTrains || $trainValue["id"] == $id) {
636             $currentInfrasection = $trainValue["current_section"];
637             $currentGbt = $cacheInfraToGbt[$currentInfrasection];
638             $allInfraSections = $cacheGbtToInfra[$currentGbt];
639             for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i++)
                ↳ {
640                 if (sizeof(array_intersect($trainValue["next_betriebsstellen_data"][$i][
                    ↳ "haltepunkte"], $allInfraSections)) != 0) {
641                     $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$i]["
                        ↳ angekommen"] = true;
642                     for ($j = 0; $j < $i; $j++) {
643                         $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$j]["
                            ↳ angekommen"] = true;

```

```

644     }
645   } else {
646     $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$i]["
        ↳ angekommen"] = false;
647   }
648 }
649 }
650 }
651 }
652
653 // Prüft für alle Fahrzeuge (falls keine ID übergeben wird) oder für ein
        ↳ Fahrzeug
654 // (falls eine ID übergeben wird), ob die Fahrstraße aktuell richtig
        ↳ eingestellt
655 // ist, sodass die nächste Betriebsstelle laut Fahrplan erreicht werden kann.
656 //
657 // Für Züge ohne Fahrplan ist der Fahrweg immer korrekt.
658 function checkIfFahrstrasseIsCorrect($id = false) {
659
660   global $allUsedTrains;
661
662   $checkAllTrains = true;
663
664   if ($id != false) {
665     $checkAllTrains = false;
666   }
667
668   foreach ($allUsedTrains as $trainIndex => $trainValue) {
669     if (($checkAllTrains || $trainValue["id"] == $id) && sizeof($trainValue["
        ↳ error"]) == 0) {
670       if ($trainValue["operates_on_timetable"]) {
671         $allUsedTrains[$trainIndex]["fahrstrasse_is_correct"] = false;
672         foreach ($trainValue["next_betriebsstellen_data"] as $stopIndex =>
            ↳ $stopValue) {
673           if (!$stopValue["angekommen"]) {
674             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$stopIndex
                ↳ ]["is_on_fahrstrasse"] = false;
675             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$stopIndex
                ↳ ]["used_haltepunkt"] = array();

```

```

676     $indexSection = 0;
677     for ($i = 0; $i < sizeof($trainValue["next_sections"]); $i++) {
678         if ($stopValue["haltepunkte"] != null) {
679             if (in_array($trainValue["next_sections"][$i], $stopValue["
        ↳ haltepunkte"]))) {
680                 if ($i >= $indexSection) {
681                     $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
        ↳ $stopIndex]["is_on_fahrstrasse"] = true;
682                     $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
        ↳ $stopIndex]["used_haltepunkt"] = $trainValue["
        ↳ next_sections"][$i];
683                     $allUsedTrains[$trainIndex]["fahrstrasse_is_correct"] = true;
684                     $i = sizeof($trainValue["next_sections"]);
685                     $indexSection = $i;
686                 }
687             }
688         }
689     }
690     } else {
691         $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$stopIndex
        ↳ ]["is_on_fahrstrasse"] = true;
692     }
693 }
694 } else {
695     $allUsedTrains[$trainIndex]["fahrstrasse_is_correct"] = true;
696 }
697 }
698 }
699 }
700
701 // Berechnet die Beschleunigungs- und Bremskurven für alle Züge (wenn keine ID
702 // übergeben wird) oder für einen Zug (wenn eine ID übergeben wird). Für Züge -
703 // die nach Fahrplan fahren - für alle Betriebsstellen, die auf der aktuell
704 // eingestellten Strecke liegen und für Züge ohne Fahrplan bis zum nächsten
705 // roten Signal.
706 function calculateFahrtverlauf($id = false, $recalibrate = false) {
707
708     global $allUsedTrains;
709     global $cacheInfraLaenge;

```

```

710 global $timeDifference;
711 global $globalFirstHaltMinTime;
712
713 $checkAllTrains = true;
714
715 if ($id != false) {
716     $checkAllTrains = false;
717 }
718
719 foreach ($allUsedTrains as $trainIndex => $trainValue) {
720     $allPossibleStops = array();
721     for($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i++) {
722         if ($trainValue["next_betriebsstellen_data"][$i]["fahrplanhalt"]) {
723             array_push($allPossibleStops, $i);
724         }
725     }
726     if (sizeof($trainValue["error"]) == 0 && $trainValue["
        ↳ fahrstrasse_is_correct"]) {
727         if ($checkAllTrains || $trainValue["id"] == $id) {
728             if ($trainValue["operates_on_timetable"]) {
729                 $nextBetriebsstelleIndex = null;
730                 $allreachedInfras = array();
731                 $wendet = false;
732                 for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i
                    ↳ ++)) {
733                     if (!$trainValue["next_betriebsstellen_data"][$i]["angekommen"] &&
                        ↳ $trainValue["next_betriebsstellen_data"][$i]["
                        ↳ is_on_fahrstrasse"] && $trainValue["next_betriebsstellen_data
                        ↳ "][$i]["fahrplanhalt"]) {
734                         $nextBetriebsstelleIndex = $i;
735                         $allUsedTrains[$trainIndex]["next_bs"] = $i;
736                         break;
737                     }
738                 }
739                 if (!isset($nextBetriebsstelleIndex)) {
740                     for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]);
                        ↳ $i++) {
741                         if (!$trainValue["next_betriebsstellen_data"][$i]["angekommen"] &&
                            ↳ $trainValue["next_betriebsstellen_data"][$i]["

```

```

742         ↪ is_on_fahrstrasse")) {
743         $nextBetriebsstelleIndex = $i;
744         break;
745     }
746 }
747 if (isset($nextBetriebsstelleIndex)) {
748     if ($allUsedTrains[$trainIndex]["next_bs"] != $trainValue["
749         ↪ next_betriebsstellen_data"][$nextBetriebsstelleIndex]["
750         ↪ betriebsstelle"] || $recalibrate) {
751         $allUsedTrains[$trainIndex]["next_bs"] = $trainValue["
752         ↪ next_betriebsstellen_data"][$nextBetriebsstelleIndex]["
753         ↪ betriebsstelle"];
754     if (intval($trainValue["next_betriebsstellen_data"][
755         ↪ $nextBetriebsstelleIndex]["zeiten"]["wendet"]) == 1) {
756         $wendet = true;
757     }
758     for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"])
759         ↪ ; $i++) {
760         if (!$trainValue["next_betriebsstellen_data"][$i]["angekommen"]
761             ↪ && $trainValue["next_betriebsstellen_data"][$i]["
762             ↪ is_on_fahrstrasse"] && $i <= $nextBetriebsstelleIndex) {
763             array_push($allreachedInfras, array("index" => $i, "infra" =>
764                 ↪ $trainValue["next_betriebsstellen_data"][$i]["
765                 ↪ used_haltepunkt"]));
766         }
767     }
768     $targetSection = $trainValue["next_betriebsstellen_data"][$
769         ↪ $nextBetriebsstelleIndex]["used_haltepunkt"];
770     $targetPosition = $cacheInfraLaenge[$targetSection];
771     $startTime = null;
772     $endTime = null;
773     $prevBetriebsstelle = null;
774     for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"])
775         ↪ ; $i++) {
776         if ($trainValue["next_betriebsstellen_data"][$i]["angekommen"])
777             ↪ {
778             $prevBetriebsstelle = $i;
779             break;

```

```

767     }
768 }
769 if ($nextBetriebsstelleIndex == 0) {
770     $startTime = microtime(true) + $timeDifference;
771     $endTime = $startTime;
772 } else {
773     $endTime = $trainValue["next_betriebsstellen_data"][
        ↳ $nextBetriebsstelleIndex]["zeiten"]["
        ↳ ankunft_soll_timestamp"];
774     if (isset($prevBetriebsstelle)) {
775         if ($trainValue["next_betriebsstellen_data"][
            ↳ $prevBetriebsstelle]["zeiten"]["verspaetung"] > 0) {
776             $startTime = $trainValue["next_betriebsstellen_data"][
                ↳ $prevBetriebsstelle]["zeiten"]["abfahrt_soll_timestamp
                ↳ "] + $trainValue["next_betriebsstellen_data"][
                ↳ $nextBetriebsstelleIndex - 1]["zeiten"]["verspaetung"
                ↳ ];
777         } else {
778             $startTime = $trainValue["next_betriebsstellen_data"][
                ↳ $prevBetriebsstelle]["zeiten"]["abfahrt_soll_timestamp
                ↳ "];
779         }
780     } else {
781         $startTime = microtime(true) + $timeDifference;
782     }
783 }
784 $reachedBetriebsstele = true;
785
786 if ($startTime < microtime(true) + $timeDifference) {
787     $startTime = microtime(true) + $timeDifference;
788 }
789
790 if (isset($trainValue["earliest_possible_start_time"])) {
791     if ($startTime < $trainValue["earliest_possible_start_time"]) {
792         $startTime = $trainValue["earliest_possible_start_time"];
793     }
794 }
795

```

```

796     $verapetung = updateNextSpeed($trainValue, $startTime, $endTime,
797         ↳ $targetSection, $targetPosition, $reachedBetriebsstele,
798         ↳ $nextBetriebsstelleIndex, $wendet, false, $allreachedInfras
799         ↳ );
800
801     if ($nextBetriebsstelleIndex != 0) {
802         $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
803             ↳ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] =
804             ↳ $verapetung;
805         $trainValue["next_betriebsstellen_data"][
806             ↳ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] =
807             ↳ $verapetung;
808     } else {
809         $end = $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
810             ↳ $nextBetriebsstelleIndex]["zeiten"]["
811             ↳ abfahrt_soll_timestamp"];
812         $start = $startTime;
813         if ($start + $verapetung + $globalFirstHaltMinTime < $end) {
814             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
815                 ↳ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] = 0;
816             $trainValue["next_betriebsstellen_data"][
817                 ↳ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] = 0;
818         } else {
819             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
820                 ↳ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] =
821                 ↳ $start + $verapetung + $globalFirstHaltMinTime - $end;
822             $trainValue["next_betriebsstellen_data"][
823                 ↳ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] =
824                 ↳ $start + $verapetung + $globalFirstHaltMinTime - $end;
825         }
826     }
827 } else {
828     if ($trainValue["current_speed"] > 0) {
829         emergencyBreak($trainValue["id"]);
830     }
831 }
832 } else {
833     $startTime = microtime(true) + $timeDifference;

```



```

820     if (isset($trainValue["earliest_possible_start_time"])) {
821         if ($startTime < $trainValue["earliest_possible_start_time"]) {
822             $startTime = $trainValue["earliest_possible_start_time"];
823         }
824     }
825     $endTime = $startTime;
826     $targetSection = null;
827     $targetPosition = null;
828     $reachedBetriebsstele = true;
829     $wendet = false;
830     $signalId = null;
831     for ($i = 0; $i < sizeof($trainValue["last_get_naechste_abschnitte"]);
832         ↪ $i++) {
833         if (isset($trainValue["last_get_naechste_abschnitte"][$i]["signal_id"]
834             ↪ "])) {
835             $signalId = $trainValue["last_get_naechste_abschnitte"][$i]["
836                 ↪ signal_id"];
837             $targetSection = $trainValue["last_get_naechste_abschnitte"][$i]["
838                 ↪ infra_id"];
839             $targetPosition = $cacheInfraLaenge[$targetSection];
840         }
841     }
842     if (!isset($signalId)) {
843         // gibt kein nächstes Signal
844         if ($trainValue["current_speed"] != 0) {
845             emergencyBreak($trainValue["id"]);
846         }
847     } else {
848         $signal = getSignalbegriff($signalId)[0]["geschwindigkeit"];
849
850         if ($signal > -25 && $signal < 0) {
851             $wendet = true;
852         }
853
854         updateNextSpeed($trainValue, $startTime, $endTime, $targetSection,
855             ↪ $targetPosition, $reachedBetriebsstele, $signalId, $wendet,
856             ↪ true, array());
857     }
858 }

```

```

853     }
854 } else {
855     if ($trainValue["current_speed"] != 0) {
856         emergencyBreak($trainValue["id"]);
857     }
858 }
859 }
860 }
861
862 // Vergleicht für ein Fahrzeug die zuletzt ermittelte Fahrstraße mit der
863     ↳ aktuellen
864 // Fahrstraße und berechnet den Fahrtverlauf neu, wenn das nötig ist.
865 function compareTwoNaechsteAbschnitte(int $id) {
866
867     global $allUsedTrains;
868     global $allTimes;
869
870     if (sizeof($allUsedTrains[$id]["error"]) == 0) {
871         $newSections = calculateNextSections($id, false);
872         $newNextSection = $newSections[0];
873         $newNextLengths = $newSections[1];
874         $newNextVMax = $newSections[2];
875         $oldNextSections = $allUsedTrains[$id]["next_sections"];
876         $oldLengths = $allUsedTrains[$id]["next_lengths"];
877         $oldNextVMax = $allUsedTrains[$id]["next_v_max"];
878         $currentSectionOld = $allUsedTrains[$id]["current_section"];
879         $keyCurrentSection = array_search($currentSectionOld, $oldNextSections);
880         $keyLatestSection = array_key_last($oldNextSections);
881         $dataIsIdentical = true;
882         $numberOfSection = $keyLatestSection - $keyCurrentSection + 1;
883         $compareNextSections = array();
884         $compareNextLengths = array();
885         $compareNextVMax = array();
886
887         for($i = $keyCurrentSection; $i <= $keyLatestSection; $i++) {
888             array_push($compareNextSections, $oldNextSections[$i]);
889             array_push($compareNextLengths, $oldLengths[$i]);
890             array_push($compareNextVMax, $oldNextVMax[$i]);
891         }

```

```

891
892 if (sizeof($newNextSection) != ($numberOfSection)) {
893     $dataIsIdentical = false;
894 } else {
895     for ($i = 0; $i < $keyLatestSection - $keyCurrentSection; $i++) {
896         if ($newNextSection[$i] != $compareNextSections[$i] || $newNextLengths[
            ↳ $i] != $compareNextLengths[$i] || $newNextVMax[$i] !=
            ↳ $compareNextVMax[$i]) {
897             $dataIsIdentical = false;
898             break;
899         }
900     }
901 }
902
903 if (!$dataIsIdentical) {
904     echo "Die Fahrstraße des Zuges mit der ID: ", $id, " hat sich geändert.\n
        ↳ ";
905     calculateNextSections($id);
906     $adresse = $allUsedTrains[$id]["adresse"];
907     $allTimes[$adresse] = array();
908     checkIfFahrstrasseIsCorrect($id);
909     calculateFahrtverlauf($id);
910 }
911 }
912 }

```

### A.3 functions\_fahrtverlauf.php

```
1 <?php
2
3 // Berechnet den Fahrtverlauf eines Fahrzeugs
4 function updateNextSpeed (array $train, float $startTime, float $endTime, int
    ↳ $targetSectionPara, int $targetPositionPara, bool $reachedBetriebsstelle
    ↳ , string $indexReachedBetriebsstelle, bool $wendet, bool $freieFahrt,
    ↳ array $allreachedInfras) {
5
6     global $useSpeedFineTuning;
7     global $next_sections;
8     global $next_lengths;
9     global $next_v_max;
10    global $allTimes;
11    global $verzoegerung;
12    global $notverzoegerung;
13    global $currentSection;
14    global $currentPosition;
15    global $currentSpeed;
16    global $targetSpeed;
17    global $targetSection;
18    global $targetPosition;
19    global $targetTime;
20    global $indexCurrentSection;
21    global $indexTargetSection;
22    global $distanceToNextStop;
23    global $trainSpeedChange;
24    global $trainPositionChange;
25    global $trainTimeChange;
26    global $cumulativeSectionLengthEnd;
27    global $cumulativeSectionLengthStart;
28    global $keyPoints;
29    global $allUsedTrains;
30    global $globalIndexBetriebsstelleFreieFahrt;
31    global $cacheSignalIDToBetriebsstelle;
32    global $useMinTimeOnSpeed;
33    global $slowDownIfTooEarly;
34    global $globalFloatingPointNumbersRoundingError;
35
```

```

36 $emptyArray = array();
37 $keyPoints = $emptyArray;
38 $cumulativeSectionLengthStart = $emptyArray;
39 $cumulativeSectionLengthEnd = $emptyArray;
40 $next_sections = $train["next_sections"];
41 $next_lengths = $train["next_lengths"];
42 $next_v_max = $train["next_v_max"];
43 $verzoegerung = $train["verzoegerung"];
44 $notverzoegerung = $train["notverzoegerung"];
45 $train_v_max = $train["v_max"];
46 $currentSection = $train["current_section"];
47 $currentPosition = $train["current_position"];
48 $currentSpeed = $train["current_speed"];
49 $train_length = $train["zuglaenge"];
50 $targetSection = $targetSectionPara;
51 $targetPosition = $targetPositionPara;
52 $targetSpeed = 0;
53 $targetTime = $endTime;
54 $indexCurrentSection = null;
55 $indexTargetSection = null;
56 $timeToNextStop = null;
57 $maxTimeToNextStop = $targetTime - $startTime;
58 $maxSpeedNextSections = 120;
59
60 if (!$freieFahrt) {
61     $targetBetriebsstelle = $train["next_betriebsstellen_data"][
        ↳ $indexReachedBetriebsstelle]["betriebsstelle"];
62 } else {
63     $targetBetriebsstelle = $cacheSignalIDToBetriebsstelle[intval(
        ↳ $indexReachedBetriebsstelle)];
64 }
65
66 // Überprüfung, ob das Fahrzeug bereits am Ziel steht
67 if ($targetSection == $currentSection && $targetPosition == $currentPosition)
    ↳ {
68     if ($currentSpeed > 0) {
69         emergencyBreak($train["id"]);
70     } else {
71         $allTimes[$train["adresse"]] = array();

```

