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, 30. 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 Zuglänge) 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 Fahrplantreue, 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 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.

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen **2**

2.1 Aufbau des Eisenbahn-Betriebs- und Experimentierfelds 2

2.2 Aufbau der *MySQL*-Datenbank 3

2.3 Ziele und Prioritätssetzung der Fahrzeugsteuerung 3

2.4 Fahrdynamik 4

2.5 Aufbau des Projekts 4

3 Ablauf der Fahrzeugsteuerung **6**

3.1 Einlesen von statischen und mehrfach verwendeten Daten aus der *MySQL*-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 Änderungen der Fahrstraße ermitteln 17

3.7 Neukalibrierung der Fahrzeugposition 18

3.8 Ermittlung von neuen Fahrzeugen im eingleisigen Netz 19

3.9 Umgang mit Fehlermeldungen von Fahrzeugen 20

4 Berechnung des Fahrtverlaufs **22**

4.1 Ermittlung der Start- und Endpositionen der Infra-Abschnitte unter Berücksichtigung der Zuglänge 22

4.2 Berechnung des Fahrtverlaufs bei einer Beschleunigung auf die maximal mögliche Geschwindigkeit 26

4.3 Konvertierung der *$keyPoints* in Echtzeitdaten 27

4.4 Überprüfung des Fahrtverlaufs auf Geschwindigkeitsüberschreitungen . 33

4.5 Neuberechnung des Fahrtverlaufs unter Berücksichtigung der Geschwindigkeitsüberschreitungen 35

4.6 Einhaltung der Mindestzeit auf einer Beharrungsfahrt 35
4.7 Berücksichtigung der Ankunftszeit bei der Berechnung des Fahrtverlaufs 39
4.8 Berücksichtigung der exakten Ankunftszeit bei der Berechnung des Fahrtverlaufs 42
4.9 Einleitung einer Gefahrenbremsung 45

5 Beispielrechnung eines Fahrtverlaufs im EBuEf 47

6 Visualisierung der Fahrtverläufe 51

7 Formeln 53
7.1 Formeln für gleichmäßig beschleunigte Bewegungen 53
7.2 Formeln für gleichförmige Bewegungen 55

8 Fazit 57
8.1 Zusammenfassung der Ergebnisse 57
8.2 Komplikationen bei dem Betrieb der Fahrzeugsteuerung im EBuEf . . . 57
8.2.1 Einhaltung der Zielposition 57
8.2.2 Ermittlung der Fahrstraßen 58
8.2.3 Kalibrierung der Position 58
8.3 Möglichkeiten für eine Weiterentwicklung der Fahrzeugsteuerung 58

A Anhang 60
A.1 fahrzeugsteuerung.php 60
A.2 functions.php 69
A.3 functions_fahrtverlauf.php 97
A.4 functions_math.php 145
A.5 functions_cache.php 146
A.6 functions_db.php 153
A.7 global_variables.php 159
A.8 speed_over_position.m 160

Abbildungsverzeichnis

1 Schienennetz des EBuEfs 2
2 Aufbau der Dateistrukturen 5
3 Ablauf der Fahrzeugsteuerung 6
4 Positionsbestimmung bei einem Fahrtrichtungswechsel 16
5 Ablaufplan der Fahrtverlaufsberechnung 24
6 Infra-Abschnitte und die zugehörigen Höchstgeschwindigkeiten 26
7 Infra-Abschnitte und die zugehörigen Höchstgeschwindigkeiten unter Berücksichtigung der Fahrzeuglänge 27
8 Fahrtverlaufsberechnung (1. Iterationsschritt) 29
9 Fahrtverlaufsberechnung (2. Iterationsschritt) 36
10 Fahrtverlaufsberechnung (3. Iterationsschritt) 36
11 Fahrtverlaufsberechnung (4. Iterationsschritt) 37
12 Einteilung des Fahrtverlaufs in *$subsections* 38
13 Fahrtverlauf unter Einhaltung der Mindestzeit 40
14 Fahrtverlauf mit reduzierter Geschwindigkeit unter Einhaltung der Ankunftszeit 41
15 Fahrtverlauf vor der Anpassung der exakten Ankunftszeit 42
16 Fahrtverlauf nach der Anpassung der exakten Ankunftszeit 44
17 Ergebnis der Fahrtverlaufsermittlung 45
18 Fahrtverlauf am Beispiel von der Fahrt von XAB nach XZO 48

Tabellenverzeichnis

1 Beschreibung der wichtigsten Tabellen der *MySQL*-Datenbank 3
2 Aufbau eines Arrays in *next_betriebsstellen_data* 10
3 Aufbau des *zeiten*-Arrays in *next_betriebsstellen_data* 11
4 Aufbau eines Eintrags aus dem *$allTimes*-Array 15
5 Verhalten eines Fahrzeugs nach dem Erreichen des Ziels 16
6 Übersicht der Fehlermeldungen 21
7 Beschreibung der verwendeten Variablen für die Fahrtverlaufsberechnung 23
8 Exemplarische Infra-Abschnitte 23
9 Exemplarische Zugdaten 25
10 Aufbau des *$subsection*-Arrays 39
11 Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung (vor der Anpassung der exakten Ankunftszeit) 43
12 Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung (nach der Anpassung der exakten Ankunftszeit) 44
13 *$keyPoints* am Beispiel von der Fahrt von XAB nach XZO 47
14 Fahrtverlauf am Beispiel von der Fahrt von XAB nach XZO 49

Code-Beispiele

1 Initialisierung der Cache Variablen *(fahrzeugsteuerung.php)* 7
2 Ermittlung der Real- und Simulationszeit *(fahrzeugsteuerung.php)* . . . 8
3 *getCalibratedPosition*() (*functions_db.php*) 19
4 *showErrors*() (*functions.php*) 21
5 *getVMaxBetweenTwoPoints*() (*functions_fahrtverlauf.php*) 28
6 *createTrainChanges*() (*functions_fahrtverlauf.php*) 30
7 *checkIfTrainIsToFastInCertainSections*() (*functions_fahrtverlauf.php*) . 34
8 *safeTrainChangeToJSONFile*() (*functions_fahrtverlauf.php*) 51
9 *getBrakeDistance*() (*functions_math.php*) 54
10 *getBrakeTime*() (*functions_math.php*) 55
11 *getTargetBrakeSpeedWithDistanceAndStartSpeed*() (*functions_math.php*) 55
12 *distanceWithSpeedToTime*() (*functions_math.php*) 56
13 *calculateDistanceforSpeedFineTuning*() (*functions_math.php*) 56

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 2 km/h-Schritten und bei Beharrungsfahrten in regelmäßigen Distanz-Intervallen in Abhängigkeit von der Simulationszeit.

Fahrstraße Beschreibt den Weg, der für das Fahrzeug durch die Stellung der Weichen vorgegeben ist.[1]

Fahrtverlauf Der Fahrtverlauf beschreibt die Positionen, Zeiten und Geschwindigkeiten für alle Beschleunigungs- und Bremsvorgänge und Beharrungsfahrten eines Fahrzeugs von der aktuellen Position bis zum nächsten Halt.

Realzeit Die Realzeit beschreibt die Zeit des Rechners/Servers, auf dem die Fahrzeuzgsteuerung 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.[2]

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

[1] Maschek (2018, S. 114)
[2] 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.[3]

Für die Dokumentation 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 Fahrtverlaufsberechnung 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 mithilfe der in Kapitel 7 hergeleiteten Formeln auf die Korrektheit 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.

[3] *EBuEf: Eisenbahn-Betriebs- und Experimentierfeld Berlin* (2021)

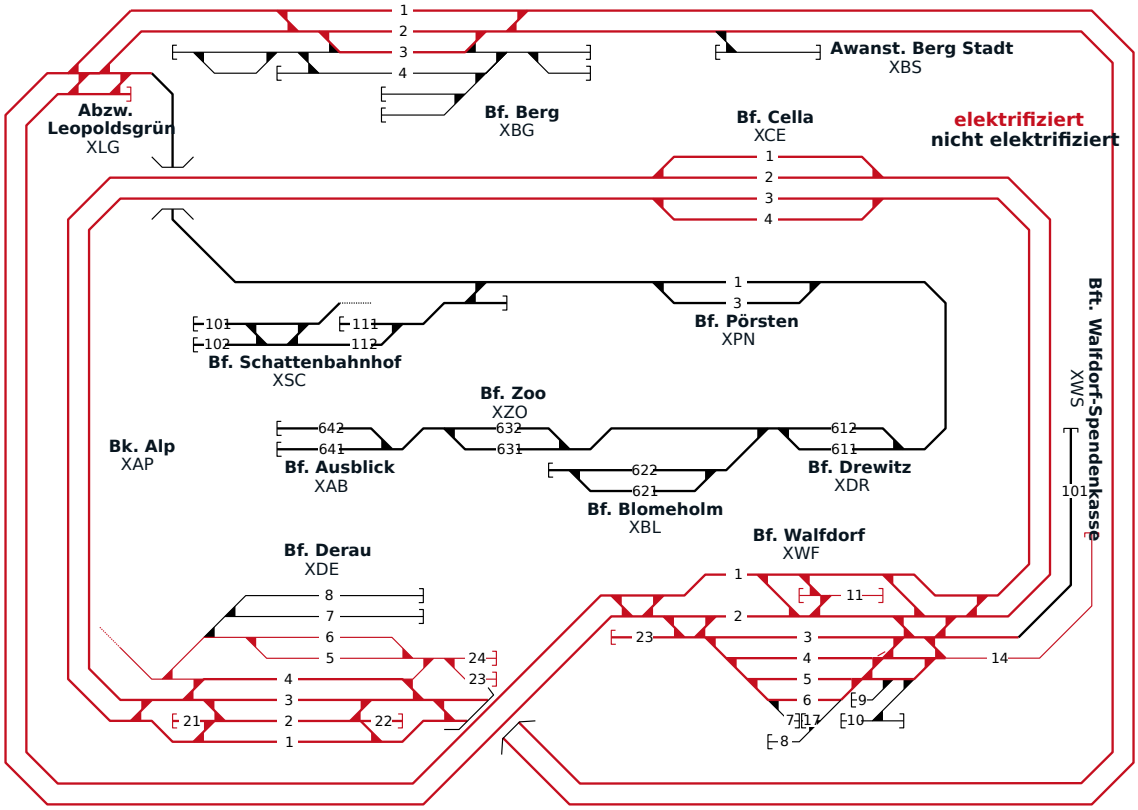

Abbildung 1: Schienennetz des EBuEfs (Quelle: www.ebuef.de/das-betriebsfeld/stellwerke; Letzter Zugriff am: 4. September 2021)

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 wird das eingleisige Netz schwarz und die zweigleisige Hauptstrecke rot dargestellt. Das eingleisige Netz ist in Infrastrukturabschnitte (Infra-Abschnitte), welche mit Blockstrecken vergleichbar sind und eine Zugfolge im festen Raumabstand ermöglichen, eingeteilt.[4] 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.[5] Zudem sind in der Datenbank alle Informationen über die Infrastruktur gespeichert. Für die Fahrzeugsteuerung essenziell sind dabei die aktuellen Signalbegriffe

[4] Pachl (2021, S. 7, 42)
[5] *RailCom - DCC-Rückmeldeprotokoll* (2019)

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

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

aller Signale und die Längen der Infra-Abschnitte.[6]

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 kleinstmöglichste Ge-

[6] *EBuEf: Eisenbahn-Betriebs- und Experimentierfeld Berlin* (2021)

schwindigkeit, 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 als der maximal zulässigen Geschwindigkeit zu fahren, wird diese 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 bei 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.[7]

Die Widerstandskräfte setzen sich aus dem Streckenwiderstand, dem Fahrzeugwiderstand und dem Anfahrwiderstand zusammen und lassen sich mit den gegebenen Daten des EBuEfs nicht vollständig berechnen.[8] Aus diesem Grund werden die Widerstandskräfte bei der Fahrzeugsteuerung nicht berücksichtigt und die Auslaufphase, welche nur von den Widerstandskräften abhängig ist, wird bei der Fahrtverlaufsberechnung vernachlässigt.

2.5 Aufbau des Projekts

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

[7] Pachl (2021, S. 23 ff.)
[8] Pachl (2021, S. 25 ff.)

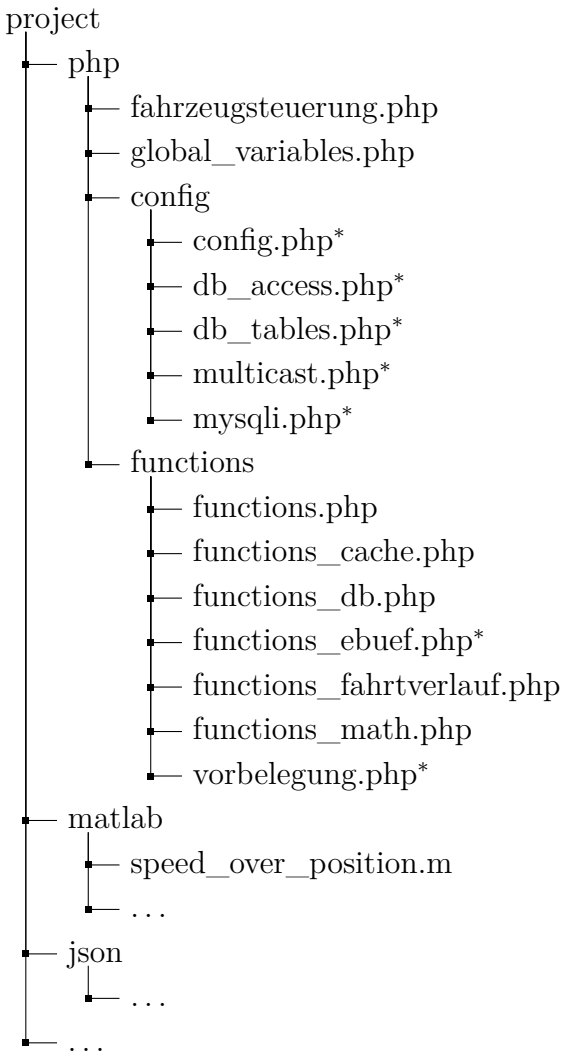

Abbildung 2: Aufbau der Dateistrukturen (Quelle: Eigene Darstellung)

Funktionen mit einem Sternchen (*) markiert. Die für die Fahrzeugsteuerung essenziellen 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.

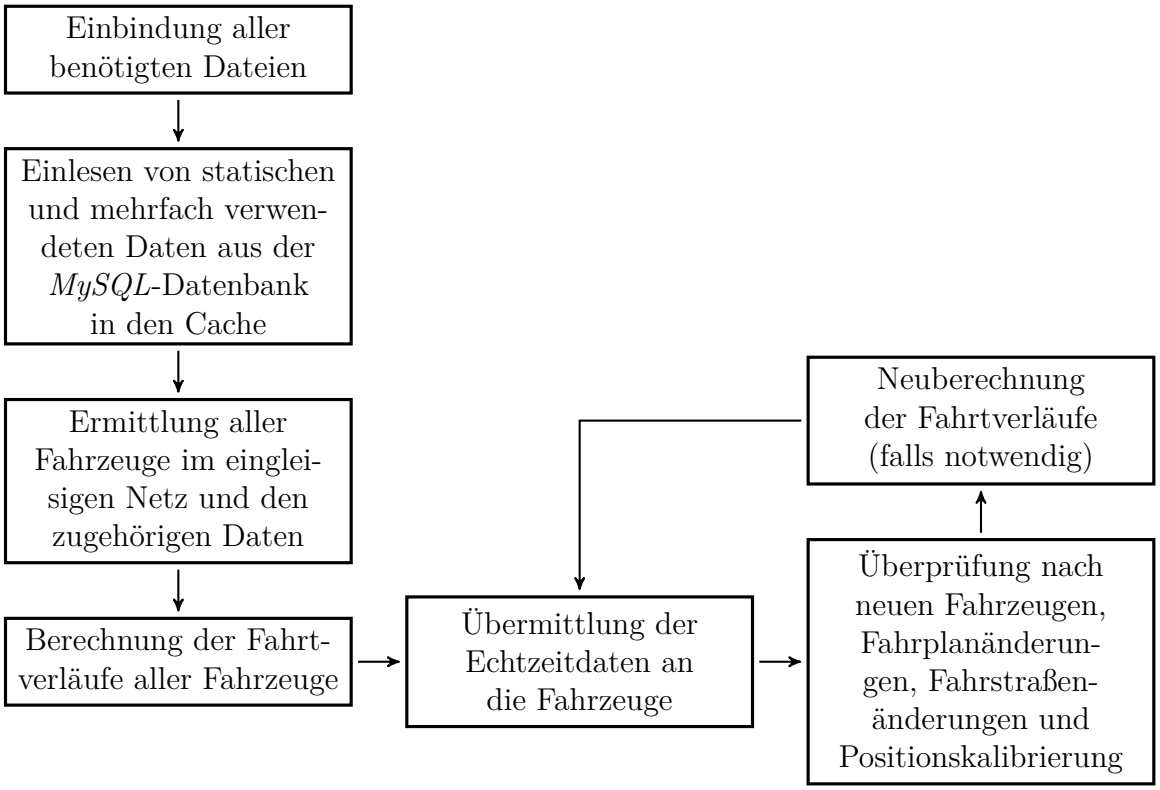

Abbildung 3: Ablauf der Fahrzeugsteuerung (Quelle: Eigene Darstellung)

3 Ablauf der Fahrzeugsteuerung

Damit die Fahrzeugsteuerung gestarten 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 ist.

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). Beispielhaft 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).

```php
// Statische Daten einlesen
$cacheInfranachbarn = createCacheInfranachbarn();
$cacheInfradaten = createCacheInfradaten();
$cacheSignaldaten = createCacheSignaldaten();
$cacheInfraLaenge = createcacheInfraLaenge();
$cacheHaltepunkte = createCacheHaltepunkte();
$cacheZwischenhaltepunkte = createChacheZwischenhaltepunkte();
$cacheInfraToGbt = createCacheInfraToGbt();
$cacheGbtToInfra = createCacheGbtToInfra();
$cacheFmaToInfra = createCacheFmaToInfra();
$cacheInfraToFma = array_flip($cacheFmaToInfra);
$cacheFahrplanSession = createCacheFahrplanSession();
$cacheSignalIDToBetriebsstelle = createCacheToBetriebsstelle();
$cacheFahrzeugeAbschnitte = createCacheFahrzeugeAbschnitte();
$cacheIDTDecoder = createCacheDecoderToAdresse();
$cacheDecoderToID = array_flip($cacheIDTDecoder);
// Werden in der Funktion getAllTrains() initialisiert
$cacheAdresseToID = array();
$cacheIDToAdresse = array();
```

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

```php
// Real- und Simulationszeit ermitteln
$simulationStartTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->sim_startzeit, "simulationszeit", null, array("outputtyp"=>"h:i:s")), "simulationszeit", null, array("inputtyp"=>"h:i:s"));
$simulationEndTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->sim_endzeit, "simulationszeit", null, array("outputtyp"=>"h:i:s")), "simulationszeit", null, array("inputtyp"=>"h:i:s"));
$simulationDuration = $cacheFahrplanSession->sim_endzeit - $cacheFahrplanSession->sim_startzeit;
$realStartTime = time();
$realEndTime = $realStartTime + $simulationDuration;
$timeDifference = $simulationStartTimeToday - $realStartTime;
```

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

geändert und im zweiten Schritt mit der Realzeit verglichen (Code-Beispiel 2).