```

72     return 0;
73 }
74 }
75
76 // Wenn ein Infra-Abschnitt eine Geschwindigkeit zulässt, die größer als die
77 // zulässige Höchstgeschwindigkeit des Fahrzeugs ist, wird die
78     ↳ Geschwindigkeit
79 // des Infra-Abschnitts reduziert.
80 if ($train_v_max != null) {
81     foreach ($next_sections as $sectionKey => $sectionValue) {
82         if ($next_v_max[$sectionKey] > $train_v_max) {
83             $next_v_max[$sectionKey] = $train_v_max;
84         }
85     }
86 }
87
88 // Ermittlung der Indexe des Start- und Zielabschnitts
89 foreach ($next_sections as $sectionKey => $sectionValue) {
90     if ($sectionValue == $currentSection) {
91         $indexCurrentSection = $sectionKey;
92     }
93
94     if ($sectionValue == $targetSection) {
95         $indexTargetSection = $sectionKey;
96     }
97 }
98
99 // Berechnet die kumulierten Abstände jedes Infra-Abschnitts für den Anfang
100 // und das Ende der Infra-Abschnitt von der aktuellen Fahrzeugposition
101 $returnCumulativeSections = createCumulativeSections($indexCurrentSection,
102     ↳ $indexTargetSection, $currentPosition, $targetPosition, $next_lengths)
103     ↳ ;
104 $cumulativeSectionLengthStart = $returnCumulativeSections[0];
105 $cumulativeSectionLengthEnd = $returnCumulativeSections[1];
106 $cumLengthEnd = array();
107 $cumLengthStart = array();
108 $sum = 0;
109
110 foreach ($next_lengths as $index => $value) {

```

```

108     if ($index >= $indexCurrentSection) {
109         $cumLengthStart[$index] = $sum;
110         $sum += $value;
111         $cumLengthEnd[$index] = $sum;
112     }
113 }
114
115 // Ermittlung der Distanz bis zum Ziel
116 $distanceToNextStop = $cumulativeSectionLengthEnd[$indexTargetSection];
117 if (getBrakeDistance($currentSpeed, $targetSpeed, $verzoegerung)>
118     ↪ $distanceToNextStop && $currentSpeed != 0) {
119     if (!isset($distanceToNextStop)) {
120         emergencyBreak($train["id"]);
121
122         return 0;
123     } else {
124         emergencyBreak($train["id"], $distanceToNextStop);
125
126         return 0;
127     }
128 }
129
130 // Ermittlung der Längen und zulässigen Höchstgeschwindigkeiten der
131 // Infra-Abschnitte inkl. Zuglänge
132 global $next_v_max_mod;
133 global $next_lengths_mod;
134 global $indexCurrentSectionMod;
135 global $indexTargetSectionMod;
136
137 $next_v_max_mod = array();
138 $next_lengths_mod = array();
139 $indexCurrentSectionMod = null;
140 $indexTargetSectionMod = null;
141
142 if ($indexCurrentSection == $indexTargetSection) {
143     $next_lengths_mod = $next_lengths;
144     $next_v_max_mod = $next_v_max;
145     $indexCurrentSectionMod = $indexCurrentSection;
146     $indexTargetSectionMod = $indexTargetSection;

```

```

146     $next_lengths_mod[$indexTargetSectionMod] = $targetPosition;
147 } else {
148     $startPosition = 0;
149     $indexStartPosition = null;
150     $indexEndPosition = null;
151
152     do {
153         $reachedTargetSection = false;
154
155         for ($j = $indexCurrentSection; $j <= $indexTargetSection; $j++) {
156             if ($startPosition >= $cumLengthStart[$j] && $startPosition <
157                 ↪ $cumLengthEnd[$j]) {
158                 $indexStartPosition = $j;
159             }
160         }
161
162         $endPosition = $cumLengthEnd[$indexStartPosition] + $strain_length;
163         $current_v_max = $next_v_max[$indexStartPosition];
164
165         if ($endPosition >= $cumLengthEnd[$indexTargetSection]) {
166             $indexEndPosition = $indexTargetSection;
167             $endPosition = $cumLengthEnd[$indexTargetSection - 1] + $targetPosition;
168             $reachedTargetSection = true;
169         } else {
170             for ($j = $indexCurrentSection; $j <= $indexTargetSection; $j++) {
171                 if ($endPosition >= $cumLengthStart[$j] && $endPosition <
172                     ↪ $cumLengthEnd[$j]) {
173                     $indexEndPosition = $j;
174                 }
175             }
176
177             for ($j = $indexStartPosition + 1; $j <= $indexEndPosition; $j++) {
178                 if ($next_v_max[$j] < $current_v_max) {
179                     $endPosition = $cumLengthStart[$j];
180                     $indexEndPosition = $j - 1;
181                 }
182             }

```



```

183     if ($reachedTargetSection) {
184         if (!($endPosition >= $distanceToNextStop)) {
185             $reachedTargetSection = false;
186         }
187     }
188
189     array_push($next_lengths_mod, ($endPosition - $startPosition));
190     array_push($next_v_max_mod, $current_v_max);
191     $startPosition = $endPosition;
192 } while (!$reachedTargetSection);
193
194 $indexCurrentSectionMod = array_key_first($next_lengths_mod);
195 $indexTargetSectionMod = array_key_last($next_lengths_mod);
196 }
197
198 // Berechnet die kumulierten Abstände jedes Infra-Abschnitts für den Anfang
199 // und das Ende der Infra-Abschnitt von der aktuellen Fahrzeugposition
200 // inkl. der Fahrzeuglänge
201 $returnCumulativeSectionsMod = createCumulativeSections(
202     ↪ $indexCurrentSectionMod, $indexTargetSectionMod, $currentPosition,
203     ↪ $next_lengths_mod[$indexTargetSectionMod], $next_lengths_mod);
204
205 global $cumulativeSectionLengthStartMod;
206 global $cumulativeSectionLengthEndMod;
207
208 $cumulativeSectionLengthStartMod = $returnCumulativeSectionsMod[0];
209 $cumulativeSectionLengthEndMod = $returnCumulativeSectionsMod[1];
210 $minTimeOnSpeedIsPossible = checkIfItsPossible($train["id"]);
211 $v_maxFirstIteration = getVMaxBetweenTwoPoints($distanceToNextStop,
212     ↪ $currentSpeed, $targetSpeed, $train["id"]);
213
214 // Anpassung an die maximale Geschwindigkeit auf der Strecke
215 for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
216     if ($next_v_max[$i] > $maxSpeedNextSections) {
217         $maxSpeedNextSections = $next_v_max[$i];
218     }
219 }
220
221 if ($maxSpeedNextSections < $v_maxFirstIteration) {
222     $v_maxFirstIteration = $maxSpeedNextSections;

```

```

219 }
220
221 // Key Points für die erste Iteration erstellen.
222 array_push($keyPoints, createKeyPoint(0, getBrakeDistance($currentSpeed,
    ↳ $v_maxFirstIteration, $verzoegerung), $currentSpeed,
    ↳ $v_maxFirstIteration));
223 array_push($keyPoints, createKeyPoint(($distanceToNextStop - getBrakeDistance
    ↳ ($v_maxFirstIteration, $targetSpeed, $verzoegerung)),
    ↳ $distanceToNextStop, $v_maxFirstIteration, $targetSpeed));
224
225 //$trainChange = convertKeyPointsToTrainChangeArray($keyPoints);
226 $trainChange = createTrainChanges(true);
227 $trainPositionChange = $trainChange[0];
228 $trainSpeedChange = $trainChange[1];
229 $speedOverPositionAllIterations = array();
230
231 // Überprüfung, ob das Fahrzeug in Infra-Abschnitten zu schnell ist
232 while (checkIfTrainIsToFastInCertainSections()["failed"]) {
233     $tempKeyPoints = $keyPoints;
234
235     // Berechnung der Echtzeitdaten
236     $trainChange = createTrainChanges(true);
237     $trainPositionChange = $trainChange[0];
238     $trainSpeedChange = $trainChange[1];
239
240     // Hinzufügen der Echtzeitdaten des vorherigen Iterationsschritt
241     // für die Visualisierung
242     array_push($speedOverPositionAllIterations, array($trainPositionChange,
        ↳ $trainSpeedChange));
243
244     // Überprüfung, ob durch den Fahrtverlauf zulässige Höchst-
245     // geschwindigkeiten überschritten werden
246     $keyPoints = recalculateKeyPoints($tempKeyPoints, $train["id"]);
247     $localKeyPointsTwo = array();
248
249     // Entfernen von doppelten $keyPoints
250     for ($i = 0; $i < sizeof($keyPoints); $i++) {
251         if ($i < sizeof($keyPoints) - 1) {

```

```

252     if (!($keyPoints[$i]["speed_0"] == $keyPoints[$i]["speed_1"] &&
        ↳ $keyPoints[$i]["speed_0"] == $keyPoints[$i + 1]["speed_0"] &&
        ↳ $keyPoints[$i]["speed_0"] == $keyPoints[$i + 1]["speed_1"])) {
253         array_push($localKeyPointsTwo, $keyPoints[$i]);
254     } else {
255         $i++;
256     }
257 } else {
258     array_push($localKeyPointsTwo, $keyPoints[$i]);
259 }
260 }
261
262 // Berechnung der Echtzeitdaten nach der Neukalibrierung
263 $keyPoints = $localKeyPointsTwo;
264 $trainChange = createTrainChanges(true);
265 $trainPositionChange = $trainChange[0];
266 $trainSpeedChange = $trainChange[1];
267 }
268
269 // Fügt die aktuelle Zeit zum ersten $keyPoint hinzu
270 $keyPoints[0]["time_0"] = $startTime;
271 $keyPoints = deleteDoubledKeyPoints($keyPoints);
272 $keyPoints = calculateTimeFromKeyPoints();
273
274 if ($useMinTimeOnSpeed && $minTimeOnSpeedIsPossible) {
275     array_push($speedOverPositionAllIterations, array($trainPositionChange,
        ↳ $trainSpeedChange));
276     toShortOnOneSpeed();
277 }
278
279 // Ermittlung der Echtzeitdaten
280 $trainChange = createTrainChanges(true);
281 $trainPositionChange = $trainChange[0];
282 $trainSpeedChange = $trainChange[1];
283 $timeToNextStop = end($keyPoints)["time_1"] - $keyPoints[0]["time_0"];
284
285 // Überprüfung, ob das Fahrzeug mit einer Verspätung am Ziel ankommt.
286 // Fahrzeuge, die ohne Fahrplan fahren, werden nicht betrachtet.
287 if (!$freieFahrt) {

```

```

288 if ($timeToNextStop > $maxTimeToNextStop) {
289     echo "Der Zug mit der Adresse ", $train["adresse"], " wird mit einer
        ↳ Verspätung von ", number_format($timeToNextStop -
        ↳ $maxTimeToNextStop, 2), " Sekunden im nächsten planmäßigen Halt (",
        ↳ $targetBetriebsstelle,") ankommen.\n";
290 } else {
291     echo "Aktuell benötigt der Zug mit der Adresse ", $train["adresse"], " ",
        ↳ number_format($timeToNextStop, 2), " Sekunden, obwohl er ",
        ↳ number_format($maxTimeToNextStop, 2), " Sekunden zur Verfügung hat
        ↳ .\n";
292
293 if ($slowDownIfTooEarly) {
294     echo "Evtl. könnte der Zug zwischendurch die Geschwindigkeit verringern,
        ↳ um Energie zu sparen.";
295
296     array_push($speedOverPositionAllIterations, array($trainPositionChange,
        ↳ $trainSpeedChange));
297     $keyPointsPreviousStep = array();
298     $finish = false;
299     $possibleSpeedRange = null;
300     $returnSpeedDecrease = checkIfTheSpeedCanBeDecreased();
301
302     while ($returnSpeedDecrease["possible"] && !$finish) {
303         $possibleSpeedRange = findMaxSpeed($returnSpeedDecrease);
304
305         if ($possibleSpeedRange["min_speed"] == $possibleSpeedRange["max_speed"]
            ↳ ") {
306             break;
307         }
308
309         $localKeyPoints = $keyPoints;
310         $newCalculatedTime = null;
311         $newKeyPoints = null;
312
313         for ($i = $possibleSpeedRange["max_speed"]; $i >= $possibleSpeedRange[
            ↳ "min_speed"]; $i = $i - 10) {
314             $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["
                ↳ speed_1"] = $i;

```

```

315     $localKeyPoints[$possibleSpeedRange["first_key_point_index"] + 1]["
        ↳ speed_0"] = $i;
316     $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["
        ↳ position_1"] = (getBrakeDistance($localKeyPoints[
        ↳ $possibleSpeedRange["first_key_point_index"]]["speed_0"], $i,
        ↳ $verzoegerung) + $localKeyPoints[$possibleSpeedRange["
        ↳ first_key_point_index"]]["position_0"]);
317     $localKeyPoints[$possibleSpeedRange["first_key_point_index"] + 1]["
        ↳ position_0"] = ($localKeyPoints[$possibleSpeedRange["
        ↳ first_key_point_index"] + 1]["position_1"] - getBrakeDistance
        ↳ ($i, $localKeyPoints[$possibleSpeedRange["
        ↳ first_key_point_index"] + 1]["speed_1"], $verzoegerung));
318     $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
319     $newCalculatedTime = $localKeyPoints[array_key_last($localKeyPoints)
        ↳ ]["time_1"];
320
321     if ($i == 10) {
322         if ($newCalculatedTime > $maxTimeToNextStop) {
323             $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["
                ↳ speed_1"] = $i + 10;
324             $localKeyPoints[$possibleSpeedRange["first_key_point_index"] +
                ↳ 1]["speed_0"] = $i + 10;
325             $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["
                ↳ position_1"] = (getBrakeDistance($localKeyPoints[
                ↳ $possibleSpeedRange["first_key_point_index"]]["speed_0"],
                ↳ ($i + 10), $verzoegerung) + $localKeyPoints[
                ↳ $possibleSpeedRange["first_key_point_index"]]["position_0"
                ↳ ]);
326             $localKeyPoints[$possibleSpeedRange["first_key_point_index"] +
                ↳ 1]["position_0"] = ($localKeyPoints[$possibleSpeedRange["
                ↳ first_key_point_index"] + 1]["position_1"] -
                ↳ getBrakeDistance(($i + 10), $localKeyPoints[
                ↳ $possibleSpeedRange["first_key_point_index"] + 1]["speed_1
                ↳ "], $verzoegerung));
327         }
328
329         $finish = true;
330         $newKeyPoints = $localKeyPoints;
331         break;

```

```

332 }
333 if (($newCalculatedTime - $startTime) > $maxTimeToNextStop) {
334     if ($i == $possibleSpeedRange["max_speed"]) {
335         $localKeyPoints = $keyPointsPreviousStep;
336         $localKeyPoints = deleteDoubledKeyPoints($localKeyPoints);
337         $keyPoints = $localKeyPoints;
338         $finish = true;
339         break;
340     }
341     $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["
        ↪ speed_1"] = $i + 10;
342     $localKeyPoints[$possibleSpeedRange["first_key_point_index"] + 1]["
        ↪ speed_0"] = $i + 10;
343     $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["
        ↪ position_1"] = (getBrakeDistance($localKeyPoints[
        ↪ $possibleSpeedRange["first_key_point_index"]]["speed_0"], (
        ↪ $i + 10), $verzoegerung) + $localKeyPoints[
        ↪ $possibleSpeedRange["first_key_point_index"]]["position_0"
        ↪ ]);
344     $localKeyPoints[$possibleSpeedRange["first_key_point_index"] + 1]["
        ↪ position_0"] = ($localKeyPoints[$possibleSpeedRange["
        ↪ first_key_point_index"] + 1]["position_1"] -
        ↪ getBrakeDistance(($i + 10), $localKeyPoints[
        ↪ $possibleSpeedRange["first_key_point_index"] + 1]["speed_1"
        ↪ ], $verzoegerung));
345     $newKeyPoints = $localKeyPoints;
346     $finish = true;
347     $keyPoints = $localKeyPoints;
348
349     break;
350 }
351 if ($i == $possibleSpeedRange["min_speed"]) {
352     $newKeyPoints = $localKeyPoints;
353     $newKeyPoints = deleteDoubledKeyPoints($newKeyPoints);
354     $keyPoints = $newKeyPoints;
355     break;
356 }
357 $newKeyPoints = $localKeyPoints;
358 }

```