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 EBuEfs 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.

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 Abfahrtszeiten wird die Funktion *getFahrplanzeiten*()* (*functions_ebuef.php*) aufgerufen, welche als Parameter den Namen der

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

Tabelle 2: Aufbau eines Arrays in *next_betriebsstellen_data*

Betriebsstelle und die Zug-ID übergeben bekommt. Die zurückgegebenen Daten werden unter dem Eintrag *zeiten* abgespeichert und um 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 mithilfe 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 Ausfahrsignal 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.

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

Tabelle 3: Aufbau des *zeiten*-Arrays in *next_betriebsstellen_data*

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 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 *checkIfFahrstrasseIsCorrrect*() (*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 *$nextBetriebsstelleIndex* abgespeichert. Sollten diese Kriterien auf keine der nächsten Betriebsstellen zutreffen, wird in einer zweiten *for*-Schleife nach denselben Kriterien (außer dem des fahrplanmäßigen Halts) nach einer Betriebsstelle gesucht und sobald eine Betriebsstelle gefunden wurde, wird die Schleife abgebrochen und der Index der Betriebsstelle unter der Variablen *$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: $v > 0\ km/h$, 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 welchem 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 werden soll (Länge

des Infra-Abschnitts).

Neben den Informationen zur Position müssen ebenfalls die Informationen der 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 angefahrene 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 der Simulationszeit gleichgesetzt. 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 Abfahrtzeit 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 dem Fahrzeug ein Wendeauftrag nach dem Erreichen des Ziel-Infra-Abschnitts 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 und die aktuelle Geschwindigkeit des Fahrzeugs 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 die 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. Dieser Wert 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 *$allTimes* alle Echtzeitdaten enthalten. Das Array beinhaltet für jedes Fahrzeug wiederum ein Array, welches unter der Adresse des Fahrzeugs abgespeichert ist, und alle Echtzeitdaten eines Fahrzeugs enthält. Der Aufbau eines Arrays mit Echtzeitdaten ist in Tabelle 4 dargestellt. In einer *while*-Schleife wird über alle Einträge des *$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*).

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

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

Sollte das der Fall sein, wird die neue Geschwindigkeit dem Fahrzeug mit der Funktion *sendFahrzeugbefehl*()* (*functions_ebuef.php*) übergeben. 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 aufsummierten Längen der Infra-Abschnitte zum ersten Mal die Fahrzeuglänge (zuzüglich des Abstands

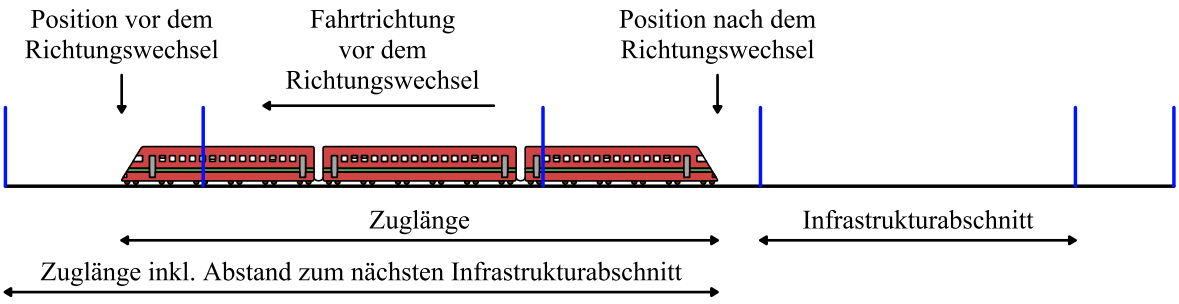

Abbildung 4: Positionsbestimmung bei einem Fahrtrichtungswechsel (Quelle: Eigene Darstellung)

	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

bis zum Ende des Infra-Abschnitts) überschreiten, 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 inklusive dem Abstand bis zum Ende des Infra-Abschnitts, kann die neue Position nicht ermittelt werden und dem Fahrzeug wird eine Fehlermeldung übergeben. 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 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. Dabei wird zwischen vier Fällen (siehe Tabelle 5) unterschieden. Für die Überprüfung, ob sich der Fahr-

plan 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*), *checkIfFahrstrasseIsCorrrect*() (*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 beschrieben 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 Änderungen der Fahrstraße ermitteln

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 *$timeCheckFahrstrasseInterval* (*fahrzeugsteuerung.php*) festgelegt werden und ist standardgemäß auf 3 Sekunden festgelegt. Bei der Ermittlung und dem Vergleich der Fahrstraße wird für jedes Fahrzeug die Funktion *compareTwoNaechsteAbschnitte*() (*functions.php*) aufgerufen. Innerhalb dieser Funktion wird die in Kapitel 3.3 erläutere Funktion *calculateNextSections*() (*functions.php*) aufgerufen, mit dem Unterschied, dass die ermittelten nächsten Infra-Abschnitte inklusive der Längen und zulässigen Höchstgeschwindigkeiten nicht dem Fahrzeug hinterlegt, sondern lokal in der Funktion gespeichert werden. Damit die ermittelten Daten für ein Fahrzeug berechnet werden können, ohne dass diese in dem Array *$allUsedTrains* 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 *checkIfFahrstrasseIsCorrrect*() (*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 präzise Fahrzeugsteuerung ist die Kenntnis über die aktuellen Positionen der Züge essenziell. Damit Ungenauigkeiten ausgeglichen werden können, muss die Position im Laufe des Betriebs kalibriert werden. 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 und die aktuellen Zeit (Realzeit) in die Datenbank, sobald der Zugkopf den Abschnitt befährt.

Für jedes Fahrzeug, welches durch die Übermittlung der Echtzeitdaten in einen neuen Infra-Abschnitt einfährt und seit der Einfahrt in den Infra-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 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. Die relative Positi-

```php
// Kalibriert die Position des Fahrzeugs anhand der Daten in der Tabelle
// 'fahrzeuge_abschnitte' neu.
function getCalibratedPosition ($id, $speed) {
  global $cacheFahrzeugeAbschnitte;
  $DB = new DB_MySQL();
  $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` = $id")[0];
  unset($DB);
  if (in_array($id, array_keys($cacheFahrzeugeAbschnitte))) {
    if ($positionReturn->unixtimestamp == $cacheFahrzeugeAbschnitte[$id]['
        ↪ unixtimestamp']) {
      return array('possible' => false);
    }
  }
  $timeDiff = time() - $positionReturn->unixtimestamp;
  $position = ($speed / 3.6) * $timeDiff;
  return array('section' => $positionReturn->infra_id, 'position' => $position)
      ↪ ;
}
```

Code-Beispiel 3: *getCalibratedPosition*() (*functions_db.php*)

on innerhalb des Infra-Abschnitts ergibt sich aus der aktuellen Geschwindigkeit des Fahrzeugs und die Differenz der Zeit zwischen dem Einfahren in den Infra-Abschnitt und der aktuellen Zeit.

relative Position = Geschwindigkeit · Zeitdifferenz (aktuelle Zeit − 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 befanden, auch alle Fahrzeuge, die nach dem Programmstart hinzugefügt wurden. 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 Funk-

tion werden alle Fahrzeuge eingelesen, denen in der *fma*-Tabelle aus der Datenbank ein Infra-Abschnitt zugeordnet ist und mit dem Array *$allTrainsOnTheTrack* verglichen. Fahrzeugadressen, die nicht in dem Array hinterlegt sind, werden in dem Rückgabe-Array unter dem Eintrag *new* zurückgegeben und alle Fahrzeugadressen, die in dem Array enthalten sind, aber bei dem Aufruf der Funktion keinem Infra-Abschnitt zugeordnet sind, werden in dem Rückgabe-Array unter dem Eintrag *removed* zurückgegeben. Nach dem Aufruf der Funktion werden für alle neuen Fahrzeuge die Funktionen *prepareTrainForRide*() (*functions.php*), *addStopsectionsForTimetable*() (*functions.php*), *calculateNextSections*() (*functions.php*), *checkIfTrainReachedHaltepunkt*() (*functions.php*), *checkIfFahrstrasseIsCorrrect*() (*functions.php*) und *calculateFahrtverlauf*() (*functions.php*) aufgerufen (siehe Kapitel 3.3 und 3.4). Alle entfernten Fahrzeuge werden aus dem Array *$allUsedTrains* entfernt und somit nicht mehr von der Fahrzeugsteuerung beachtet.

3.9 Umgang mit Fehlermeldungen von Fahrzeugen

Wenn es bei einem Fahrzeug zu einem Steuerungskonflikt kommt, 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*() (*functions.php*) aufgerufen, welche die Fehlermeldungen aller Fahrzeuge ausgibt (Code-Beispiel 4). Für das Beheben einer Fehlermeldung muss das Fahrzeug händisch vom Schienennetz genommen und gewartet werden. Hat die Fahrzeugsteuerung dies registriert, kann das Fahrzeug zurück 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 Fahhrzeugsteuerung implementiert sind.

```php
// Gibt die vorhanden Fehlermeldungen für alle Fahrzeuge an.
function showErrors() {

  global $allUsedTrains;
  global $trainErrors;

  $foundError = false;
  echo "Hier werden für alle Züge mögliche Fehler angezeigt:\n\n";

  foreach ($allUsedTrains as $trainIndex => $trainValue) {
    if (sizeof($trainValue['error']) != 0) {
      $foundError = true;
      echo 'Zug ID: ', $trainValue['id'], "\n";
      $index = 1;

      foreach ($trainValue['error'] as $error) {
        echo "\t", $index, ". Fehler:\t", $trainErrors[$error], "\n";
        $index++;
      }

      echo "\n";
    }
  }

  if (!$foundError) {
    echo "Keiner der Züge hat eine Fehlermeldung.\n";
  }
}
```

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

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 sogenannten *$keyPoints*, welche in einem Array die Start- und Zielgeschwindigkeit (*speed_0* und *speed_1*), die Start- und Endposition (*position_0* und *position_1*) und die Start- und Endzeit (*time_0* und *time_1*) der einzelnen Beschleunigungen bzw. Verzögerungen abspeichern. Für die Überprüfung, ob ein Fahrzeug die zulässige Höchstgeschwindigkeit in einem Infra-Abschnitt überschreitet sowie die exakte Positionsbestimmung, werden die *$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 *$allUsedTrains*-Array, Start- und Endzeit der Fahrt (*$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.

4.1 Ermittlung der Start- und Endpositionen der Infra-Abschnitte unter Berücksichtigung der Zuglänge

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 EBuEf nicht. Dafür wurden exemplarisch die Zugdaten, welche in Tabelle 9 definiert wurden, verwendet.

Die zuvor ermittelten nächsten Infra-Abschnitte inklusive derer Längen und zulässigen Höchstgeschwindigkeiten müssen für die Berechnung des Fahrtverlaufs angepasst werden, da ein Fahrzeug erst beschleunigen darf, wenn das komplette Fahr-

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_lenghts (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
$trainPositionChange (Array)	Alle absoluten Positionen des Fahrtverlaufs
$trainSpeedChange (Array)	Alle Geschwindigkeiten des Fahrtverlaufs

Tabelle 7: Beschreibung der verwendeten Variablen für die Fahrtverlaufsberechnung

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

Tabelle 8: Exemplarische Infra-Abschnitte

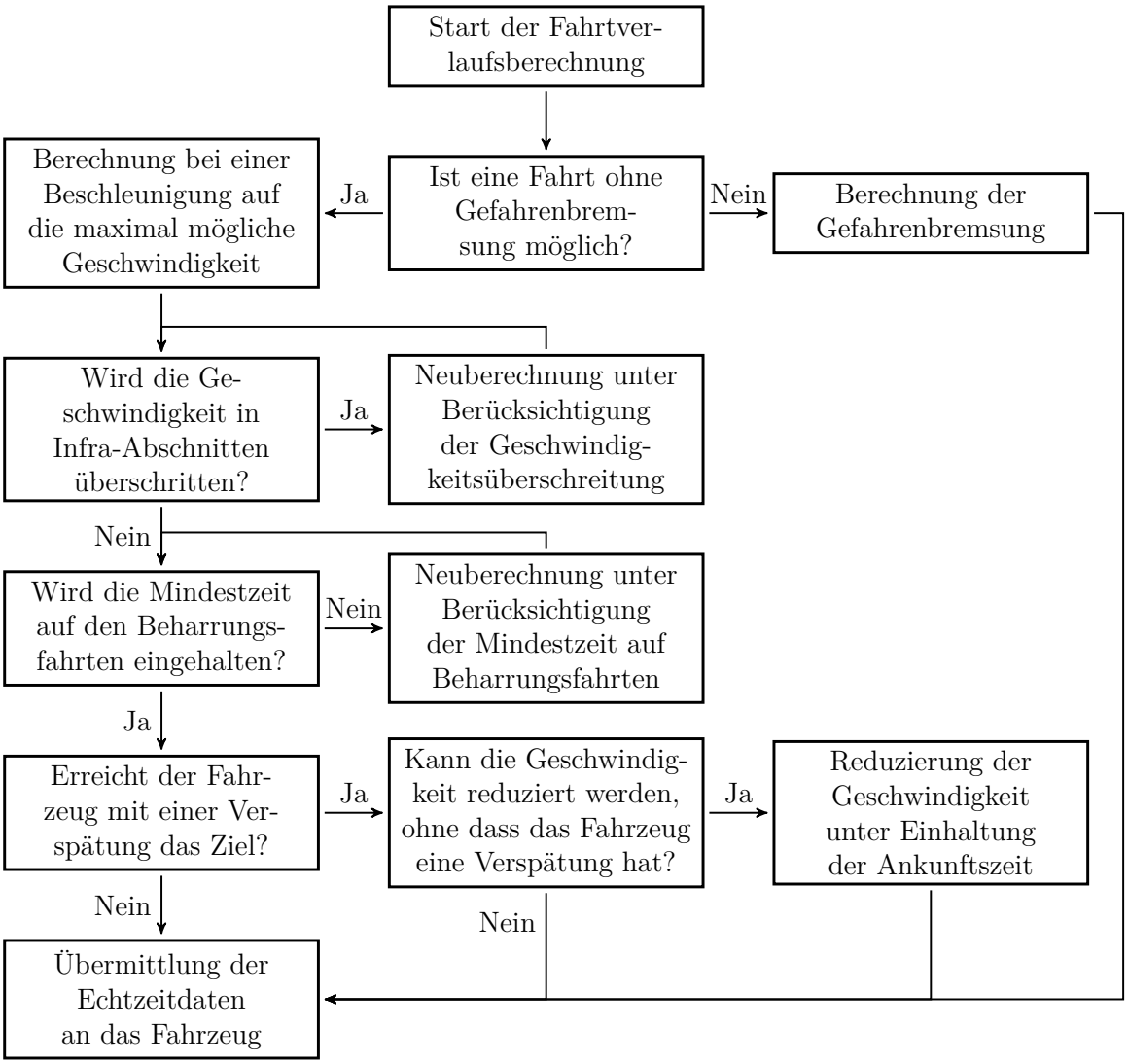

Abbildung 5: Ablaufplan der Fahrtverlaufsberechnung (Quelle: Eigene Darstellung)

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

Tabelle 9: Exemplarische Zugdaten

zeug 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. Der letzte Infra-Abschnitt ist der Infra-Abschnitt mit der ID 1010 und einer Länge von ebenfalls 290 m.

Bei der Berücksichtigung der Fahrzeuglänge wird mit einer *for*-Schleife über alle Infra-Abschnitte iteriert und die Zuglänge auf die Länge das 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

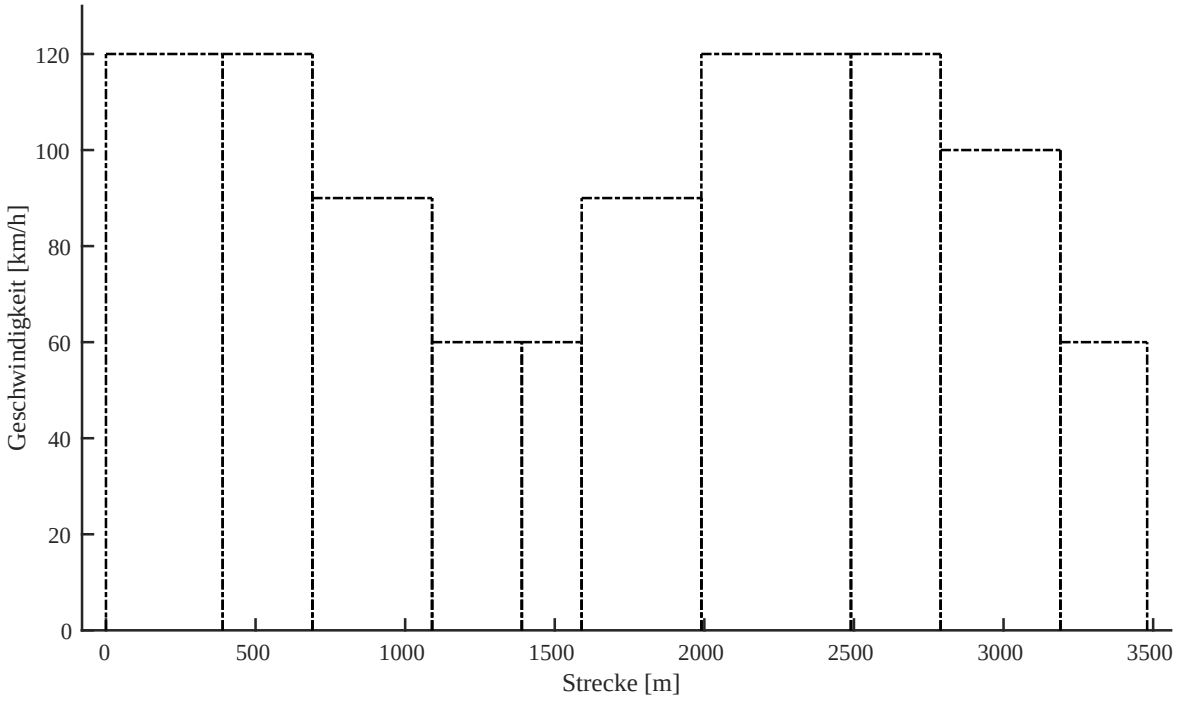

Abbildung 6: Infra-Abschnitte und die zugehörigen Höchstgeschwindigkeiten (Quelle: Eigene Darstellung)

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 Umwandlung ist essenziell für die Überprüfung, in welchem Infra-Abschnitt ein Fahrzeug sich aktuell befindet. Die neu berechneten Infra-Abschnitte werden in der Abbildung 7 rot dargestellt und beschreiben die maximale Geschwindigkeit, die ein Fahrzeug an der jeweiligen Position fahren darf.