```

359     $keyPointsPreviousStep = $localKeyPoints;
360
361     if ($newKeyPoints != null) {
362         $keyPoints = $newKeyPoints;
363     }
364
365     $keyPoints = deleteDoubledKeyPoints($keyPoints);
366     $returnSpeedDecrease = checkIfTheSpeedCanBeDecreased();
367 }
368
369 $keyPoints = calculateTimeFromKeyPoints();
370
371 if ($useSpeedFineTuning && $returnSpeedDecrease["possible"]) {
372     $trainChangeReturn = createTrainChanges(true);
373     $trainPositionChange = $trainChangeReturn[0];
374     $trainSpeedChange = $trainChangeReturn[1];
375     $newCalculatedTime = $keyPoints[array_key_last($keyPoints)]["time_1"];
376     speedFineTuning(($maxTimeToNextStop - ($newCalculatedTime - $startTime
        ↳ )) , $returnSpeedDecrease["range"][array_key_last(
        ↳ $returnSpeedDecrease["range"])]["KeyPoint_index"]);
377 }
378
379 $keyPoints = calculateTimeFromKeyPoints();
380 $timeToNextStop = end($keyPoints)["time_1"] - $keyPoints[0]["time_0"];
381
382 echo "\nDurch die Anpassung der Geschwindigkeit benötigt der Zug mit der
    ↳ Adresse ", $train["adresse"], " jetzt ", number_format(
    ↳ $timeToNextStop, 2), " Sekunden bis\n";
383
384 if (abs($timeToNextStop - $maxTimeToNextStop) <
    ↳ $globalFloatingPointNumbersRoundingError) {
385     echo "zum nächsten planmäßigen Halt (", $targetBetriebsstelle, ") und
        ↳ wird diesen genau pünktlich erreichen.\n";
386 } else if (($timeToNextStop - $maxTimeToNextStop) > 0) {
387     echo "zum nächsten planmäßigen Halt (", $targetBetriebsstelle, ") und
        ↳ wird diesen mit einer Verspätung von ", number_format(
        ↳ $timeToNextStop - $maxTimeToNextStop, 2), " Sekunden erreichen
        ↳ .\n";
388 } else {

```

```

389         echo "zum nächsten planmäßigen Halt (" , $targetBetriebsstelle, ") und
           ↳ wird diesen " , number_format($timeToNextStop -
           ↳ $maxTimeToNextStop, 2), " Sekunden zu früh erreichen.\n";
390     }
391 } else {
392     echo "Dadurch, dass \$slowDownIfTooEarly = true ist, wird das Fahrzeug "
           ↳ , number_format($maxTimeToNextStop - $timeToNextStop, 2), "
           ↳ Sekunden zu früh am Ziel ankommen.";
393 }
394 }
395 } else {
396     echo "Der Zug mit der Adresse " , $train["adresse"], " fährt aktuell ohne
           ↳ Fahrplan bis zum nächsten auf Halt stehendem Signal (Signal ID: " ,
           ↳ $indexReachedBetriebsstelle, " , Betriebsstelle: " ,
           ↳ $targetBetriebsstelle, ").\n";
397 }
398
399 // Berechnung der Echtzeitdaten
400 $returnTrainChanges = createTrainChanges(false);
401 $trainPositionChange = $returnTrainChanges[0];
402 $trainSpeedChange = $returnTrainChanges[1];
403 $trainTimeChange = $returnTrainChanges[2];
404 $trainRelativePosition = $returnTrainChanges[3];
405 $trainSection = $returnTrainChanges[4];
406 $trainIsSpeedChange = $returnTrainChanges[5];
407 $trainTargetReached = array();
408 $trainBetriebsstelleName = array();
409 $trainWendet = array();
410 $allReachedTargets = array();
411 $allreachedInfrasIndex = array();
412 $allreachedInfrasID = array();
413 $allreachedInfrasUsed = array();
414
415 foreach ($allreachedInfras as $value) {
416     array_push($allreachedInfrasIndex, $value["index"]);
417     array_push($allreachedInfrasID, $value["infra"]);
418 }
419
420 foreach ($trainPositionChange as $key => $value) {

```



```

421     $trainBetriebsstelleName[$key] = $targetBetriebsstelle;
422     if (array_key_last($trainPositionChange) != $key) {
423         $trainTargetReached[$key] = false;
424         $trainWendet[$key] = false;
425     } else {
426         if ($wendet) {
427             $trainWendet[$key] = true;
428         } else {
429             $trainWendet[$key] = false;
430         }
431
432         if ($reachedBetriebsstelle) {
433             $trainTargetReached[$key] = true;
434         } else {
435             $trainTargetReached[$key] = false;
436         }
437     }
438 }
439
440 for($i = sizeof($trainSection) - 1; $i >= 0; $i--) {
441     if (in_array($trainSection[$i], $allreachedInfrasID) && !in_array(
442         ↪ $trainSection[$i], $allreachedInfrasUsed)) {
443         array_push($allreachedInfrasUsed, $trainSection[$i]);
444         $Infracindex = array_search($trainSection[$i], $allreachedInfrasID);
445         $allReachedTargets[$i] = $allreachedInfrasIndex[$Infracindex];
446     } else {
447         $allReachedTargets[$i] = null;
448     }
449 }
450 ksort($allReachedTargets);
451 $returnArray = array();
452 $address = $train["adresse"];
453 $trainID = array();
454 $id = $train["id"];
455
456 foreach ($trainPositionChange as $key => $value) {
457     $trainID[$key] = $id;
458 }

```

```

459 foreach ($trainPositionChange as $trainPositionChangeIndex =>
    ↳ $trainPositionChangeValue) {
460     array_push($returnArray, array("live_position" => $trainPositionChangeValue
        ↳ ,
461         "live_speed" => $trainSpeedChange[$trainPositionChangeIndex],
462         "live_time" => $trainTimeChange[$trainPositionChangeIndex],
463         "live_relative_position" => $trainRelativePosition[
            ↳ $trainPositionChangeIndex],
464         "live_section" => $trainSection[$trainPositionChangeIndex],
465         "live_is_speed_change" => $trainIsSpeedChange[$trainPositionChangeIndex],
466         "live_target_reached" => $trainTargetReached[$trainPositionChangeIndex],
467         "id" => $trainID[$trainPositionChangeIndex],
468         "wendet" => $trainWendet[$trainPositionChangeIndex],
469         "betriebsstelle" => $trainBetriebsstelleName[$trainPositionChangeIndex],
470         "live_all_targets_reached" => $allReachedTargets[
            ↳ $trainPositionChangeIndex]));
471 }
472
473 $allTimes[$adress] = $returnArray;
474 safeTrainChangeToJSONFile($indexCurrentSection, $indexTargetSection,
    ↳ $indexCurrentSectionMod, $indexTargetSectionMod,
    ↳ $speedOverPositionAllIterations);
475
476 return (end($trainTimeChange) - $trainTimeChange[0]) - ($endTime - $startTime
    ↳ );
477 }
478
479 // Ermittelt die maximale Geschwindigkeit zwischen zwei Punkten
480 function getVMaxBetweenTwoPoints(float $distance, int $v_0, int $v_1, int $id)
    ↳ {
481
482     global $verzoegerung;
483     global $globalFloatingPointNumbersRoundingError;
484
485     $v_max = array();
486
487     for ($i = 0; $i <= 120; $i = $i + 10) {
488         if ((getBrakeDistance($v_0, $i, $verzoegerung) + getBrakeDistance($i, $v_1,
            ↳ $verzoegerung)) < ($distance +

```

```

        ↪ $globalFloatingPointNumbersRoundingError)) {
489     array_push($v_max, $i);
490 }
491 }
492
493 if (sizeof($v_max) == 0) {
494     if ($v_0 == 0 && $v_1 == 0 && $distance > 0) {
495         echo "Der zug müsste langsamer als 10 km/h fahren, um das Ziel zu
        ↪ erreichen.";
496     } else {
497         emergencyBreak($id);
498     }
499 } else {
500     if ($v_0 == $v_1 && max($v_max) < $v_0) {
501         $v_max = array($v_0);
502     }
503 }
504
505 return max($v_max);
506 }
507
508 // Erstellt einen $keyPoint
509 function createKeyPoint (float $position_0, float $position_1, int $speed_0,
        ↪ int $speed_1) {
510     return array("position_0" => $position_0, "position_1" => $position_1, "
        ↪ speed_0" => $speed_0, "speed_1" => $speed_1);
511 }
512
513 // Ermittelt aus den $keyPoint die Echtzeitdaten (nur Geschwindigkeit und
        ↪ Position)
514 function convertKeyPointsToTrainChangeArray (array $keyPoints) {
515
516     global $verzoegerung;
517
518     $trainSpeedChangeReturn = array();
519     $trainPositionChnageReturn = array();
520     array_push($trainPositionChnageReturn, $keyPoints[0]["position_0"]);
521     array_push($trainSpeedChangeReturn, $keyPoints[0]["speed_0"]);
522

```

```

523 for ($i = 0; $i <= (sizeof($keyPoints) - 2); $i++) {
524     if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
525         for ($j = $keyPoints[$i]["speed_0"]; $j < $keyPoints[$i]["speed_1"]; $j =
            ↪ $j + 2) {
526             array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn)
            ↪ + getBrakeDistance($j, ($j + 2), $verzoegerung)));
527             array_push($trainSpeedChangeReturn, ($j + 2));
528         }
529     } elseif ($keyPoints[$i]["speed_0"] > $keyPoints[$i]["speed_1"]) {
530         for ($j = $keyPoints[$i]["speed_0"]; $j > $keyPoints[$i]["speed_1"]; $j =
            ↪ $j - 2) {
531             array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn)
            ↪ + getBrakeDistance($j, ($j - 2), $verzoegerung)));
532             array_push($trainSpeedChangeReturn, ($j - 2));
533         }
534     }
535     array_push($trainPositionChnageReturn, $keyPoints[$i + 1]["position_0"]);
536     array_push($trainSpeedChangeReturn, $keyPoints[$i + 1]["speed_0"]);
537 }
538
539 if (end($keyPoints)["speed_0"] < end($keyPoints)["speed_1"]) {
540     for ($j = end($keyPoints)["speed_0"]; $j < end($keyPoints)["speed_1"]; $j =
        ↪ $j + 2) {
541         array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
        ↪ getBrakeDistance($j, ($j + 2), $verzoegerung)));
542         array_push($trainSpeedChangeReturn, ($j + 2));
543     }
544 } else if (end($keyPoints)["speed_0"] > end($keyPoints)["speed_1"]) {
545     for ($j = end($keyPoints)["speed_0"]; $j > end($keyPoints)["speed_1"]; $j =
        ↪ $j - 2) {
546         array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
        ↪ getBrakeDistance($j, ($j - 2), $verzoegerung)));
547         array_push($trainSpeedChangeReturn, ($j - 2));
548     }
549 }
550
551 return array($trainPositionChnageReturn, $trainSpeedChangeReturn);
552 }
553

```

```

554 // Wandelt die Daten der Infra-Abschnitte und der Iterationsschritte der
555 // Fahrtverlaufsrechnung in JSON-Dateien um, damit die Fahrtverläufe
556 // visuell dargestellt werden können.
557 function safeTrainChangeToJSONFile(int $indexCurrentSection, int
    ↪ $indexTargetSection, int $indexCurrentSectionMod, int
    ↪ $indexTargetSectionMod, array $speedOverPositionAllIterations) {
558
559     global $trainPositionChange;
560     global $trainSpeedChange;
561     global $next_v_max;
562     global $cumulativeSectionLengthEnd;
563     global $next_v_max_mod;
564     global $cumulativeSectionLengthEndMod;
565
566     $speedOverPosition = array_map('toArr', $trainPositionChange,
    ↪ $trainSpeedChange);
567     $speedOverPosition = json_encode($speedOverPosition);
568     $fp = fopen('../json/speedOverPosition.json', 'w');
569     fwrite($fp, $speedOverPosition);
570     fclose($fp);
571
572     $v_maxFromUsedSections = array();
573
574     for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
575         array_push($v_maxFromUsedSections, $next_v_max[$i]);
576     }
577
578     $VMaxOverCumulativeSections = array_map('toArr', $cumulativeSectionLengthEnd,
    ↪ $v_maxFromUsedSections);
579     $VMaxOverPositionsJSon = json_encode($VMaxOverCumulativeSections);
580     $fp = fopen('../json/VMaxOverCumulativeSections.json', 'w');
581     fwrite($fp, $VMaxOverPositionsJSon);
582     fclose($fp);
583
584     $v_maxFromUsedSections = array();
585
586     for ($i = $indexCurrentSectionMod; $i <= $indexTargetSectionMod; $i++) {
587         array_push($v_maxFromUsedSections, $next_v_max_mod[$i]);
588     }

```

```

589
590 $VMaxOverCumulativeSectionsMod = array_map('toArr',
    ↪ $cumulativeSectionLengthEndMod, $v_maxFromUsedSections);
591 $VMaxOverPositionsJSoN = json_encode($VMaxOverCumulativeSectionsMod);
592 $fp = fopen('../json/VMaxOverCumulativeSectionsMod.json', 'w');
593 fwrite($fp, $VMaxOverPositionsJSoN);
594 fclose($fp);
595
596 $jsonReturn = array();
597
598 for ($i = 0; $i < sizeof($speedOverPositionAllIterations); $i++) {
599     $iteration = array_map('toArr', $speedOverPositionAllIterations[$i][0],
    ↪ ↪ $speedOverPositionAllIterations[$i][1]);
600     array_push($jsonReturn, $iteration);
601 }
602
603 $speedOverPosition = json_encode($jsonReturn);
604 $fp = fopen('../json/speedOverPosition_prevIterations.json', 'w');
605 fwrite($fp, $speedOverPosition);
606 fclose($fp);
607 }
608
609 // Überprüft, ob das Fahrzeug in Infra-Abschnitten die zulässige
610 // Höchstgeschwindigkeit überschreitet
611 function checkIfTrainIsToFastInCertainSections() {
612
613     global $trainPositionChange;
614     global $trainSpeedChange;
615     global $cumulativeSectionLengthStartMod;
616     global $next_v_max_mod;
617     global $indexTargetSectionMod;
618
619     $faillSections = array();
620
621     foreach ($trainPositionChange as $trainPositionChangeKey =>
    ↪ ↪ $trainPositionChangeValue) {
622         foreach ($cumulativeSectionLengthStartMod as
    ↪ ↪ $cumulativeSectionLengthStartKey =>
    ↪ ↪ $cumulativeSectionLengthStartValue) {

```

```

623     if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
624         if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
            ↳ $cumulativeSectionLengthStartKey - 1]) {
625             array_push($faieldSections, ($cumulativeSectionLengthStartKey - 1));
626         }
627
628         break;
629     } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
630         if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
631             if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
                ↳ $cumulativeSectionLengthStartKey]) {
632                 array_push($faieldSections, $cumulativeSectionLengthStartKey);
633             }
634
635             break;
636         }
637     }
638 }
639 }
640
641 if (sizeof($faieldSections) == 0) {
642     return array("failed" => false);
643 } else {
644     return array("failed" => true, "failed_sections" => array_unique(
        ↳ $faieldSections));
645 }
646 }
647
648 // Löscht $keyPoint, bei denen Start- und Zielgeschwindigkeit identisch ist.
649 // Der erste $keyPoint wird dabei nicht betrachtet.
650 function deleteDoubledKeyPoints($temporaryKeyPoints) {
651     do {
652         $foundDoubledKeyPoints = false;
653         $doubledIndex = array();
654
655         for ($i = 1; $i < (sizeof($temporaryKeyPoints) - 1); $i++) {
656             if ($temporaryKeyPoints[$i]["speed_0"] == $temporaryKeyPoints[$i][
                ↳ speed_1"]) {
657                 $foundDoubledKeyPoints = true;

```

```

658     array_push($doubledIndex, $i);
659 }
660 }
661
662 foreach ($doubledIndex as $index) {
663     unset($temporaryKeyPoints[$index]);
664 }
665
666 $temporaryKeyPoints = array_values($temporaryKeyPoints);
667 } while ($foundDoubledKeyPoints);
668
669 return $temporaryKeyPoints;
670 }
671
672 // Ermittelt die Zeiten der $keyPoint ausgehend vom ersten $keyPoint. Mit dem
673 // Parameter $inputKeyPoints können $keyPoints übergeben werden, bei den die
674 // Zeit ermittelt wird. Wenn keine $keyPoints übergeben werden, werden die
675 // globalen $keyPoints verwendet. Mit dem Parameter $skippingKeys können
676 // $keyPoints übersprungen werden. Das auslassen von $keyPoints ist für die
677 // Funktion postponeSubsection() relevant.
678 function calculateTimeFromKeyPoints($inputKeyPoints = null, $skippingKeys =
    ↪ null) {
679
680     global $keyPoints;
681     global $verzoegerung;
682
683     if ($inputKeyPoints == null) {
684         $localKeyPoints = $keyPoints;
685     } else {
686         $localKeyPoints = $inputKeyPoints;
687     }
688
689     $keys = array_keys($localKeyPoints);
690
691     if ($skippingKeys != null) {
692         foreach ($skippingKeys as $skip) {
693             unset($keys[array_search($skip, $keys)]);
694         }
695     }

```



```

696
697 $keys = array_values($keys);
698
699 for ($i = 0; $i < (sizeof($keys) - 1); $i++) {
700     $localKeyPoints[$keys[$i]]["time_1"] = getBrakeTime($localKeyPoints[$keys[
        ↳ $i]]["speed_0"], $localKeyPoints[$keys[$i]]["speed_1"],
        ↳ $verzoeigerung) + $localKeyPoints[$keys[$i]]["time_0"];
701     $localKeyPoints[$keys[$i] + 1]["time_0"] = distanceWithSpeedToTime(
        ↳ $localKeyPoints[$keys[$i]]["speed_1"], ($localKeyPoints[$keys[$i] +
        ↳ 1]["position_0"]) - $localKeyPoints[$keys[$i]]["position_1"]) +
        ↳ $localKeyPoints[$keys[$i]]["time_1"];
702 }
703
704 $localKeyPoints[end($keys)]["time_1"] = getBrakeTime($localKeyPoints[end(
        ↳ $keys)]["speed_0"], $localKeyPoints[end($keys)]["speed_1"],
        ↳ $verzoeigerung) + $localKeyPoints[end($keys)]["time_0"];
705
706 return $localKeyPoints;
707 }
708
709 // Echtzeitdatenermittlung eines Fahrtverlaufs auf Grundlage der $keyPoints.
710 // Mit dem Parameter $onlyPositionAndSpeed kann festgelegt werden, ob nur die
711 // Position und Geschwindigkeit berechnet werden soll.
712 function createTrainChanges(bool $onlyPositionAndSpeed) {
713
714     global $keyPoints;
715     global $verzoeigerung;
716     global $cumulativeSectionLengthStart;
717     global $cumulativeSectionLengthEnd;
718     global $next_sections;
719     global $indexCurrentSection;
720     global $indexTargetSection;
721     global $currentPosition;
722     global $globalFloatingPointNumbersRoundingError;
723     global $globalTimeUpdateInterval;
724
725     $returnTrainSpeedChange = array();
726     $returnTrainTimeChange = array();
727     $returnTrainPositionChange = array();

```

```

728 $returnTrainRelativePosition = array();
729 $returnTrainSection = array();
730 $returnIsSpeedChange = array();
731
732 // Ermittelt für alle bis auf den letzten $keyPoint die Echtzeitdaten der
733 // Zeit, Geschwindigkeit und Position
734 for ($i = 0; $i < (sizeof($keyPoints) - 1); $i++) {
735     array_push($returnTrainTimeChange, $keyPoints[$i]["time_0"]);
736     array_push($returnTrainSpeedChange, $keyPoints[$i]["speed_0"]);
737     array_push($returnTrainPositionChange, $keyPoints[$i]["position_0"]);
738     array_push($returnIsSpeedChange, true);
739     if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
740         for ($j = ($keyPoints[$i]["speed_0"] + 2); $j <= $keyPoints[$i]["speed_1"]
741             ↪ ]; $j = $j + 2) {
742             array_push($returnTrainPositionChange, (end($returnTrainPositionChange)
743                 ↪ + getBrakeDistance(($j - 2), $j, $verzögerung)));
744             array_push($returnTrainSpeedChange, $j);
745             array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
746                 ↪ getBrakeTime(($j - 2), $j, $verzögerung))));
747             array_push($returnIsSpeedChange, true);
748         }
749     } else {
750         for ($j = ($keyPoints[$i]["speed_0"] - 2); $j >= $keyPoints[$i]["speed_1"]
751             ↪ ]; $j = $j - 2) {
752             array_push($returnTrainPositionChange, (end($returnTrainPositionChange)
753                 ↪ + getBrakeDistance(($j + 2), $j, $verzögerung)));
754             array_push($returnTrainSpeedChange, $j);
755             array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
756                 ↪ getBrakeTime(($j + 2), $j, $verzögerung))));
757             array_push($returnIsSpeedChange, true);
758         }
759     }
760 }
761
762 // Ermittelt für die Strecke zwischen zwei $keyPoints die Echtzeitdaten
763 // der Zeit, Geschwindigkeit und Position
764 $startPosition = $keyPoints[$i]["position_1"];
765 $endPosition = $keyPoints[$i + 1]["position_0"];
766 $speedToNextKeyPoint = $keyPoints[$i]["speed_1"];
767 $distanceForOneTimeInterval = $speedToNextKeyPoint / 3.6;

```

```

761
762     for ($position = $startPosition + $distanceForOneTimeInterval; $position <
        ↪ $endPosition; $position = $position + $distanceForOneTimeInterval) {
763         array_push($returnTrainPositionChange, $position);
764         array_push($returnTrainSpeedChange, $speedToNextKeyPoint);
765         array_push($returnTrainTimeChange, end($returnTrainTimeChange) +
        ↪ $globalTimeUpdateInterval);
766         array_push($returnIsSpeedChange, false);
767     }
768 }
769 array_push($returnTrainPositionChange, $keyPoints[array_key_last($keyPoints)
        ↪ ][ "position_1" ] - getBrakeDistance($keyPoints[array_key_last(
        ↪ $keyPoints)][ "speed_0" ], $keyPoints[array_key_last($keyPoints)][ "
        ↪ speed_1" ], $verzoeigerung));
770 array_push($returnTrainSpeedChange, $keyPoints[array_key_last($keyPoints)][ "
        ↪ speed_0" ]);
771 array_push($returnTrainTimeChange, $keyPoints[array_key_last($keyPoints)][ "
        ↪ time_0" ]);
772 array_push($returnIsSpeedChange, true);
773
774 // Ermittelt für den letzten $keyPoint die Echtzeitdaten der Zeit,
775 // Geschwindigkeit und Position
776 if ($keyPoints[array_key_last($keyPoints)][ "speed_0" ] < $keyPoints[
        ↪ array_key_last($keyPoints)][ "speed_1" ]) {
777     for ($j = ($keyPoints[array_key_last($keyPoints)][ "speed_0" ] + 2); $j <=
        ↪ $keyPoints[array_key_last($keyPoints)][ "speed_1" ]; $j = $j + 2) {
778         array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
        ↪ getBrakeDistance(($j - 2), $j, $verzoeigerung)));
779         array_push($returnTrainSpeedChange, $j);
780         array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
        ↪ getBrakeTime(($j - 2), $j, $verzoeigerung))));
781         array_push($returnIsSpeedChange, true);
782     }
783 } else {
784     for ($j = ($keyPoints[array_key_last($keyPoints)][ "speed_0" ] - 2); $j >=
        ↪ $keyPoints[array_key_last($keyPoints)][ "speed_1" ]; $j = $j - 2) {
785         array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
        ↪ getBrakeDistance(($j + 2), $j, $verzoeigerung)));
786         array_push($returnTrainSpeedChange, $j);

```

```

787     array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
788         ↳ getBrakeTime(($j + 2), $j, $verzoeigerung))));
789 }
790 }
791
792 if ($onlyPositionAndSpeed) {
793     return array($returnTrainPositionChange, $returnTrainSpeedChange);
794 } else {
795     // Ermittelt die relativen Positionen innerhalb der Infra-Abschnitte
796     // zu den absoluten Positionen
797     foreach ($returnTrainPositionChange as $absolutPositionKey =>
798         ↳ $absolutPositionValue) {
799         foreach ($cumulativeSectionLengthStart as $sectionStartKey =>
800             ↳ $sectionStartValue) {
801             if ($absolutPositionValue >= $sectionStartValue && $absolutPositionValue
802                 ↳ < $cumulativeSectionLengthEnd[$sectionStartKey]) {
803                 if ($sectionStartKey == $indexCurrentSection && $sectionStartKey ==
804                     ↳ $indexTargetSection) {
805                     $returnTrainRelativePosition[$absolutPositionKey] =
806                         ↳ $absolutPositionValue + $currentPosition;
807                     $returnTrainSection[$absolutPositionKey] = $next_sections[
808                         ↳ $sectionStartKey];
809                 } else if ($sectionStartKey == $indexCurrentSection) {
810                     $returnTrainRelativePosition[$absolutPositionKey] =
811                         ↳ $absolutPositionValue + $currentPosition;
812                     $returnTrainSection[$absolutPositionKey] = $next_sections[
813                         ↳ $sectionStartKey];
814                 } else if ($sectionStartKey == $indexTargetSection) {
815                     $returnTrainRelativePosition[$absolutPositionKey] =
816                         ↳ $absolutPositionValue - $sectionStartValue;
817                     $returnTrainSection[$absolutPositionKey] = $next_sections[
818                         ↳ $sectionStartKey];
819                 } else {
820                     $returnTrainRelativePosition[$absolutPositionKey] =
821                         ↳ $absolutPositionValue - $sectionStartValue;
822                     $returnTrainSection[$absolutPositionKey] = $next_sections[
823                         ↳ $sectionStartKey];
824                 }
825             }
826         }
827     }
828 }

```

```

813         break;
814     } else if ($absolutPositionKey == array_key_last(
        ↳ $returnTrainPositionChange) && abs($absolutPositionValue -
        ↳ floatval($cumulativeSectionLengthEnd[$sectionStartKey])) <
        ↳ $globalFloatingPointNumbersRoundingError) {
815         $returnTrainRelativePosition[$absolutPositionKey] =
            ↳ $cumulativeSectionLengthEnd[$sectionStartKey] -
            ↳ $sectionStartValue;
816         $returnTrainSection[$absolutPositionKey] = $next_sections[
            ↳ $sectionStartKey];
817         break;
818     } else {
819         debugMessage("Eine absolute Position konnte kein Infra-Abschnitt und
            ↳ keine relative Position in einem Infra-Abschnitt zugeordnet
            ↳ werden.");
820     }
821 }
822 }
823
824 return array($returnTrainPositionChange, $returnTrainSpeedChange,
    ↳ $returnTrainTimeChange, $returnTrainRelativePosition,
    ↳ $returnTrainSection, $returnIsSpeedChange);
825 }
826 }
827
828 // Überprüft, ob es zwischen zwei benachbarten $keyPoints zu einer
    ↳ Geschwindigkeits-
829 // überschreitung kommt.
830 function recalculateKeyPoints(array $tempKeyPoints, int $id) {
831
832     $returnKeyPoints = array();
833     $numberOfPairs = sizeof($tempKeyPoints) / 2;
834
835     for($j = 0; $j < $numberOfPairs; $j++) {
836         $i = $j * 2;
837         $return = checkBetweenTwoKeyPoints($tempKeyPoints, $i, $id);
838
839         foreach ($return as $keyPoint) {
840             array_push($returnKeyPoints, $keyPoint);

```

```

841     }
842 }
843
844 return $returnKeyPoints;
845 }
846
847 // Überprüft, ob zwischen zwei $keyPoints die zulässige Höchstgeschwindigkeit
848 // überschritten wird
849 function checkBetweenTwoKeyPoints(array $temKeyPoints, int $keyPointIndex, int
    ↪ $id) {
850
851     global $trainPositionChange;
852     global $trainSpeedChange;
853     global $cumulativeSectionLengthStartMod;
854     global $cumulativeSectionLengthEndMod;
855     global $next_v_max_mod;
856     global $verzoegerung;
857     global $indexTargetSectionMod;
858
859     $failedSections = array();
860     $groupedFailedSections = array();
861     $returnKeyPoints = array();
862     $failedPositions = array();
863     $failedSpeeds = array();
864
865     foreach ($trainPositionChange as $trainPositionChangeKey =>
    ↪ $trainPositionChangeValue) {
866         if ($trainPositionChangeValue >= $temKeyPoints[$keyPointIndex]["position_0"
    ↪ ] && $trainPositionChangeValue <= $temKeyPoints[$keyPointIndex + 1][
    ↪ "position_1"]) {
867             foreach ($cumulativeSectionLengthStartMod as
    ↪ $cumulativeSectionLengthStartKey =>
    ↪ $cumulativeSectionLengthStartValue) {
868                 if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
869                     if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
    ↪ $cumulativeSectionLengthStartKey - 1]) {
870                         array_push($failedSections, ($cumulativeSectionLengthStartKey - 1));
871                         array_push($failedSpeeds, $trainSpeedChange[$trainPositionChangeKey
    ↪ ]);

```

```

872         $failedPositions[$trainPositionChangeKey] = $trainPositionChange[
            ↳ $trainPositionChangeKey];
873     }
874     break;
875 } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
876     if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
877         if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
            ↳ $cumulativeSectionLengthStartKey]) {
878             array_push($failedSections, $cumulativeSectionLengthStartKey);
879             array_push($failedSpeeds, $trainSpeedChange[
                ↳ $trainPositionChangeKey]);
880             $failedPositions[$trainPositionChangeKey] = $trainPositionChange[
                ↳ $trainPositionChangeKey];
881         }
882         break;
883     }
884 }
885 }
886 }
887 }
888
889 // Alle Infra_abschnitte zwischen denn beiden KeyPoints, bei denen die
890 // zulässige Höchstgeschwindigkeit überschritten wird
891 $failedSections = array_unique($failedSections);
892
893 // Wenn es kein Fehler gibt, werden die beiden KeyPoints zurückgegeben und
894 // wen es einen Fehler gibt, wird der erste der beiden KeyPoints im
895 // $returnKeyPoints gespeichert
896 if (sizeof($failedSections) == 0) {
897     return array($temKeyPoints[$keyPointIndex], $temKeyPoints[$keyPointIndex +
        ↳ 1]);
898 } else {
899     $returnKeyPoints[0]["speed_0"] = $temKeyPoints[$keyPointIndex]["speed_0"];
900     $returnKeyPoints[0]["position_0"] = $temKeyPoints[$keyPointIndex]["
        ↳ position_0"];
901 }
902
903 // Einteilung der benachbarten failedSections in zusammenhängende Gruppen
904 $previous = NULL;

```

```

905 $index = 0;
906 foreach($failedSections as $key => $value) {
907     if($value > $previous + 1) {
908         $index++;
909     }
910     $groupedFailedSections[$index][] = $value;
911     $previous = $value;
912 }
913
914 // Iteration über die zusammenhängenden $failedSections
915 foreach ($groupedFailedSections as $groupSectionsIndex => $groupSectionsValue
    ↪ ) {
916     $firstFailedPositionIndex = null;
917     $lastFailedPositionIndex = null;
918     $firstFailedPosition = null;
919     $lastFailedPosition = null;
920     $lastElement = array_key_last($returnKeyPoints);
921     $failedSection = null;
922
923     // Ermittlung der Section mit der kleinsten v_max von allen $failedSections
924     // in der Gruppe
925     if (sizeof($groupSectionsValue) == 1) {
926         $failedSection = $groupSectionsValue[0];
927     } else {
928         $slowestSpeed = 200;
929         for ($i = 0; $i <= (sizeof($groupSectionsValue) - 1); $i++) {
930             if ($next_v_max_mod[$groupSectionsValue[$i]] < $slowestSpeed) {
931                 $slowestSpeed = $next_v_max_mod[$groupSectionsValue[$i]];
932                 $failedSection = $groupSectionsValue[$i];
933             }
934         }
935     }
936
937     // Start- und Endposition der $failedSection
938     $failedSectionStart = $cumulativeSectionLengthStartMod[$failedSection];
939     $failedSectionEnd = $cumulativeSectionLengthEndMod[$failedSection];
940
941     // Bestimmung der ersten und letzten Position, in der es in der
    ↪ $failedSection

```



```

942 // zu einer Geschwindigkeitsüberschreitung kommt
943 foreach ($failedPositions as $failPositionIndex => $failPositionValue) {
944     if ($failPositionValue > $failedSectionStart && $failPositionValue <
        ↳ $failedSectionEnd) {
945         if ($firstFailedPositionIndex == null) {
946             $firstFailedPositionIndex = $failPositionIndex;
947         }
948         $lastFailedPositionIndex = $failPositionIndex;
949     }
950 }
951
952 // Bestimmung des letzten Punktes, bei dem die Geschwindigkeit noch
953 // nicht zu schnell war
954 //
955 // Wenn der Punkt davor außerhalb der failedSection liegt
956 // => Startpunkt = Anfang der Section
957 // Wenn der Punkt davor innerhalb der failed Section liegt
958 // => Startpunkt = der Punkt davor
959 if ($firstFailedPositionIndex != 0) {
960     if ($trainPositionChange[$firstFailedPositionIndex - 1] <
        ↳ $failedSectionStart) {
961         $firstFailedPosition = $failedSectionStart;
962     } else {
963         $firstFailedPosition = $trainPositionChange[$firstFailedPositionIndex -
            ↳ 1];
964     }
965 } else {
966     $firstFailedPosition = $failedSectionStart;
967 }
968
969 // Bestimmung der ersten Position, bei dem die Geschwindigkeit nicht
970 // mehr zu hoch war.
971 if ($lastFailedPositionIndex != array_key_last($trainPositionChange)) {
972     if ($trainPositionChange[$lastFailedPositionIndex + 1] >
        ↳ $failedSectionEnd) {
973         $lastFailedPosition = $failedSectionEnd;
974     } else {
975         $lastFailedPosition = $trainPositionChange[$lastFailedPositionIndex +
            ↳ 1];

```