4.2 Berechnung des Fahrtverlaufs 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 mithilfe 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

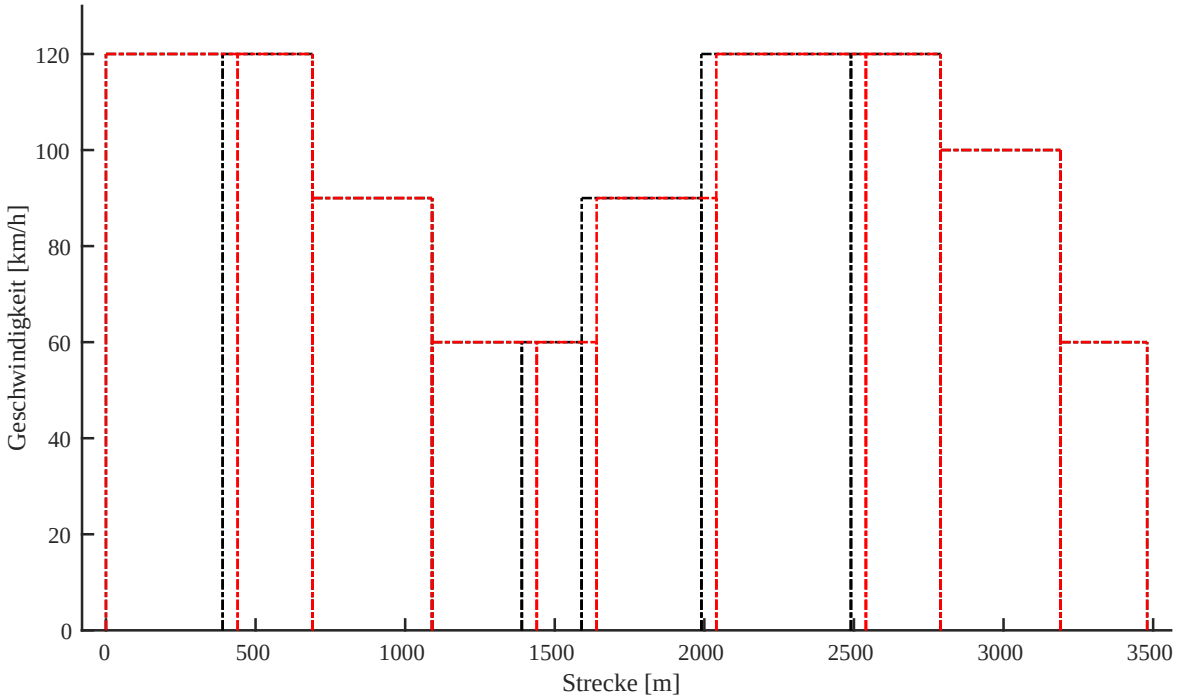

Abbildung 7: Infra-Abschnitte und die zugehörigen Höchstgeschwindigkeiten unter Berücksichtigung der Fahrzeuglänge (Quelle: Eigene Darstellung)

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.

Durch die gegebene Startgeschwindigkeit und die größtmögliche Geschwindigkeit wird ein erster Fahrtverlauf berechnet, wobei zwei *$keyPoints* erzeugt werden. In der Darstellung 8 ist der erste Iterationsschritt der Fahrtverlaufsermmittlung abgebildet, wobei die beiden *$keyPoints* in Echtzeitdaten umgewandelt wurden (siehe Kapitel 4.3).

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

```php
// Ermittelt die maximale Geschwindigkeit zwischen zwei Positionen
function getVMaxBetweenTwoPoints(float $distance, int $v_0, int $v_1, int $id) {

  global $verzoegerung;
  global $globalFloatingPointNumbersRoundingError;

  $v_max = array();

  for ($i = 0; $i <= 120; $i = $i + 10) {
    if ((getBrakeDistance($v_0, $i, $verzoegerung) + getBrakeDistance($i, $v_1, $verzoegerung)) < ($distance + $globalFloatingPointNumbersRoundingError)) {
      array_push($v_max, $i);
    }
  }

  if (sizeof($v_max) == 0) {
    if ($v_0 == 0 && $v_1 == 0 && $distance > 0) {
      echo 'Der zug müsste langsamer als 10 km/h fahren, um das Ziel zu erreichen.';
    } else {
      emergencyBreak($id);
    }
  } else {
    if ($v_0 == $v_1 && max($v_max) < $v_0) {
      $v_max = array($v_0);
    }
  }

  return max($v_max);
}
```

Code-Beispiel 5: *getVMaxBetweenTwoPoints()* (*functions_fahrtverlauf.php*)

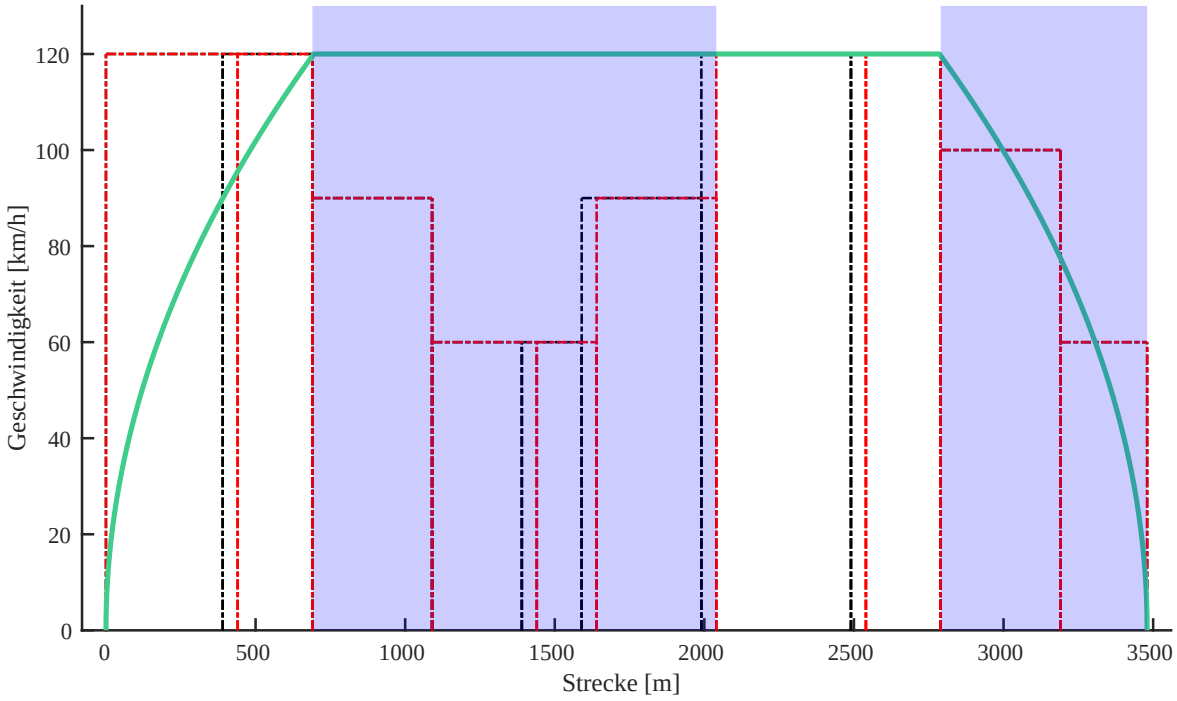

Abbildung 8: Fahrtverlaufsberechnung (1. Iterationsschritt) (Quelle: Eigene Darstellung)

(*$returnTrainSection*) und ob eine Geschwindigkeitsveränderung vorliegt (*$returnIsSpeedChange*) an.

Im ersten Schritt wird mit einer *for*-Schleife über alle *$keyPoints* iteriert (Code-Beispiel 6; Zeile 24 – 57). Für jeden *$keyPoint* wird in einer zweiten *for*-Schleife in 2 km/h-Schritten von der Startgeschwindigkeit (*speed_0*) bis zur Zielgeschwindigkeit (*speed_1*) iteriert und für jede Geschwindigkeitsveränderung wird die benötigte Zeit und Strecke errechnet, die Geschwindigkeit gespeichert und dass es sich um eine Geschwindigkeitsanpassung handelt (Code-Beispiel 6; Zeile 32 – 39).