```

976     }
977 } else {
978     $lastFailedPosition = $failedSectionEnd;
979 }
980
981 $returnKeyPoints[$lastElement + 1]["position_1"] = $firstFailedPosition;
982 $returnKeyPoints[$lastElement + 1]["speed_1"] = $next_v_max_mod[
    ↪ $failedSection];
983 $returnKeyPoints[$lastElement + 2]["position_0"] = $lastFailedPosition;
984 $returnKeyPoints[$lastElement + 2]["speed_0"] = $next_v_max_mod[
    ↪ $failedSection];
985 }
986
987 // Zielwerte des letzten $keyPoint vom zweiten $keyPoint übernehmen
988 $returnKeyPoints[array_key_last($returnKeyPoints) + 1]["position_1"] =
    ↪ $temKeyPoints[$keyPointIndex + 1]["position_1"];
989 $returnKeyPoints[array_key_last($returnKeyPoints)]["speed_1"] = $temKeyPoints
    ↪ [$keyPointIndex + 1]["speed_1"];
990 $numberOfPairs = sizeof($returnKeyPoints) / 2;
991
992 for($j = 0; $j < $numberOfPairs; $j++) {
993     $i = $j * 2;
994     $distance = $returnKeyPoints[$i + 1]["position_1"] - $returnKeyPoints[$i]["
    ↪ position_0"];
995     $vMax = getVMaxBetweenTwoPoints($distance, $returnKeyPoints[$i]["speed_0"],
    ↪ $returnKeyPoints[$i + 1]["speed_1"], $id);
996     $returnKeyPoints[$i]["speed_1"] = $vMax;
997     $returnKeyPoints[$i]["position_1"] = $returnKeyPoints[$i]["position_0"] +
    ↪ getBrakeDistance($returnKeyPoints[$i]["speed_0"], $vMax,
    ↪ $verzoeigerung);
998     $returnKeyPoints[$i + 1]["speed_0"] = $vMax;
999     $returnKeyPoints[$i + 1]["position_0"] = $returnKeyPoints[$i + 1]["
    ↪ position_1"] - getBrakeDistance($vMax, $returnKeyPoints[$i + 1]["
    ↪ speed_1"], $verzoeigerung);
1000 }
1001
1002 return $returnKeyPoints;
1003 }
1004

```

```

1005 // Wenn ein Key Point beschleunigt und der nächste Key Point abbremst, wird
1006 // die Geschwindigkeit zwischen den beiden KeyPoints als $v_maxBetweenKeyPoints
1007 // gespeichert und als $v_minBetweenKeyPoints der größere Wert von
1008 // $keyPoints[$i]["speed_0"] und $keyPoints[$i + 1]["speed_1"]
1009 function checkIfTheSpeedCanBeDecreased() {
1010
1011     global $keyPoints;
1012     global $returnPossibleSpeed;
1013
1014     $returnPossibleSpeed = array();
1015
1016     for ($i = 0; $i < (sizeof($keyPoints) - 1); $i++) {
1017         $v_maxBetweenKeyPoints = $keyPoints[$i]["speed_1"];
1018         $v_minBetweenKeyPoints = null;
1019
1020         if ($keyPoints[$i]["speed_0"] < $v_maxBetweenKeyPoints && $keyPoints[$i +
            ↳ 1]["speed_1"] < $v_maxBetweenKeyPoints) {
1021             $v_minBetweenKeyPoints = $keyPoints[$i]["speed_0"];
1022             if ($keyPoints[$i + 1]["speed_1"] > $v_minBetweenKeyPoints) {
1023                 $v_minBetweenKeyPoints = $keyPoints[$i + 1]["speed_1"];
1024             }
1025         }
1026
1027         if (isset($v_minBetweenKeyPoints)) {
1028             if ($v_minBetweenKeyPoints == 0 && $v_maxBetweenKeyPoints >= 10) {
1029                 $v_minBetweenKeyPoints = 10;
1030             } else if ($v_minBetweenKeyPoints == 0 && $v_maxBetweenKeyPoints == 10) {
1031                 $v_minBetweenKeyPoints = null;
1032             }
1033         }
1034
1035         if ($v_minBetweenKeyPoints != null) {
1036             if ($v_minBetweenKeyPoints % 10 != 0) {
1037                 $rest = $v_minBetweenKeyPoints % 10;
1038                 $v_minBetweenKeyPoints = $v_minBetweenKeyPoints - $rest + 10;
1039             }
1040
1041             array_push($returnPossibleSpeed, array("KeyPoint_index" => $i, "values"
            ↳ => range($v_minBetweenKeyPoints, $v_maxBetweenKeyPoints, 10)));

```

```

1042     }
1043 }
1044 if (sizeof($returnPossibleSpeed) > 0) {
1045     return array("possible" => true, "range" => $returnPossibleSpeed);
1046 } else {
1047     return array("possible" => false, "range" => array());
1048 }
1049 }
1050
1051 // Wenn in 'global_variables.php' der Variablen $useSpeedFineTuning der Wert
1052 // 'true' zugewiesen ist und das Fahrzeug zu früh an der nächsten
1053     ↳ Betriebsstelle
1054 // ankommt, wird überprüft, ob durch eine vorzeitige Einleitung einer Verzö
1055     ↳ gerung
1056 // die exakte Ankunftszeit eingehalten werden kann.
1057 function speedFineTuning(float $timeDiff, int $index) {
1058
1059     global $keyPoints;
1060     global $verzoegerung;
1061     global $globalTimeOnOneSpeed;
1062     global $useMinTimeOnSpeed;
1063
1064     $speed_0 = $keyPoints[$index]["speed_1"];
1065     $speed_1 = null;
1066     $availableDistance = $keyPoints[$index + 1]["position_0"] - $keyPoints[$index
1067         ↳ ]["position_1"];
1068     $timeBetweenKeyPoints = $keyPoints[$index + 1]["time_0"] - $keyPoints[$index
1069         ↳ ]["time_1"];
1070     $availableTime = $timeBetweenKeyPoints + $timeDiff;
1071
1072     if ($keyPoints[$index + 1]["speed_1"] != 0) {
1073         $speed_1 = $keyPoints[$index + 1]["speed_1"];
1074         $lengthDifference = calculateDistanceforSpeedFineTuning($keyPoints[$index +
1075             ↳ 1]["speed_0"], $keyPoints[$index + 1]["speed_1"],
1076             ↳ $availableDistance, $availableTime);
1077
1078         if ($useMinTimeOnSpeed) {
1079             if (distanceWithSpeedToTime($speed_0, $availableDistance -
1080                 ↳ $lengthDifference) > $globalTimeOnOneSpeed &&

```

```

1074     ↪ distanceWithSpeedToTime($speed_1, $lengthDifference) >
1075     ↪ $globalTimeOnOneSpeed) {
1076     $keyPoints[$index + 1]["position_0"] = $keyPoints[$index + 1]["
1077         ↪ position_0"] - $lengthDifference;
1078     $keyPoints[$index + 1]["position_1"] = $keyPoints[$index + 1]["
1079         ↪ position_1"] - $lengthDifference;
1080 }
1081 } else {
1082     $keyPoints[$index + 1]["position_0"] = $keyPoints[$index + 1]["position_0
1083         ↪ "] - $lengthDifference;
1084     $keyPoints[$index + 1]["position_1"] = $keyPoints[$index + 1]["position_1
1085         ↪ "] - $lengthDifference;
1086 }
1087 } else if ($keyPoints[$index + 1]["speed_0"] > 10) {
1088     $speed_1 = 10;
1089     $lengthDifference = calculateDistanceforSpeedFineTuning($keyPoints[$index +
1090         ↪ 1]["speed_0"],10, $availableDistance, $availableTime);
1091
1092     if ($useMinTimeOnSpeed) {
1093         if (distanceWithSpeedToTime($speed_0, $availableDistance -
1094             ↪ $lengthDifference) > $globalTimeOnOneSpeed &&
1095             ↪ distanceWithSpeedToTime($speed_1, $lengthDifference) >
1096             ↪ $globalTimeOnOneSpeed) {
1097             $firstKeyPoint = createKeyPoint(($keyPoints[$index + 1]["position_0"] -
1098                 ↪ $lengthDifference),($keyPoints[$index + 1]["position_0"] -
1099                 ↪ $lengthDifference + getBrakeDistance($keyPoints[$index + 1]["
1100                 ↪ speed_0"],10, $verzoegerung)), $keyPoints[$index + 1]["speed_0"
1101                 ↪ ],10);
1102             $secondKeyPoint = createKeyPoint(($keyPoints[$index + 1]["position_1"] -
1103                 ↪ getBrakeDistance(10, 0, $verzoegerung)), $keyPoints[$index + 1]["
1104                 ↪ position_1"],10,$keyPoints[$index + 1]["speed_1"]);
1105             $keyPoints[$index + 1] = $secondKeyPoint;
1106             array_splice( $keyPoints, ($index + 1), 0, array($firstKeyPoint));
1107         }
1108     } else {
1109         $firstKeyPoint = createKeyPoint(($keyPoints[$index + 1]["position_0"] -
1110             ↪ $lengthDifference),($keyPoints[$index + 1]["position_0"] -
1111             ↪ $lengthDifference + getBrakeDistance($keyPoints[$index + 1]["
1112             ↪ speed_0"],10, $verzoegerung)), $keyPoints[$index + 1]["speed_0"],10)

```

```

1094     ↪ ;
1095     $secondKeyPoint = createKeyPoint(($keyPoints[$index + 1]["position_1"] -
1096     ↪ getBrakeDistance(10, 0, $verzoegerung)), $keyPoints[$index + 1]["
1097     ↪ position_1"], 10, $keyPoints[$index + 1]["speed_1"]);
1098     $keyPoints[$index + 1] = $secondKeyPoint;
1099     array_splice( $keyPoints, ($index + 1), 0, array($firstKeyPoint));
1100 }
1101 }
1102 }
1103 // Sucht den KeyPoint der zu maximalen Geschwindigkeit beschleunigt
1104 // Wenn die maximale Geschwindigkeit mehrfach erreicht wird, wird
1105 // der letzte dieser KeyPoints genommen
1106 //
1107 // Zu dem Index wird auch die Speed Range abgespeichert wie bei
1108 // checkIfTheSpeedCanBeDecreased()
1109 function findMaxSpeed(array $speedDecrease) {
1110     $maxSpeed = 0;
1111     $minSpeed = 0;
1112     $keyPointIndex = null;
1113
1114     for ($i = 0; $i < sizeof($speedDecrease["range"]); $i++) {
1115         if (max($speedDecrease["range"][$i]["values"]) >= $maxSpeed) {
1116             $maxSpeed = max($speedDecrease["range"][$i]["values"]);
1117             $minSpeed = min($speedDecrease["range"][$i]["values"]);
1118             $keyPointIndex = $speedDecrease["range"][$i]["KeyPoint_index"];
1119         }
1120     }
1121     return array("min_speed" => $minSpeed, "max_speed" => $maxSpeed, "
1122     ↪ first_key_point_index" => $keyPointIndex);
1123 }
1124
1125 // Überprüft beim Start der Fahrtverlaufsrechnung, ob es möglich ist einen
1126 ↪ Fahrtverlauf zu ermitteln
1127 function checkIfItsPossible(int $id) {
1128     global $currentSpeed;
1129     global $distanceToNextStop;
1130     global $verzoegerung;

```

```

1128 global $globalTimeOnOneSpeed;
1129 global $errorMinTimeOnSpeed;
1130
1131 $minTimeIsPossible = true;
1132
1133 if ($currentSpeed == 0) {
1134     $distance_0 = getBrakeDistance(0, 10, $verzoegerung);
1135     $distance_1 = getBrakeDistance(10, 0, $verzoegerung);
1136     $time = distanceWithSpeedToTime(10, $distanceToNextStop - $distance_0 -
        ↳ $distance_1);
1137
1138     if ($time < $globalTimeOnOneSpeed) {
1139         $minTimeIsPossible = false;
1140
1141         if ($errorMinTimeOnSpeed) {
1142             emergencyBreak($id);
1143         }
1144
1145         echo "Der Zug schafft es ohne Notbremsung am Ziel anzukommen, kann aber
        ↳ nicht die mind. Zeit einhalten.\n";
1146     }
1147 } else {
1148     if (getBrakeDistance($currentSpeed, 0, $verzoegerung) !=
        ↳ $distanceToNextStop) {
1149         $distance_0 = getBrakeDistance($currentSpeed, 10, $verzoegerung);
1150         $distance_1 = getBrakeDistance(10, 0, $verzoegerung);
1151         $time = distanceWithSpeedToTime(10, $distanceToNextStop - $distance_0 -
        ↳ $distance_1);
1152
1153         if ($time < $globalTimeOnOneSpeed) {
1154             $minTimeIsPossible = false;
1155
1156             if ($errorMinTimeOnSpeed) {
1157                 emergencyBreak($id);
1158             }
1159
1160             echo "Der Zug schafft es, ohne Notbremsung am Ziel anzukommen.\n";
1161         }
1162     }

```

```

1163 }
1164 return $minTimeIsPossible;
1165 }
1166
1167 // Überprüft, ob der vorgeschriebene Wert aus der Variablen
1168     ↳ $globalTimeOnOneSpeed
1169 // eingehalten wird, falls die Variable $useMinTimeOnSpeed den Wert 'true' hat
1170 function toShortOnOneSpeed () {
1171
1172     global $keyPoints;
1173     global $verzoeigerung;
1174
1175     $localKeyPoints = $keyPoints;
1176     $subsections = createSubsections($localKeyPoints);
1177
1178     while (toShortInSubsection($subsections)) {
1179         $breakesOnly = true;
1180         foreach ($subsections as $sectionKey => $sectionValue) {
1181             if ($sectionValue["failed"]) {
1182                 if (!$sectionValue["brakes_only"]) {
1183                     $breakesOnly = false;
1184                 }
1185
1186                 $return = postponeSubsection($localKeyPoints, $sectionValue);
1187
1188                 if (!$return["fail"]) {
1189                     $localKeyPoints = $return["keyPoints"];
1190                 } else {
1191                     if (!$sectionValue["brakes_only"]) {
1192                         $localKeyPoints[$sectionValue["max_index"]]["speed_1"] -= 10;
1193                         $localKeyPoints[$sectionValue["max_index"] + 1]["speed_0"] -= 10;
1194                         $localKeyPoints[$sectionValue["max_index"]]["position_1"] =
1195                             ↳ $localKeyPoints[$sectionValue["max_index"]]["position_0"] +
1196                             ↳ getBrakeDistance($localKeyPoints[$sectionValue["max_index"]][
1197                                 ↳ "speed_0"], $localKeyPoints[$sectionValue["max_index"]][
1198                                 ↳ "speed_1"], $verzoeigerung);
1199                         $localKeyPoints[$sectionValue["max_index"] + 1]["position_0"] =
1200                             ↳ $localKeyPoints[$sectionValue["max_index"] + 1]["position_1"]
1201                             ↳ - getBrakeDistance($localKeyPoints[$sectionValue["max_index"]

```



```

1195         ↪ ] + 1]["speed_0"], $localKeyPoints[$sectionValue["max_index"]
1196         ↪ + 1]["speed_1"], $verzoegerung);
1197     $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1198     $localKeyPoints = deleteDoubledKeyPoints($localKeyPoints);
1199     break;
1200 }
1201 }
1202 $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1203 $localKeyPoints = array_values($localKeyPoints);
1204 $subsections = createSubsections($localKeyPoints);
1205
1206 if ($breakesOnly) {
1207     break;
1208 }
1209 }
1210 $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1211 $keyPoints = $localKeyPoints;
1212 }
1213
1214 // Überprüft, ob innerhalb einer $subsection Beschleunigungs- und Bremsvorgänge
1215     ↪ später bzw. früher eingeleitet werden können.
1216 function postponeSubsection (array $localKeyPoints, array $subsection) {
1217
1218     global $globalTimeOnOneSpeed;
1219     global $verzoegerung;
1220
1221     $deletedKeyPoints = array();
1222     $numberOfKeyPoints = sizeof($subsection["indexes"]);
1223     $indexMaxSection = array_search($subsection["max_index"], $subsection["
1224         ↪ indexes"]);
1225     $indexLastKeyPoint = array_key_last($subsection["indexes"]);
1226
1227     if ($subsection["is_prev_section"]) {
1228         $timeDiff = $localKeyPoints[$subsection["indexes"][0]]["time_0"] -
1229             ↪ $localKeyPoints[$subsection["indexes"][0] - 1]["time_1"] -
1230             ↪ $globalTimeOnOneSpeed;
1231         if ($timeDiff < 0) {

```

```

1228 $positionDiff = abs($timeDiff) * $localKeyPoints[$subsection["indexes"]
    ↳ ][0]["speed_0"] / 3.6;
1229 if (!( $localKeyPoints[$subsection["indexes"][0]]["position_1"] +
    ↳ $positionDiff > $localKeyPoints[$subsection["indexes"]
    ↳ $indexMaxSection + 1]["position_0"] )) {
1230 $localKeyPoints[$subsection["indexes"][0]]["position_0"] +=
    ↳ $positionDiff;
1231 $localKeyPoints[$subsection["indexes"][0]]["position_1"] +=
    ↳ $positionDiff;
1232 if ( $localKeyPoints[$subsection["indexes"][0]]["position_1"] >
    ↳ $localKeyPoints[$subsection["indexes"][0] + 1]["position_0"] ) {
1233 array_push($deletedKeyPoints, $subsection["indexes"][0] + 1);
1234 $numberOfKeyPoints -= 1;
1235 $v_0 = $localKeyPoints[$subsection["indexes"][0]]["speed_0"];
1236 $v_1 = $localKeyPoints[$subsection["indexes"][0] + 1]["speed_1"];
1237 $localKeyPoints[$subsection["indexes"][0]]["position_1"] =
    ↳ $localKeyPoints[$subsection["indexes"][0]]["position_0"] +
    ↳ getBrakeDistance($v_0, $v_1, $verzoegerung);
1238 $localKeyPoints[$subsection["indexes"][0]]["speed_1"] = $v_1;
1239 }
1240 $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
    ↳ $deletedKeyPoints);
1241 }
1242 }
1243 }
1244
1245 for ($i = 1; $i <= $indexMaxSection; $i++) {
1246 if (!in_array($subsection["indexes"][$i], $deletedKeyPoints)) {
1247 $timeDiff = $localKeyPoints[$subsection["indexes"][$i]]["time_0"] -
    ↳ $localKeyPoints[$subsection["indexes"][$i] - 1]["time_1"] -
    ↳ $globalTimeOnOneSpeed;
1248 if ($timeDiff < 0) {
1249 $positionDiff = abs($timeDiff) * $localKeyPoints[$subsection["indexes"]
    ↳ [$i]]["speed_0"] / 3.6;
1250 if (!( $localKeyPoints[$subsection["indexes"][$i]]["position_1"] +
    ↳ $positionDiff > $localKeyPoints[$subsection["indexes"]
    ↳ $indexMaxSection + 1]["position_0"] )) {
1251 $localKeyPoints[$subsection["indexes"][$i]]["position_0"] +=
    ↳ $positionDiff;

```

```

1252     $localKeyPoints[$subsection["indexes"][$i]]["position_1"] +=
        ↪ $positionDiff;
1253     if ($i < $indexMaxSection && $localKeyPoints[$subsection["indexes"][$i]
        ↪ ]["position_1"] > $localKeyPoints[$subsection["indexes"][$i] +
        ↪ 1]["position_0"]) {
1254         array_push($deletedKeyPoints, ($subsection["indexes"][$i] + 1));
1255         $numberOfKeyPoints -= 1;
1256         $v_0 = $localKeyPoints[$subsection["indexes"][$i]]["speed_0"];
1257         $v_1 = $localKeyPoints[$subsection["indexes"][$i] + 1]["speed_1"];
1258         $localKeyPoints[$subsection["indexes"][$i]]["position_1"] =
            ↪ $localKeyPoints[$subsection["indexes"][$i]]["position_0"] +
            ↪ getBrakeDistance($v_0, $v_1, $verzoeigerung);
1259         $localKeyPoints[$subsection["indexes"][$i]]["speed_1"] = $v_1;
1260     }
1261     $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
        ↪ $deletedKeyPoints);
1262 }
1263 }
1264 }
1265 }
1266
1267 if ($subsection["is_next_section"]) {
1268     $timeDiff = $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint] +
        ↪ 1]["time_0"] - $localKeyPoints[$subsection["indexes"][$
        ↪ $indexLastKeyPoint]]["time_1"] - $globalTimeOnOneSpeed;
1269     if ($timeDiff < 0) {
1270         $positionDiff = abs($timeDiff) * $localKeyPoints[$indexLastKeyPoint]["
        ↪ speed_1"] / 3.6;
1271         if (!(($localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["
        ↪ position_0"] - $positionDiff < $localKeyPoints[$subsection["indexes"
        ↪ ""][$indexMaxSection]]["position_0"]))) {
1272             $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["position_0"
        ↪ ] -= $positionDiff;
1273             $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["position_1"
        ↪ ] -= $positionDiff;
1274             if ($localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["
        ↪ position_0"] < $localKeyPoints[$subsection["indexes"][$
        ↪ $indexLastKeyPoint] - 1]["position_1"])) {

```

```

1275     array_push($deletedKeyPoints, ($subsection["indexes"][
        ↳ $indexLastKeyPoint] - 1));
1276     $numberOfKeyPoints -= 1;
1277     $v_0 = $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint] -
        ↳ 1]["speed_0"];
1278     $v_1 = $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["
        ↳ speed_1"];
1279     $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["
        ↳ position_0"] = $localKeyPoints[$subsection["indexes"][
        ↳ $indexLastKeyPoint]]["position_1"] - getBrakeDistance($v_0,
        ↳ $v_1, $verzoegerung);
1280     $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["speed_0"]
        ↳ = $v_0;
1281 }
1282 $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
        ↳ $deletedKeyPoints);
1283 }
1284 }
1285 }
1286
1287 for ($i = $indexLastKeyPoint - 1; $i > $indexMaxSection; $i--) {
1288     if (!in_array($i, $deletedKeyPoints)) {
1289         $timeDiff = $localKeyPoints[$subsection["indexes"][$i + 1]]["time_0"] -
            ↳ $localKeyPoints[$subsection["indexes"][$i]]["time_1"] -
            ↳ $globalTimeOnOneSpeed;
1290         if ($timeDiff < 0) {
1291             $positionDiff = abs($timeDiff) * $localKeyPoints[$indexLastKeyPoint]["
                ↳ speed_1"] / 3.6;
1292             if (!($localKeyPoints[$subsection["indexes"][$i]]["position_0"] -
                ↳ $positionDiff < $localKeyPoints[$subsection["indexes"][
                ↳ $indexMaxSection]]["position_0"])) {
1293                 $localKeyPoints[$subsection["indexes"][$i]]["position_0"] -=
                    ↳ $positionDiff;
1294                 $localKeyPoints[$subsection["indexes"][$i]]["position_1"] -=
                    ↳ $positionDiff;
1295                 if ($i > ($indexMaxSection + 1) && $localKeyPoints[$subsection["
                    ↳ indexes"][$i]]["position_0"] < $localKeyPoints[$subsection["
                    ↳ indexes"][$i] - 1]]["position_1"]) {
1296                     array_push($deletedKeyPoints, ($subsection["indexes"][$i] - 1));

```

```

1297     $numberOfKeyPoints -= 1;
1298     $v_0 = $localKeyPoints[$subsection["indexes"][$i] - 1]["speed_0"];
1299     $v_1 = $localKeyPoints[$subsection["indexes"][$i]]["speed_1"];
1300     $localKeyPoints[$subsection["indexes"][$i]]["position_0"] =
        ↳ $localKeyPoints[$subsection["indexes"][$i]]["position_1"] -
        ↳ getBrakeDistance($v_0, $v_1, $verzoegerung);
1301     $localKeyPoints[$subsection["indexes"][$i]]["speed_0"] = $v_0;
1302 }
1303 $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
        ↳ $deletedKeyPoints);
1304 }
1305 }
1306 }
1307 }
1308
1309 $keys = $subsection["indexes"];
1310
1311 foreach ($deletedKeyPoints as $index) {
1312     unset($keys[array_search($index, $keys)]);
1313 }
1314
1315 // Ordnet die Abschnitte zwischen zwei $subsection den $subsections zu
1316 $keys = array_values($keys);
1317 $failed = false;
1318
1319 if ($subsection["is_prev_section"]) {
1320     if ($localKeyPoints[$keys[0]]["time_0"] - $localKeyPoints[$keys[0] - 1][
        ↳ "time_1"] < $globalTimeOnOneSpeed) {
1321         $failed = true;
1322     }
1323 }
1324
1325 if ($subsection["is_next_section"]) {
1326     if ($localKeyPoints[end($keys) + 1]["time_0"] - $localKeyPoints[end($keys)
        ↳ ]["time_1"] < $globalTimeOnOneSpeed) {
1327         $failed = true;
1328     }
1329 }
1330

```

```

1331 for ($i = 1; $i < sizeof($keys); $i++) {
1332     if ($localKeyPoints[$keys[$i]]["time_0"] - $localKeyPoints[$keys[$i - 1]]["
        ↳ time_1"] < $globalTimeOnOneSpeed) {
1333         $failed = true;
1334         break;
1335     }
1336 }
1337
1338 if ($failed) {
1339     return array("fail" => true, "keyPoints" => array());
1340 } else {
1341     foreach ($deletedKeyPoints as $index) {
1342         unset($localKeyPoints[$index]);
1343     }
1344
1345     return array("fail" => false, "keyPoints" => $localKeyPoints);
1346 }
1347 }
1348
1349 // Erstellt mittels der $keyPoints die $subsections
1350 function createSubsections (array $localKeyPoints) {
1351
1352     global $globalTimeOnOneSpeed;
1353
1354     $keyPoints = $localKeyPoints;
1355     $subsections = array();
1356     $subsection = array("max_index" => null, "indexes" => array(), "
        ↳ is_prev_section" => false, "is_next_section" => false);
1357     $maxIndex = null;
1358
1359     for($i = 0; $i < sizeof($keyPoints); $i++) {
1360         if ($i > 0) {
1361             if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"] && $keyPoints[
                ↳ $i - 1]["speed_0"] > $keyPoints[$i - 1]["speed_1"] || $i == sizeof(
                ↳ $keyPoints) - 1) {
1362                 if ($i == sizeof($keyPoints) - 1) {
1363                     array_push($subsection["indexes"], $i);
1364                 }
1365

```

```

1366     array_push($subsections, $subsection);
1367     $subsection["indexes"] = array();
1368 }
1369 }
1370
1371 if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
1372     $subsection["max_index"] = $i;
1373 }
1374
1375 array_push($subsection["indexes"], $i);
1376 }
1377
1378 // Überprüfung der Abschnitte zwischen zwei $subsections
1379 for ($i = 1; $i < sizeof($subsections); $i++) {
1380     $firstIndex = $subsections[$i]["indexes"][array_key_first($subsections[$i][
1381         ↪ "indexes"])]);
1382
1383     if ($keyPoints[$firstIndex]["time_0"] - $keyPoints[$firstIndex - 1]["time_1
1384         ↪ "] < $globalTimeOnOneSpeed) {
1385         $subsections[$i]["is_prev_section"] = true;
1386         $subsections[$i]["failed"] = true;
1387     } else {
1388         $subsections[$i]["is_prev_section"] = false;
1389         $subsections[$i]["failed"] = false;
1390     }
1391 }
1392
1393 for ($i = sizeof($subsections) - 1; $i >= 0; $i--) {
1394     $isFirstSubsection = false;
1395     $isLastSubsection = false;
1396
1397     if ($i == 0) {
1398         $isFirstSubsection = true;
1399     }
1400
1401     if ($i == sizeof($subsections) - 1) {
1402         $isLastSubsection = true;
1403     }

```

```

1403     if ($subsections[$i]["failed"] || failOnSubsection($keyPoints, $subsections
1404         ↪ [$i])) {
1405         $subsections[$i]["failed"] = true;
1406
1407         if (!$isFirstSubsection) {
1408             $subsections[$i]["is_prev_section"] = true;
1409         }
1410
1411         if (!$isLastSubsection) {
1412             if (!$subsections[$i + 1]["is_prev_section"]) {
1413                 $subsections[$i]["is_next_section"] = true;
1414             }
1415         } else {
1416             $subsections[$i]["failed"] = false;
1417         }
1418     }
1419
1420     for ($i = 0; $i < sizeof($subsections); $i++) {
1421         if (!isset($subsections[$i]["max_index"])) {
1422             $subsections[$i]["brakes_only"] = true;
1423             $subsections[$i]["max_index"] = $subsections[$i]["indexes"][0];
1424         } else {
1425             $subsections[$i]["brakes_only"] = false;
1426         }
1427     }
1428
1429     $subsections = array_values($subsections);
1430
1431     return array_reverse($subsections);
1432 }
1433
1434 // Überprüfung, ob es in einer $subsection zu einer Unterschreitung
1435 // der Mindestzeit kommt
1436 function failOnSubsection(array $keyPoints, array $subsection) {
1437
1438     global $globalTimeOnOneSpeed;
1439
1440     $failed = false;

```



```

1441
1442     for ($i = 1; $i < sizeof($subsection["indexes"]); $i++) {
1443         if ($keyPoints[$subsection["indexes"][$i]]["time_0"] - $keyPoints[
1444             ↳ $subsection["indexes"][$i] - 1]["time_1"] < $globalTimeOnOneSpeed) {
1445             $failed = true;
1446             break;
1447         }
1448     }
1449     return $failed;
1450 }
1451
1452 function toShortInSubsection (array $subsections) {
1453
1454     $foundError = false;
1455
1456     foreach ($subsections as $subsection) {
1457         if ($subsection["failed"]) {
1458             $foundError = true;
1459             break;
1460         }
1461     }
1462
1463     return $foundError;
1464 }
1465
1466 function createCumulativeSections ($indexCurrentSection, $indexTargetSection,
1467     ↳ $currentPosition, $targetPosition, $next_lengths) {
1468
1469     $cumLength = array();
1470     $sum = 0;
1471
1472     foreach ($next_lengths as $index => $value) {
1473         if ($index >= $indexCurrentSection) {
1474             $sum += $value;
1475             $cumLength[$index] = $sum;
1476         }
1477     }
1478     // Berechnung der kumulierten Start- und Endlängen der Abschnitte

```

```

1478 for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
1479     if ($indexCurrentSection == $indexTargetSection) {
1480         $cumulativeSectionLengthStart[$i] = 0;
1481         $cumulativeSectionLengthEnd[$i] = $targetPosition - $currentPosition;
1482     } else {
1483         if ($i == $indexCurrentSection) {
1484             $cumulativeSectionLengthStart[$i] = 0;
1485             $cumulativeSectionLengthEnd[$i] = $cumLength[$i] - $currentPosition;
1486         } else if ($i == $indexTargetSection) {
1487             $cumulativeSectionLengthStart[$i] = $cumLength[$i - 1] -
                ↳ $currentPosition;
1488             $cumulativeSectionLengthEnd[$i] = $cumLength[$i - 1] + $targetPosition -
                ↳ $currentPosition;
1489         } else {
1490             $cumulativeSectionLengthStart[$i] = $cumLength[$i - 1] -
                ↳ $currentPosition;
1491             $cumulativeSectionLengthEnd[$i] = $cumLength[$i] - $currentPosition;
1492         }
1493     }
1494 }
1495
1496 return array($cumulativeSectionLengthStart, $cumulativeSectionLengthEnd);
1497 }
1498
1499 function toArr(){
1500     return func_get_args();
1501 }
1502
1503 // Ermittelt die Echtzeitdaten für eine Gefahrenbremsung
1504 function emergencyBreak ($id, $distanceToNextStop = 0) {
1505
1506     global $allUsedTrains;
1507     global $timeDifference;
1508     global $allTimes;
1509
1510     $targetSpeed = 0;
1511     $returnArray = array();
1512     $time = microtime(true) + $timeDifference;
1513     $currentSpeed = $allUsedTrains[$id]["current_speed"];

```

```

1514 $notverzögerung = $allUsedTrains[$id]["notverzögerung"];
1515 $currentSection = $allUsedTrains[$id]["current_section"];
1516
1517 echo "Der Zug mit der Adresse: ", $allUsedTrains[$id]["adresse"], " leitet
    ↳ jetzt eine Notbremsung ein.\n";
1518
1519 if (getBrakeDistance($currentSpeed, $targetSpeed, $notverzögerung) <=
    ↳ $distanceToNextStop) {
1520     for ($i = $currentSpeed; $i >= 0; $i = $i - 2) {
1521         array_push($returnArray, array("live_position" => 0, "live_speed" => $i,
            ↳ "live_time" => $time, "live_relative_position" => 0, "live_section"
            ↳ => $currentSection, "live_is_speed_change" => true, "
            ↳ live_target_reached" => false, "id" => $id, "wendet" => false, "
            ↳ betriebsstelle" => 'Notbremsung', "live_all_targets_reached" =>
            ↳ null));
1522         $time = $time + getBrakeTime($i, $i - 1, $notverzögerung);
1523     }
1524 } else {
1525     $targetSpeedNotbremsung = getTargetBrakeSpeedWithDistanceAndStartSpeed(
        ↳ $distanceToNextStop, $notverzögerung, $currentSpeed);
1526     $speedBeforeStop = intval($targetSpeedNotbremsung / 2) * 2;
1527
1528     if ($speedBeforeStop >= 10) {
1529         for ($i = $currentSpeed; $i >= 10; $i = $i - 2) {
1530             array_push($returnArray, array("live_position" => 0, "live_speed" => $i,
                ↳ "live_time" => $time, "live_relative_position" => 0, "
                ↳ live_section" => $currentSection, "live_is_speed_change" => true,
                ↳ "live_target_reached" => false, "id" => $id, "wendet" => false,
                ↳ "betriebsstelle" => 'Notbremsung', "live_all_targets_reached" =>
                ↳ null));
1531             $time = $time + getBrakeTime($i, $i - 1, $notverzögerung);
1532         }
1533         array_push($returnArray, array("live_position" => 0, "live_speed" => 0, "
            ↳ live_time" => $time, "live_relative_position" => 0, "live_section"
            ↳ => $currentSection, "live_is_speed_change" => true, "
            ↳ live_target_reached" => false, "id" => $id, "wendet" => false, "
            ↳ betriebsstelle" => 'Notbremsung', "live_all_targets_reached" =>
            ↳ null));
1534     } else {

```

```

1535     array_push($returnArray, array("live_position" => 0, "live_speed" =>
        ↳ $currentSpeed, "live_time" => $time, "live_relative_position" => 0,
        ↳ "live_section" => $currentSection, "live_is_speed_change" => true,
        ↳ "live_target_reached" => false, "id" => $id, "wendet" => false, "
        ↳ betriebsstelle" => 'Notbremsung', "live_all_targets_reached" =>
        ↳ null));
1536     $time = $time + getBrakeTime($currentSpeed, $currentSpeed - 1,
        ↳ $notverzoegerung);
1537     array_push($returnArray, array("live_position" => 0, "live_speed" => 0, "
        ↳ live_time" => $time, "live_relative_position" => 0, "live_section"
        ↳ => $currentSection, "live_is_speed_change" => true, "
        ↳ live_target_reached" => false, "id" => $id, "wendet" => false, "
        ↳ betriebsstelle" => 'Notbremsung', "live_all_targets_reached" =>
        ↳ null));
1538 }
1539 }
1540
1541 $allTimes[$allUsedTrains[$id]["adresse"]] = $returnArray;
1542 array_push($allUsedTrains[$id]["error"], 3);
1543
1544 return 0;
1545 }

```

## A.4 functions\_math.php