Im Anschluss werden die Echtzeitdaten für die Beharrungsfahrten berechnet. Da Beharrungsfahrten immer zwischen zwei *$keyPoints* existieren, wird für jeden *$keyPoint* die Beharrungsfahrt zum nächsten *$keyPoint* berechnet und der letzte *$keyPoint* wird ausgelassen (Code-Beispiel 6; Zeile 41 – 56). Das Distanzintervall, in welchem die Echtzeitdaten für Beharrungsfahrten ermittelt werden, kann über die globale Variable *$globalDistanceUpdateInterval* (*global_variables.php*) festgelegt werden. Der Standardwert wurde auf 1 m festgelegt.

Für die Ermittlung der zugehörigen Infra-Abschnitte und den relativen Positionen in den Infra-Abschnitten, wird über alle absoluten Positionen (*$returnTrainPositionChange*) iteriert und in einer zweiten *foreach*-Schleife über alle Anfangspositionen der

Infra-Abschnitte (*$cumulativeSectionLengthStart*). Sobald sich eine absolute Position in einem Infra-Abschnitt befindet, wird der Infra-Abschnitt inklusive der relativen Position ermittelt und die zweite *foreach*-Schleife wird mit dem Befehl *break* abgebrochen (Code-Beispiel 6; Zeile 63 – 96).

```
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 $globalDistanceUpdateInterval;
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); $i++) {
27     array_push($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
32     $itDir = ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) ? 2 : -2;
33
34     for ($j = ($keyPoints[$i]["speed_0"] + $itDir); $j <= $keyPoints[$i]["
```

```
      ↪ speed_1"]; $j = $j + $itDir) {
35      array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
            ↪  getBrakeDistance(($j - $itDir), $j, $verzoegerung)));
36      array_push($returnTrainSpeedChange, $j);
37      array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
            ↪ getBrakeTime(($j - $itDir), $j, $verzoegerung))));
38      array_push($returnIsSpeedChange, true);
39    }
40
41    // Überprüft, ob nach dem $keyPoint eine Beharrungsfahrt stattfindet
42    if ($i != array_key_last($keyPoints)) {
43      // Ermittelt für die Strecke zwischen zwei $keyPoints die Echtzeitdaten
44      // der Zeit, Geschwindigkeit und Position
45      $startPosition = $keyPoints[$i]["position_1"];
46      $endPosition = $keyPoints[$i + 1]["position_0"];
47      $speedToNextKeyPoint = $keyPoints[$i]["speed_1"];
48      $timeForOneTimeInterval = distanceWithSpeedToTime($speedToNextKeyPoint,
            ↪ $globalDistanceUpdateInterval);
49
50      for ($position = $startPosition + $globalDistanceUpdateInterval;
            ↪ $position < $endPosition; $position = $position +
            ↪ $globalDistanceUpdateInterval) {
51        array_push($returnTrainPositionChange, $position);
52        array_push($returnTrainSpeedChange, $speedToNextKeyPoint);
53        array_push($returnTrainTimeChange, end($returnTrainTimeChange) +
              ↪ $timeForOneTimeInterval);
54        array_push($returnIsSpeedChange, false);
55      }
56    }
57  }
58  array_push($returnTrainPositionChange, $keyPoints[array_key_last($keyPoints)
      ↪ ]["position_1"] - getBrakeDistance($keyPoints[array_key_last(
      ↪ $keyPoints)]["speed_0"],$keyPoints[array_key_last($keyPoints)]["
      ↪ speed_1"],$verzoegerung));
59  array_push($returnTrainSpeedChange, $keyPoints[array_key_last($keyPoints)]["
      ↪ speed_0"]);
60  array_push($returnTrainTimeChange, $keyPoints[array_key_last($keyPoints)]["
      ↪ time_0"]);
61  array_push($returnIsSpeedChange, true);
```

```
62
63  if ($onlyPositionAndSpeed) {
64    return array($returnTrainPositionChange, $returnTrainSpeedChange);
65  } else {
66    // Ermittelt die relativen Positionen innerhalb der Infra-Abschnitte
67    // zu den absoluten Positionen
68    foreach ($returnTrainPositionChange as $absolutPositionKey =>
        ↪ $absolutPositionValue) {
69      foreach ($cumulativeSectionLengthStart as $sectionStartKey =>
          ↪ $sectionStartValue) {
70        if ($absolutPositionValue >= $sectionStartValue && $absolutPositionValue
            ↪  < $cumulativeSectionLengthEnd[$sectionStartKey]) {
71          if ($sectionStartKey == $indexCurrentSection && $sectionStartKey ==
              ↪ $indexTargetSection) {
72            $returnTrainRelativePosition[$absolutPositionKey] =
                ↪ $absolutPositionValue + $currentPosition;
73            $returnTrainSection[$absolutPositionKey] = $next_sections[
                ↪ $sectionStartKey];
74          } else if ($sectionStartKey == $indexCurrentSection) {
75            $returnTrainRelativePosition[$absolutPositionKey] =
                ↪ $absolutPositionValue + $currentPosition;
76            $returnTrainSection[$absolutPositionKey] = $next_sections[
                ↪ $sectionStartKey];
77          } else if ($sectionStartKey == $indexTargetSection) {
78            $returnTrainRelativePosition[$absolutPositionKey] =
                ↪ $absolutPositionValue - $sectionStartValue;
79            $returnTrainSection[$absolutPositionKey] = $next_sections[
                ↪ $sectionStartKey];
80          } else {
81            $returnTrainRelativePosition[$absolutPositionKey] =
                ↪ $absolutPositionValue - $sectionStartValue;
82            $returnTrainSection[$absolutPositionKey] = $next_sections[
                ↪ $sectionStartKey];
83          }
84          break;
85        } else if ($absolutPositionKey == array_key_last(
            ↪ $returnTrainPositionChange) && abs($absolutPositionValue -
            ↪ floatval($cumulativeSectionLengthEnd[$sectionStartKey])) <
            ↪ $globalFloatingPointNumbersRoundingError) {
```

```
86            $returnTrainRelativePosition[$absolutPositionKey] =
                  ↪ $cumulativeSectionLengthEnd[$sectionStartKey] -
                  ↪ $sectionStartValue;
87            $returnTrainSection[$absolutPositionKey] = $next_sections[
                  ↪ $sectionStartKey];
88            break;
89          } else {
90            debugMessage("Einer absoluten Position konnte kein Infra-Abschnitt und
                  ↪  keine relative Position in einem Infra-Abschnitt zugeordnet
                  ↪ werden.");
91          }
92        }
93      }
94
95      return array($returnTrainPositionChange, $returnTrainSpeedChange,
            ↪ $returnTrainTimeChange, $returnTrainRelativePosition,
            ↪ $returnTrainSection, $returnIsSpeedChange);
96    }
97  }
```

Code-Beispiel 6: *createTrainChanges*() (*functions_fahrtverlauf.php*)

4.4 Überprüfung des Fahrtverlaufs auf 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 *$faildSections*-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*).

```php
// Überprüft, ob das Fahrzeug in Infra-Abschnitten die zulässige
// Höchstgeschwindigkeit überschreitet
function checkIfTrainIsToFastInCertainSections() {

  global $trainPositionChange;
  global $trainSpeedChange;
  global $cumulativeSectionLengthStartMod;
  global $next_v_max_mod;
  global $indexTargetSectionMod;

  $faildSections = array();

  foreach ($trainPositionChange as $trainPositionChangeKey =>
      ↪ $trainPositionChangeValue) {
    foreach ($cumulativeSectionLengthStartMod as
        ↪ $cumulativeSectionLengthStartKey =>
        ↪ $cumulativeSectionLengthStartValue) {
      if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
        if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
            ↪ $cumulativeSectionLengthStartKey - 1]) {
          array_push($faildSections, ($cumulativeSectionLengthStartKey -1));
        }

        break;
      } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
        if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
          if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
              ↪ $cumulativeSectionLengthStartKey]) {
            array_push($faildSections, $cumulativeSectionLengthStartKey);
          }

          break;
        }
      }
    }
  }

  if (sizeof($faildSections) == 0) {
    return array("failed" => false);
  } else {
    return array("failed" => true, "failed_sections" => array_unique(
        ↪ $faildSections));
  }
}
```

Code-Beispiel 7: *checkIfTrainIsToFastInCertainSections*() (*functions_fahrtverlauf-.php*)

4.5 Neuberechnung des Fahrtverlaufs unter Berücksichtigung der Geschwindigkeitsüberschreitungen

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 mithilfe 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 Abbildungen 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 *$globalTimeOnOneSpeed* in der Datei *global_variables.php* eine Mindestzeit festgelegt werden, die ein Fahrzeug auf einer Beharrungsfahrt mindestens einhalten muss. Ebenfalls kann über die Variablen *$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.