```
1 <?php
2
3 // Ermittelt die Geschwindigkeit, die ein Fahrzeug in einem Bremsvorgang
4 // nach einer gegebenen Distanz hat.
5 function getTargetBrakeSpeedWithDistanceAndStartSpeed (float $distance, float
    ↳ $verzoeigerung, int $speed) {
6     return sqrt((-2 * $verzoeigerung * $distance) + (pow(($speed / 3.6), 2)))*3.6;
7 }
8
9 // Ermittelt die Distanz, um die eine Verzögerung "verschoben" werden müsste,
10 // damit die exakte Ankunftszeit eingehalten werden kann.
11 function calculateDistanceforSpeedFineTuning(int $v_0, int $v_1, float
    ↳ $distance, float $time) : float {
12     return $distance - (($distance - $time * $v_1 / 3.6)/($v_0 / 3.6 - $v_1 /
    ↳ 3.6)) * ($v_0 / 3.6);
13 }
14
15 // Ermittelt die Distanz für Brems- und Verzögerungsvorgänge
16 function getBrakeTime (float $v_0, float $v_1, float $verzoeigerung) {
17     return abs((( $v_1/3.6)/$verzoeigerung) - (( $v_0/3.6)/$verzoeigerung));
18 }
19
20 // Ermittelt die Zeit, die ein Fahrzeug bei einer gegebenen Strecke für
21 // eine gegebene Distanz benötigt
22 function distanceWithSpeedToTime (int $v, float $distance) {
23     return (($distance)/($v / 3.6));
24 }
25
26 // Ermittlung der Strecke für eine Beschleunigung bzw. Verzögerung
27 function getBrakeDistance (float $v_0, float $v_1, float $verzoeigerung) {
28     return abs(0.5 * ((pow($v_0/3.6,2) - pow($v_1/3.6, 2))/($verzoeigerung)));
29 }
```

## A.5 functions\_\_cache.php

```
1 <?php
2
3 function createcacheInfraLaenge() {
4     $DB = new DB_MySQL();
5     $returnArray = array();
6     $infraLaenge = $DB->select("SELECT '".DB_TABLE_INFRAZUSTAND."'.'.id', '".
        ↳ DB_TABLE_INFRAZUSTAND."'.'.laenge' FROM '".DB_TABLE_INFRAZUSTAND."'
        ↳ WHERE '".DB_TABLE_INFRAZUSTAND."'.'.type' = '".gleis"."'");
7     unset($DB);
8     foreach ($infraLaenge as $data) {
9         if ($data->laenge != null) {
10             $returnArray[$data->id] = intval($data->laenge);
11         }
12     }
13     return $returnArray;
14 }
15
16 function createCacheHaltepunkte() : array{
17
18     $DB = new DB_MySQL();
19     $returnArray = array();
20
21     $betriebsstellen = $DB->select("SELECT '".DB_TABLE_BETRIEBSSTELLEN_DATEN."'.'.
        ↳ parent_kuerzel' FROM '".DB_TABLE_BETRIEBSSTELLEN_DATEN."' WHERE '".
        ↳ DB_TABLE_BETRIEBSSTELLEN_DATEN."'.'.parent_kuerzel' IS NOT NULL");
22     unset($DB);
23
24     foreach ($betriebsstellen as $betriebsstelle) {
25         $returnArray[$betriebsstelle->parent_kuerzel][0] = array();
26         $returnArray[$betriebsstelle->parent_kuerzel][1] = array();
27     }
28
29     foreach ($returnArray as $betriebsstelleKey => $betriebsstelleValue) {
30         $DB = new DB_MySQL();
31         $name = $betriebsstelleKey;
32         $name .= "%";
33         $asig = "ASig";
34         $bksig = "BkSig";
```

```

35 $vsig = "VSig";
36 $ja = "ja";
37 if ($betriebsstelleKey == 'XAB' || $betriebsstelleKey == "XBL") {
38     $haltepunkte = $DB->select("SELECT '".DB_TABLE_SIGNALE_STANDORTE.'".'.
        ↳ freimelde_id', '".DB_TABLE_SIGNALE_STANDORTE.'".'.wirkrichtung' FROM
        ↳ '".DB_TABLE_SIGNALE_STANDORTE.'" WHERE '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'".'.betriebsstelle' LIKE '$name' AND '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'".'.freimelde_id' IS NOT NULL AND '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'".'.fahrplanhalt' = '$ja'");
39     unset($DB);
40 } else if ($betriebsstelleKey == 'XTS') {
41     $haltepunkte = $DB->select("SELECT '".DB_TABLE_SIGNALE_STANDORTE.'".'.
        ↳ freimelde_id', '".DB_TABLE_SIGNALE_STANDORTE.'".'.wirkrichtung' FROM
        ↳ '".DB_TABLE_SIGNALE_STANDORTE.'" WHERE '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'".'.betriebsstelle' LIKE '$name' AND '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'".'.freimelde_id' IS NOT NULL AND '".
        ↳ DB_TABLE_SIGNALE_STANDORTE . "'.'.signaltyp' = '$bksig'");
42     unset($DB);
43 } else if ($betriebsstelleKey == 'XLG') {
44     $haltepunkte = $DB->select("SELECT '".DB_TABLE_SIGNALE_STANDORTE.'".'.
        ↳ freimelde_id', '".DB_TABLE_SIGNALE_STANDORTE.'".'.wirkrichtung' FROM
        ↳ '".DB_TABLE_SIGNALE_STANDORTE.'" WHERE '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'".'.betriebsstelle' LIKE '$name' AND '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'".'.freimelde_id' IS NOT NULL AND '".
        ↳ DB_TABLE_SIGNALE_STANDORTE . "'.'.signaltyp' != '$vsig'");
45     unset($DB);
46 } else {
47     $haltepunkte = $DB->select("SELECT '".DB_TABLE_SIGNALE_STANDORTE . "'.'.
        ↳ freimelde_id', '".DB_TABLE_SIGNALE_STANDORTE . "'.'.wirkrichtung'
        ↳ FROM '".DB_TABLE_SIGNALE_STANDORTE . "' WHERE '".
        ↳ DB_TABLE_SIGNALE_STANDORTE . "'.'.betriebsstelle' LIKE '$name' AND '
        ↳ ".DB_TABLE_SIGNALE_STANDORTE . "'.'.freimelde_id' IS NOT NULL AND
        ↳ '".DB_TABLE_SIGNALE_STANDORTE . "'.'.signaltyp' = '$asig'");
48     unset($DB);
49 }
50
51 foreach ($haltepunkte as $haltepunkt) {
52     if ($haltepunkt->wirkrichtung == 0) {

```

```

53     array_push($returnArray[$betriebsstelleKey][0], intval($haltepunkt->
        ↳ freimelde_id));
54 } elseif ($haltepunkt->wirkrichtung == 1) {
55     array_push($returnArray[$betriebsstelleKey][1], intval($haltepunkt->
        ↳ freimelde_id));
56 }
57 }
58 }
59 $returnArray["XSC"][1] = array(734, 732, 735, 733, 692); // In der Datenbank
        ↳ ist für Richtung 1 für diese Abschnitte fahrplanhalt auf nein
        ↳ eingestellt
60 return $returnArray;
61 }
62
63 function createCacheZwischenhaltepunkte() {
64     $DB = new DB_MySQL();
65     $allZwischenhalte = array();
66     $returnArray = array();
67     $zwischenhalte = $DB->select("SELECT DISTINCT '".DB_TABLE_SIGNALE_STANDORTE.'"
        ↳ 'betriebsstelle' FROM '".DB_TABLE_SIGNALE_STANDORTE.'" WHERE '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'".'betriebsstelle' IS NOT NULL");
68     unset($DB);
69     foreach ($zwischenhalte as $halt) {
70         array_push($allZwischenhalte, $halt->betriebsstelle);
71     }
72     foreach ($allZwischenhalte as $halt) {
73         $DB = new DB_MySQL();
74         $zwischenhalte = $DB->select("SELECT '".DB_TABLE_SIGNALE_STANDORTE.'".'
        ↳ freimelde_id' FROM '".DB_TABLE_SIGNALE_STANDORTE.'" WHERE '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'".'betriebsstelle' = '$halt' AND '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'".'freimelde_id' IS NOT NULL");
75         unset($DB);
76         if (sizeof($zwischenhalte) == 1) {
77             if (sizeof(explode("_", $halt)) == 2) {
78                 $returnArray[$halt] = intval($zwischenhalte[0]->freimelde_id);
79             }
80         }
81     }
82     return $returnArray;

```



```

83 }
84
85 function createCacheInfraToGbt () {
86     $DB = new DB_MySQL();
87     $infraArray = array();
88     $returnArray = array();
89     $allInfra = $DB->select("SELECT '".DB_TABLE_FMA_GBT."'.'infra_id' FROM '".
        ↳ DB_TABLE_FMA_GBT."' WHERE '".DB_TABLE_FMA_GBT."'.'infra_id' IS NOT
        ↳ NULL");
90     unset($DB);
91     foreach ($allInfra as $infra) {
92         array_push($infraArray, intval($infra->infra_id));
93     }
94     foreach ($infraArray as $infra) {
95         $DB = new DB_MySQL();
96         $gbt = $DB->select("SELECT '".DB_TABLE_FMA_GBT."'.'gbt_id' FROM '".
        ↳ DB_TABLE_FMA_GBT."' WHERE '".DB_TABLE_FMA_GBT."'.'infra_id' = '
        ↳ $infra'")[0]->gbt_id;
97         unset($DB);
98         $returnArray[$infra] = intval($gbt);
99     }
100     return $returnArray;
101 }
102
103 function createCacheGbtToInfra () {
104
105     $DB = new DB_MySQL();
106
107     $returnArray = array();
108
109     $allGbt = $DB->select("SELECT DISTINCT '".DB_TABLE_FMA_GBT."'.'gbt_id' FROM '
        ↳ '".DB_TABLE_FMA_GBT."' WHERE '".DB_TABLE_FMA_GBT."'.'gbt_id' IS NOT
        ↳ NULL");
110     unset($DB);
111
112     foreach ($allGbt as $gbt) {
113         $DB = new DB_MySQL();
114         $gbt = $gbt->gbt_id;

```

```

115     $infras = $DB->select("SELECT '".DB_TABLE_FMA_GBT."'.'infra_id' FROM '".
        ↳ DB_TABLE_FMA_GBT."' WHERE '".DB_TABLE_FMA_GBT."'.'gbt_id' = '$gbt'")
        ↳ ;
116     unset($DB);
117     $returnArray[$gbt] = array();
118     foreach ($infras as $infra) {
119         array_push($returnArray[$gbt], intval($infra->infra_id));
120     }
121 }
122 return $returnArray;
123 }
124
125 function createCacheFmaToInfra () {
126     $DB = new DB_MySQL();
127     $returnArray = array();
128     $fmaToInfra = $DB->select("SELECT '".DB_TABLE_FMA_GBT."'.'infra_id', '".
        ↳ DB_TABLE_FMA_GBT."'.'fma_id' FROM '".DB_TABLE_FMA_GBT."' WHERE '".
        ↳ DB_TABLE_FMA_GBT."'.'fma_id' IS NOT NULL");
129     unset($DB);
130     foreach ($fmaToInfra as $value) {
131         $returnArray[intval($value->fma_id)] = intval($value->infra_id);
132     }
133     return $returnArray;
134 }
135
136 function createCacheToBetriebsstelle() {
137     $DB = new DB_MySQL();
138     $returnArray = array();
139     $fmaToInfra = $DB->select("SELECT '".DB_TABLE_SIGNALE_STANDORTE."'.'id', '".
        ↳ DB_TABLE_SIGNALE_STANDORTE."'.'betriebsstelle' FROM '".
        ↳ DB_TABLE_SIGNALE_STANDORTE."'");
140     unset($DB);
141     foreach ($fmaToInfra as $value) {
142         $returnArray[intval($value->id)] = $value->betriebsstelle;
143     }
144     return $returnArray;
145 }
146
147 function createCacheFahrzeugeAbschnitte () {

```

```

148 $DB = new DB_MySQL();
149 $returnArray = array();
150 $fahrzeugeAbschnitte = $DB->select("SELECT '".DB_TABLE_FAHRZEUGE_ABSCHNITTE.'"
    ↳ 'fahrzeug_id', '".DB_TABLE_FAHRZEUGE_ABSCHNITTE.'" 'infra_id', '".
    ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE.'" 'unixtimestamp' FROM '".
    ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE.'"");
151 unset($DB);
152 foreach ($fahrzeugeAbschnitte as $fahrzeug) {
153     $returnArray[intval($fahrzeug->fahrzeug_id)][intval($fahrzeug
    ↳ ->infra_id)] = intval($fahrzeug
154     $returnArray[intval($fahrzeug->fahrzeug_id)[intval($fahrzeug->unixtimestamp)] = intval(
    ↳ $fahrzeug->unixtimestamp);
155 }
156 return $returnArray;
157 }
158
159 function createCacheDecoderToAdresse () {
160     $DB = new DB_MySQL();
161     $returnArray = array();
162     $decoderToAdresse = $DB->select("SELECT '".DB_TABLE_FAHRZEUGE.'" 'id', '".
    ↳ DB_TABLE_FAHRZEUGE.'" 'adresse' FROM '".DB_TABLE_FAHRZEUGE.'"");
163     unset($DB);
164     foreach ($decoderToAdresse as $fahrzeug) {
165         $returnArray[intval($fahrzeug->id)] = intval($fahrzeug->adresse);
166     }
167     return $returnArray;
168 }
169
170 function createCacheFahrplanSession() {
171     $DB = new DB_MySQL();
172     $fahrplanData = $DB->select("SELECT * FROM '".DB_TABLE_FAHRPLAN_SESSION.'" '
    ↳ WHERE '".DB_TABLE_FAHRPLAN_SESSION.'" 'status' = '".1"."'");
173     unset($DB);
174
175     return $fahrplanData[0];
176 }

```

## A.6 functions\_db.php

```
1 <?php
2
3 // Ermittelt die Daten aller Fahrzeuge.
4 function getAllTrains () {
5
6     global $cacheAdresseToID;
7     global $cacheIDToAdresse;
8     global $globalTrainVMax;
9
10    $allAddresses = getAllAddresses();
11    $DB = new DB_MySQL();
12    $allTrains = array();
13    $id = null;
14
15    foreach ($allAddresses as $adress) {
16        $train_fahrzeuge = get_object_vars($DB->select("SELECT '".
            ↳ DB_TABLE_FAHRZEUGE.".'. 'id', '".DB_TABLE_FAHRZEUGE.".'. 'adresse', '".
            ↳ DB_TABLE_FAHRZEUGE.".'. 'speed', '".DB_TABLE_FAHRZEUGE.".'. 'dir', '".
            ↳ DB_TABLE_FAHRZEUGE.".'. 'zugtyp', '".DB_TABLE_FAHRZEUGE.".'. 'zuglaenge
            ↳ ', '".DB_TABLE_FAHRZEUGE.".'. 'verzoegerung', '".DB_TABLE_FAHRZEUGE."
            ↳ '. 'zustand' FROM '".DB_TABLE_FAHRZEUGE.".' WHERE '".
            ↳ DB_TABLE_FAHRZEUGE.".'. 'adresse' = $adress")[0]);
17        $id = $train_fahrzeuge["id"];
18        $train_daten = $DB->select("SELECT '".DB_TABLE_FAHRZEUGE_DATEN.".'. 'baureihe
            ↳ ' FROM '".DB_TABLE_FAHRZEUGE_DATEN.".' WHERE '".
            ↳ DB_TABLE_FAHRZEUGE_DATEN.".'. 'id' = $id")[0]->baureihe;
19        $train_baureihe = $DB->select("SELECT '".DB_TABLE_FAHRZEUGE_BAUREIHEN.".'. '
            ↳ vmax' FROM '".DB_TABLE_FAHRZEUGE_BAUREIHEN.".' WHERE '".
            ↳ DB_TABLE_FAHRZEUGE_BAUREIHEN.".'. 'nummer' = $train_daten");
20
21        if (sizeof($train_baureihe) != 0) {
22            $train_baureihe_return["v_max"] = intval($train_baureihe[0]->vmax);
23        } else {
24            $train_baureihe_return["v_max"] = $globalTrainVMax;
25        }
26
27        $id = intval($train_fahrzeuge["id"]);
28        $cacheAdresseToID[intval($train_fahrzeuge["adresse"])] = intval($id);
```

```

29     $returnArray = array_merge($train_fahrzeuge, $train_baureihe_return);
30     $allTrains[$id] = $returnArray;
31 }
32
33 unset($DB);
34
35 $cacheIDToAdresse = array_flip($cacheAdresseToID);
36
37 return $allTrains;
38 }
39
40 // Ermittelt alle Fahrzeuge im eingleisigen Netz und gibt die neu hinzugefügten
41 // und entfernten Fahrzeuge getrennt zurück.
42 function updateAllTrainsOnTheTrack () {
43
44     global $allTrainsOnTheTrack;
45     $newTrains = array();
46     $removedTrains = array();
47     $allTrains = array();
48     $DB = new DB_MySQL();
49     $foundTrains = $DB->select("SELECT DISTINCT '".DB_TABLE_FMA.'"'. '
        ↳ decoder_adresse' FROM '".DB_TABLE_FMA.'" WHERE '".DB_TABLE_FMA.'"'. '
        ↳ decoder_adresse' IS NOT NULL AND '".DB_TABLE_FMA.'"'. 'decoder_adresse'
        ↳ <> '".0"."'");
50     unset($DB);
51
52     foreach ($foundTrains as $train) {
53         array_push($allTrains, intval($train->decoder_adresse));
54         if (!in_array($train->decoder_adresse, $allTrainsOnTheTrack)) {
55             array_push($newTrains, intval($train->decoder_adresse));
56         }
57     }
58
59     foreach ($allTrainsOnTheTrack as $train) {
60         if (!in_array($train, $allTrains)) {
61             array_push($removedTrains, $train);
62         }
63     }
64 }

```

```

65     $allTrainsOnTheTrack = $allTrains;
66     return array("new"=>$newTrains, "removed"=>$removedTrains);
67 }
68
69 // Ermittelt alle Fahrzeuge im eingleisigen Netz.
70 function findTrainsOnTheTracks () {
71
72     global $allTrainsOnTheTrack;
73
74     $DB = new DB_MySQL();
75     $foundTrains = $DB->select("SELECT DISTINCT '".DB_TABLE_FMA.'"'. '
        ↳ decoder_adresse' FROM '".DB_TABLE_FMA.'" WHERE '".DB_TABLE_FMA.'"'. '
        ↳ decoder_adresse' IS NOT NULL AND '".DB_TABLE_FMA.'"'. 'decoder_adresse'
        ↳ <> '".0"."'");
76     unset($DB);
77
78     foreach ($foundTrains as $train) {
79         if (!in_array($train->decoder_adresse, $allTrainsOnTheTrack)) {
80             array_push($allTrainsOnTheTrack, intval($train->decoder_adresse));
81             prepareTrainForRide($train->decoder_adresse);
82         }
83     }
84 }
85
86 // Bestimmung der Position eines Zuges über die Adresse.
87 function getPosition(int $adresse) {
88
89     $returnPosition = array();
90     $DB = new DB_MySQL();
91     $position = $DB->select("SELECT '".DB_TABLE_FMA.'"'. 'fma_id' FROM '".
        ↳ DB_TABLE_FMA.'" WHERE '".DB_TABLE_FMA.'"'. 'decoder_adresse' = $adresse"
        ↳ );
92     unset($DB);
93
94     if (sizeof($position) != 0) {
95         for ($i = 0; $i < sizeof($position); $i++) {
96             array_push($returnPosition, intval(get_object_vars($position[$i])["fma_id
                ↳ "]]));
97         }

```

```

98     }
99
100     return $returnPosition;
101 }
102
103 // Ermittelt die Fahrplandaten eines Zuges
104 function getNextBetriebsstellen (int $id) {
105
106     $DB = new DB_MySQL();
107     $returnBetriebsstellen = array();
108     $betriebsstellen = $DB->select("SELECT '".DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN.'"
        ↳ '`.`betriebsstelle' FROM '".DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN.'" WHERE
        ↳ '".DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN.'"`.`zug_id' = $id ORDER BY '".
        ↳ DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN.'"`.`id' ASC");
109     unset($DB);
110
111     foreach ($betriebsstellen as $betriebsstellenIndex => $betriebsstellenValue)
        ↳ {
112         array_push($returnBetriebsstellen, $betriebsstellenValue->betriebsstelle);
113     }
114
115     if (sizeof($betriebsstellen) == 0) {
116         debugMessage("Zu dieser Zug ID sind keine nächsten Betriebsstellen im
        ↳ Fahrplan vorhanden.");
117     }
118
119     return $returnBetriebsstellen;
120 }
121
122 // Bestimmt das zugehörige Signal (falls vorhanden) für einen Abschnitt und
        ↳ eine
123 // Richtung.
124 function getSignalForSectionAndDirection(int $section, int $dir) {
125
126     $DB = new DB_MySQL();
127     $signal = $DB->select("SELECT '".DB_TABLE_SIGNALE_STANDORTE.'"`.`id' FROM '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'" WHERE '".DB_TABLE_SIGNALE_STANDORTE.'"`.`
        ↳ freimelde_id' = $section AND '".DB_TABLE_SIGNALE_STANDORTE.'"`.`
        ↳ wirkrichtung' = $dir");

```

```

128     unset($DB);
129
130     if ($signal != null) {
131         $signal = intval(get_object_vars($signal[0])["id"]);
132     }
133
134     return $signal;
135 }
136
137 // Kalibriert die Position des Fahrzeugs neu anhand der Daten in der Tabelle
138 // 'fahrzeuge_abschnitte'
139 function getCalibratedPosition ($id, $speed) {
140
141     global $cacheFahrzeugeAbschnitte;
142
143     $DB = new DB_MySQL();
144     $positionReturn = $DB->select("SELECT '".DB_TABLE_FAHRZEUGE_ABSCHNITTE.'".'. '
        ↳ infra_id', '".DB_TABLE_FAHRZEUGE_ABSCHNITTE.'".'. 'unixtimestamp' FROM '".
        ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE.'" WHERE '".
        ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE.'".'. 'fahrzeug_id' = $id")[0];
145     unset($DB);
146
147     if (in_array($id, array_keys($cacheFahrzeugeAbschnitte))) {
148         if ($positionReturn->unixtimestamp == $cacheFahrzeugeAbschnitte[$id]["
            ↳ unixtimestamp"]) {
149             return array("possible" => false);
150         }
151     }
152
153     $timeDiff = time() - $positionReturn->unixtimestamp;
154     $position = ($speed / 3.6) * $timeDiff;
155
156     return array("section" => $positionReturn->infra_id, "position" => $position)
        ↳ ;
157 }
158
159 // Liest die Adressen aller Fahrzeuge ein.
160 function getAllAdresses () : array {
161

```



```

162 $zustand = array("0", "1");
163 $returnAdresses = array();
164
165 echo "Alle Züge, die den Zustand ", implode(", ", $zustand), " haben, werden
    ↳ eingelesen.\n\n";
166
167 $DB = new DB_MySQL();
168 $adresses = $DB->select("SELECT '".DB_TABLE_FAHRZEUGE.'"'. 'adresse', '".
    ↳ DB_TABLE_FAHRZEUGE.'"'. 'zustand' FROM '".DB_TABLE_FAHRZEUGE.'"');
169 unset($DB);
170
171 foreach ($adresses as $adressIndex => $adressValue) {
172     if (in_array($adressValue->zustand, $zustand)) {
173         array_push($returnAdresses, (int) $adressValue->adresse);
174     }
175 }
176
177 return $returnAdresses;
178 }

```

## A.7 global\_variables.php

```
1 <?php
2
3 $globalNotverzoegerung = 2; // Bremsverzögerung bei einer Notbremsung
4 $globalTrainVMax = 10; // Maximale Geschwindigkeit, wenn keine vorgegeben ist
5 $globalSpeedInCurrentSection = 60; // Maximale Geschwindigkeit im aktuellen
   ↳ Abschnitt
6 $globalFirstHaltMinTime = 20; // Mindesthaltezeit am ersten fahrplanmäßigen
   ↳ Halt
7 $globalIndexBetriebsstelleFreieFahrt = 999999999999;
8 $globalFloatingPointNumbersRoundingError = 0.0000000001;
9 $globalTimeOnOneSpeed = 20;
10 $globalTimeUpdateInterval = 1;
11
12 $useSpeedFineTuning = true;
13 $useMinTimeOnSpeed = true;
14 $errorMinTimeOnSpeed = false;
15 $slowDownIfTooEarly = true;
16 $useRecalibration = true;
```

## A.8 speed\_over\_position.m

```
1 %% Load cumulative sections
2
3 fname = '../json/VMaxOverCumulativeSections.json';
4 fid = fopen(fname);
5 raw = fread(fid,inf);
6 str = char(raw');
7 fclose(fid);
8 vmaxOverPosition = jsondecode(str);
9 vmaxOverPosition_Position = vmaxOverPosition(:,1);
10 vmaxOverPosition_v_max = vmaxOverPosition(:,2);
11
12 %% Load modified cumulative sections
13
14 fname = '../json/VMaxOverCumulativeSectionsMod.json';
15 fid = fopen(fname);
16 raw = fread(fid,inf);
17 str = char(raw');
18 fclose(fid);
19 vmaxOverPosition_mod = jsondecode(str);
20 vmaxOverPosition_Position_mod = vmaxOverPosition_mod(:,1);
21 vmaxOverPosition_v_max_mod = vmaxOverPosition_mod(:,2);
22
23 %% Load speed over position
24
25 fname = '../json/speedOverPosition.json';
26 fid = fopen(fname);
27 raw = fread(fid,inf);
28 str = char(raw');
29 fclose(fid);
30 val = jsondecode(str);
31
32 speedOverPosition_x_v1 = val(:,1);
33 speedOverPosition_y_v1 = val(:,2);
34
35 %% Load speed over position (all iterationsteps)
36
37 fname_it = '../json/speedOverPosition_prevIterations.json';
38 fid_it = fopen(fname_it);
```

```

39 raw_it = fread(fid_it,inf);
40 str_it = char(raw_it');
41 fclose(fid_it);
42 val_it = jsondecode(str_it);
43
44 %% Plot
45
46 hold on
47 figure(1)
48
49 % Plot infrastructuresections
50
51 p = line([0 0], [0 vmaxOverPosition_v_max(1)], 'LineStyle', '-.', 'LineWidth', 2, '
    ↳ color', 'black', 'DisplayName', ['Infra-Abschnitte']);
52 line([0 vmaxOverPosition_Position(1)], [vmaxOverPosition_v_max(1)
    ↳ vmaxOverPosition_v_max(1)], 'LineStyle', '-.', 'LineWidth', 2, 'color', 'black
    ↳ ', 'HandleVisibility', 'off');
53
54 for i = 1:size(vmaxOverPosition_Position) - 1
55     line([vmaxOverPosition_Position(i) vmaxOverPosition_Position(i + 1)], [
        ↳ vmaxOverPosition_v_max(i + 1) vmaxOverPosition_v_max(i + 1)], '
        ↳ LineStyle', '-.', 'LineWidth', 2, 'color', 'black', 'HandleVisibility', '
        ↳ off');
56     line([vmaxOverPosition_Position(i) vmaxOverPosition_Position(i)], [0
        ↳ vmaxOverPosition_v_max(i + 1)], 'LineStyle', '-.', 'LineWidth', 2, 'color'
        ↳ ', 'black', 'HandleVisibility', 'off');
57     line([vmaxOverPosition_Position(i) vmaxOverPosition_Position(i)], [0
        ↳ vmaxOverPosition_v_max(i)], 'LineStyle', '-.', 'LineWidth', 2, 'color', '
        ↳ black', 'HandleVisibility', 'off');
58     line([vmaxOverPosition_Position(i + 1) vmaxOverPosition_Position(i + 1)], [0
        ↳ vmaxOverPosition_v_max(i + 1)], 'LineStyle', '-.', 'LineWidth', 2, 'color
        ↳ ', 'black', 'HandleVisibility', 'off');
59 end
60
61 % Plot modified iterationsteps (incl. trainlength)
62 line([0 0], [0 vmaxOverPosition_v_max_mod(1)], 'LineStyle', '-.', 'LineWidth', 2, '
    ↳ color', 'red', 'HandleVisibility', 'off');
63 line([0 vmaxOverPosition_Position_mod(1)], [vmaxOverPosition_v_max_mod(1)
    ↳ vmaxOverPosition_v_max_mod(1)], 'LineStyle', '-.', 'LineWidth', 2, 'color', '

```

```

        ↪ red', 'DisplayName', ['Infra-Abschnitte' newline 'inkl. Zuglänge']);
64
65 for i = 1:size(vmaxOverPosition_Position_mod) - 1
66     line([vmaxOverPosition_Position_mod(i) vmaxOverPosition_Position_mod(i + 1)
        ↪ ], [vmaxOverPosition_v_max_mod(i + 1) vmaxOverPosition_v_max_mod(i +
        ↪ 1)], 'LineStyle', '-.', 'LineWidth', 2, 'color', 'red', 'HandleVisibility',
        ↪ 'off');
67     line([vmaxOverPosition_Position_mod(i) vmaxOverPosition_Position_mod(i)], [0
        ↪ vmaxOverPosition_v_max_mod(i + 1)], 'LineStyle', '-.', 'LineWidth', 2, '
        ↪ color', 'red', 'HandleVisibility', 'off');
68     line([vmaxOverPosition_Position_mod(i) vmaxOverPosition_Position_mod(i)], [0
        ↪ vmaxOverPosition_v_max_mod(i)], 'LineStyle', '-.', 'LineWidth', 2, 'color
        ↪ ', 'red', 'HandleVisibility', 'off');
69     line([vmaxOverPosition_Position_mod(i + 1) vmaxOverPosition_Position_mod(i +
        ↪ 1)], [0 vmaxOverPosition_v_max_mod(i + 1)], 'LineStyle', '-.', '
        ↪ LineWidth', 2, 'color', 'red', 'HandleVisibility', 'off');
70 end
71
72 % Plot all iterationsteps
73 for i = 1:length(val_it)
74     plot(val_it{i}(:,1), val_it{i}(:,2), '.', 'markersize', 8, 'Color', [0.6 0.6
        ↪ 0.6], 'DisplayName', legend_name);
75
76 end
77
78 % PLOT speedcurve
79
80 plot(speedOverPosition_x_v1, speedOverPosition_y_v1, 'LineWidth', 4, 'Color', [0.25
        ↪ 0.80 0.54], 'DisplayName', 'Fahrtverlauf');
81
82 %% Fillobjects
83
84 %fill([0, 0, 1090, 1090, 0], [0, 200, 200, 0, 0], 'b', 'facealpha', .2, 'LineStyle
        ↪ ', 'none');
85
86 %% Adding text
87
88 %text(20,17,'1','Interpreter','latex','fontSize', 40);
89

```

```

90 %% Format plot
91
92 p.LineWidth = 2;
93 box off
94 fontSize = 18;
95 xlabel("Strecke [m]", 'FontSize', fontSize);
96 ylabel("Geschwindigkeit [km/h]", 'FontSize', fontSize);
97 x0=10;
98 y0=10;
99 width=1100;
100 height=600;
101 axis([-80 max(vmaxOverPosition_Position)+80 0 max(vmaxOverPosition_v_max)+10]);
102 axis([-20 max(vmaxOverPosition_Position)+20 0 max(vmaxOverPosition_v_max)+5]);
103 set(gcf, 'position', [x0,y0,width,height]);
104 set(gcf, 'PaperPositionMode', 'auto');
105 set(gca, 'FontSize', 18);
106 set(gca, 'Linewidth', 2);
107
108 t = gca;
109 exportgraphics(t, 'SpeedOverPosition.pdf', 'ContentType', 'vector');
110 hold off

```

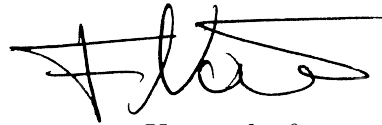
## Literatur

- Büker, T., Hennig, E. & Schotten, S. (2020). Kapazitätsberechnung im moving block – die tücke im detail. *Eisenbahntechnische Rundschau*, 7+8, 32–37. Zugriff auf [www.via-con.de/wp-content/uploads/32\\_37\\_Bueker\\_Hennig\\_Schotten\\_FA.pdf](http://www.via-con.de/wp-content/uploads/32_37_Bueker_Hennig_Schotten_FA.pdf) (Letzter Zugriff am: 21. September 2021)
- Ebuef: *Eisenbahn-Betriebs- und Experimentierfeld Berlin*. (2021). [www.ebuef.de](http://www.ebuef.de). EBUef e.V.; c/o Technische Universität Berlin; Fachgebiet Bahnbetrieb und Infrastruktur. (Letzter Zugriff am: 11. September 2021)
- Maschek, U. (2018). *Sicherung des Schienenverkehrs* (4. Aufl.). Springer-Verlag. (ISBN: 978-3-658-22878-1)
- Pachl, J. (2021). *Systemtechnik des Schienenverkehrs* (10. Aufl.). Springer-Verlag. (ISBN: 978-3-658-31165-0)
- RailCom - *DCC-Rückmeldeprotokoll* (Norm Nr. RCN-217). (2019, Dezember). (Rail-Community – Verband der Hersteller Digitaler Modellbahnprodukte e.V.)
- Richard, H. & Sander, M. (2011). *Technische mechanik. dynamik: Grundlagen - effektiv und anwendungsnah* (2. Aufl.). Vieweg+Teubner Verlag. (ISBN: 978-3-658-05027-6)
- The IEEE and The Open Group. (2018). *The open group base specifications issue 7 – ieee std 1003.1, 2018 edition*. New York, NY, USA: IEEE. Zugriff auf <https://pubs.opengroup.org/onlinepubs/9699919799> (Letzter Zugriff am: 16. September 2021)

## Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken wurden als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum

A handwritten signature in black ink, consisting of stylized, cursive letters.

Unterschrift