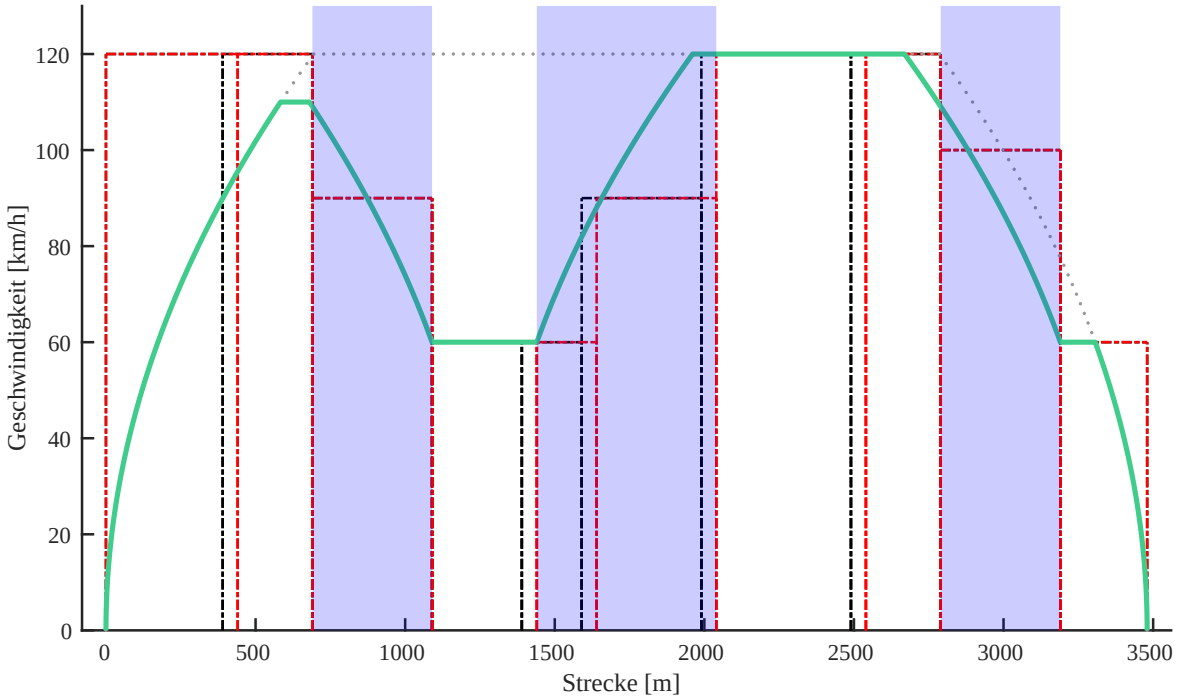

Abbildung 9: Fahrtverlaufsberechnung (2. Iterationsschritt) (Quelle: Eigene Darstellung)

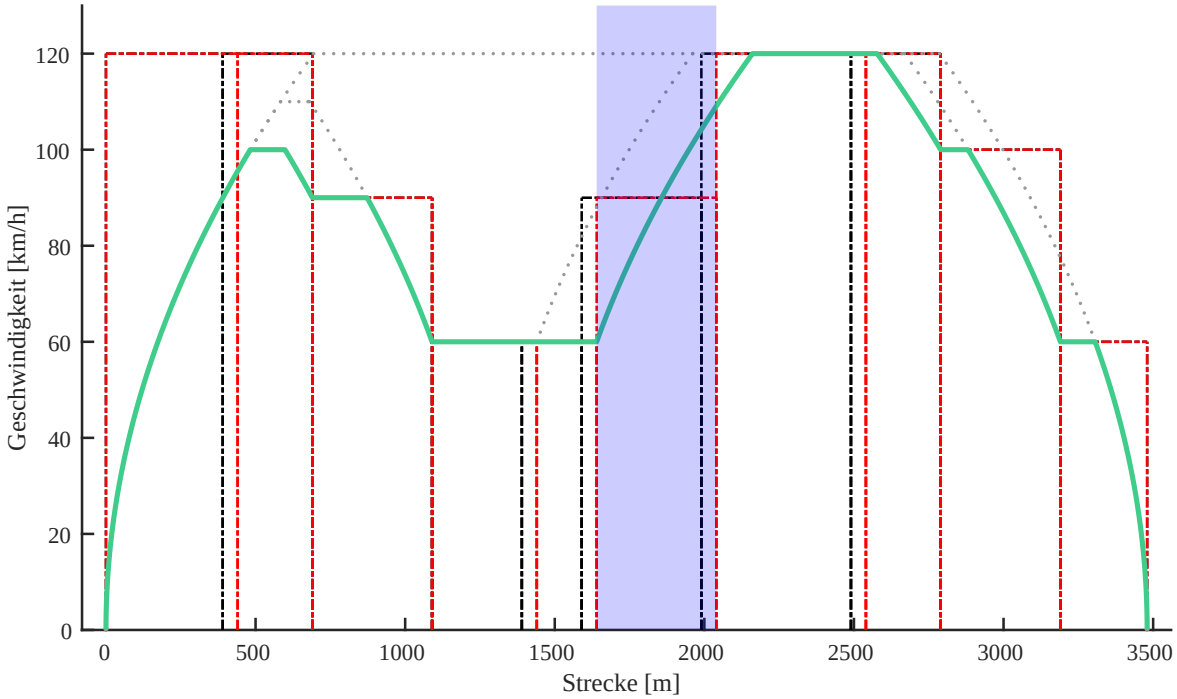

Abbildung 10: Fahrtverlaufsberechnung (3. Iterationsschritt) (Quelle: Eigene Darstellung)

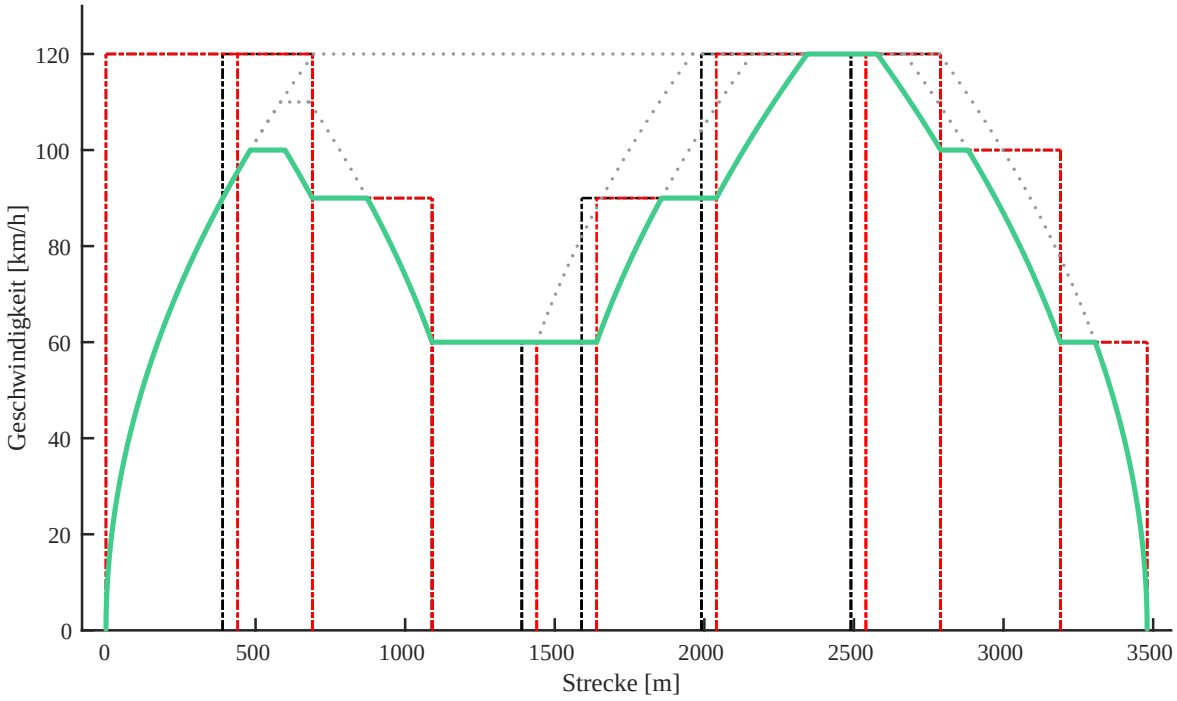

Abbildung 11: Fahrtverlaufsberechnung (4. Iterationsschritt) (Quelle: Eigene Darstellung)

Im Falle einer Fehlermeldung würde das Fahrzeug nicht losfahren bzw. eine Gefahrenbremsung einleiten, falls das Fahrzeug aktuell eine Geschwindigkeit $v > 0\ km/h$ hat.

Wenn auf einem Abschnitt die Mindestzeit nicht eingehalten werden kann, kann eine Beschleunigung später eingeleitet, eine Verzögerung vorzeitiger eingeleitet 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 *$subsection* unterteilt, welche lila bzw. gelb hinterlegt sind. Durch diese Einteilung kann verhindert werden, dass es zu Konflikten kommt. Falls die Beschleunigungen bzw. Verzögerungen so weit nach hinten bzw. nach vorne verschoben werden müssen, kann die maximale Geschwindigkeit auf dieser *$subsection* reduziert werden und die zur Verfügung stehende Strecke vergrößert werden. Wie in Darstellung 12 zu erkennen ist, 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

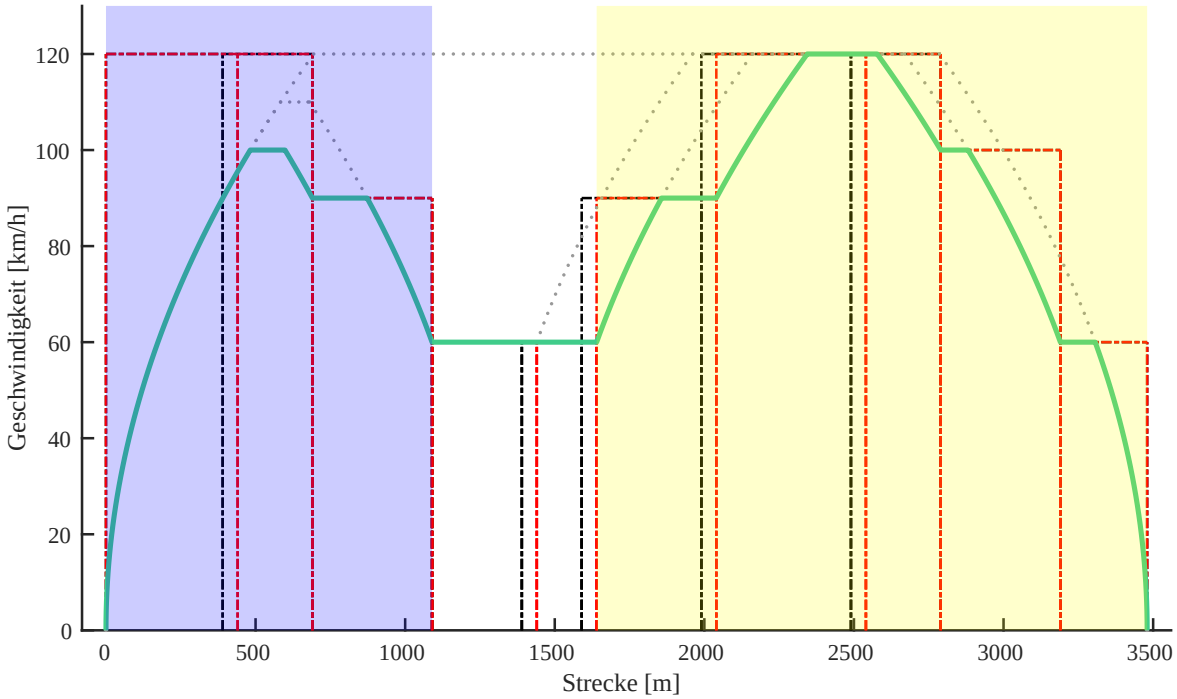

Abbildung 12: Einteilung des Fahrtverlaufs in *$subsections* (Quelle: Eigene Darstellung)

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.

Nachdem die *$subsections* mittels der Funktion *createSubsections*() (*functions_fahrtverlauf.php*) erstellt wurden und mit der Funktion *array_reverse*() in umgekehrter Reihenfolge in dem Array *$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 *$subsections*, bei denen die Mindestzeit für die Beharrungsfahrten nicht eingehalten wird (*failed* == *true*), wird überprüft, ob eine Verschiebung der Beschleunigungen bzw. Verzögerungen möglich ist. Bei der Verschiebung einer Beschleunigung bzw. Verzögerung wird die Differenz zwischen der Mindestzeit einer Beharrungsfahrt (*$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* der Zielgeschwindigkeit des ersten *$keyPoints* zugewiesen. Gleiches geschieht bei der Verzögerung in umgekehrter

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

Tabelle 10: Aufbau des *$subsection*-Arrays

Reihenfolge.

Nach der Verschiebung wird überprüft, ob auf allen Beharrungsfahrten 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 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 den Beharrungsfahrten abgebildet. Für den Fall, dass das Fahrzeug auf einer Geschwindigkeit die Mindestzeit nicht einhält und als Nächstes beschleunigen würde, kann die Beschleunigung 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.

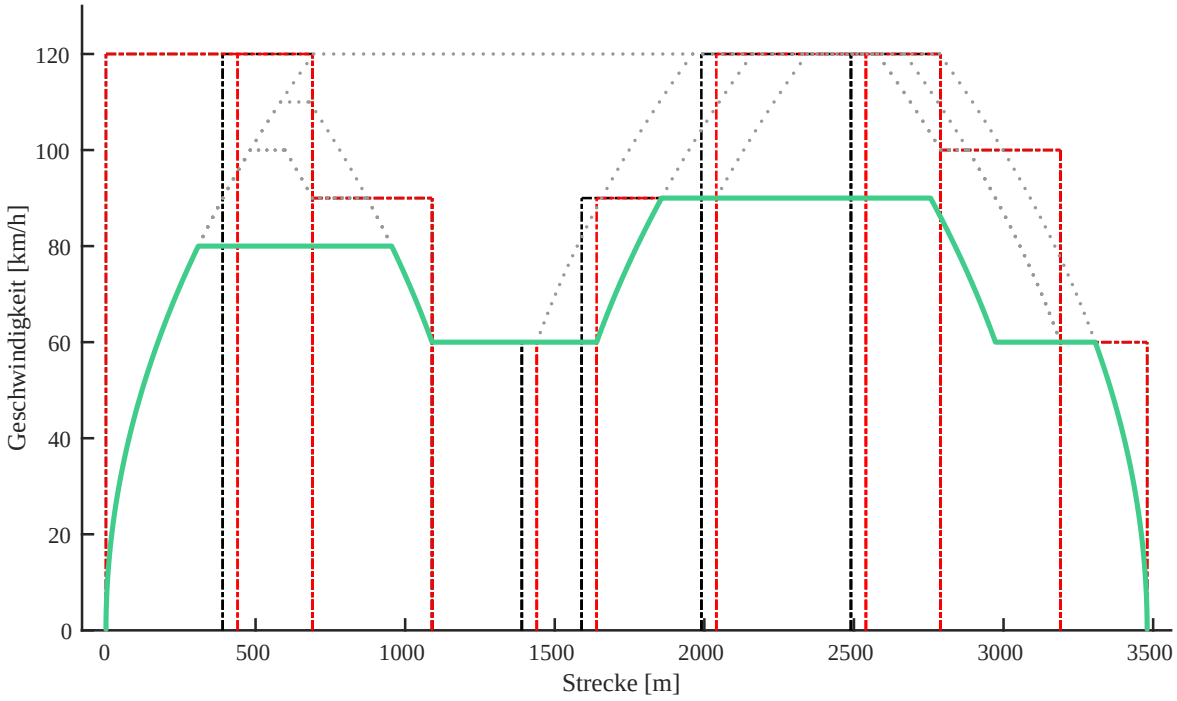

Abbildung 13: Fahrtverlauf unter Einhaltung der Mindestzeit (Quelle: Eigene Darstellung)

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*, abhängig davon, 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 *$returnSpeedDecrease* abgespeichert. Der Eintrag *possible* gibt an, ob das Fahrzeug auf dem gesamten Fahrtverlauf die Geschwindigkeit reduzieren könnte und wird als Boolescher Wert (*true/false*) abgespeichert. In dem Array *range* werden alle Indexe der möglichen *$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 die Geschwindig-

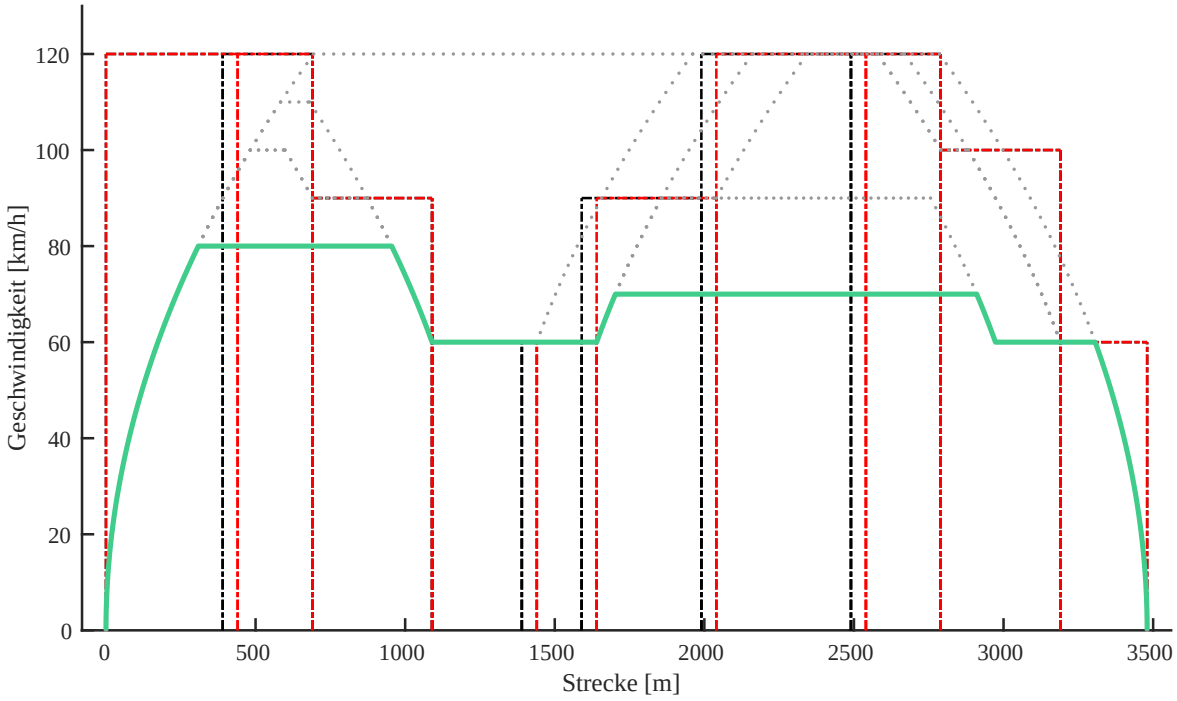

Abbildung 14: Fahrtverlauf mit reduzierter Geschwindigkeit unter Einhaltung der Ankunftszeit (Quelle: Eigene Darstellung)

keit in einer *while*-Schleife versucht 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 *$keyPoints* dieselbe Höchstgeschwindigkeit haben, wird der letzte dieser *$keyPoints* ermittelt. Im Anschluss 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.

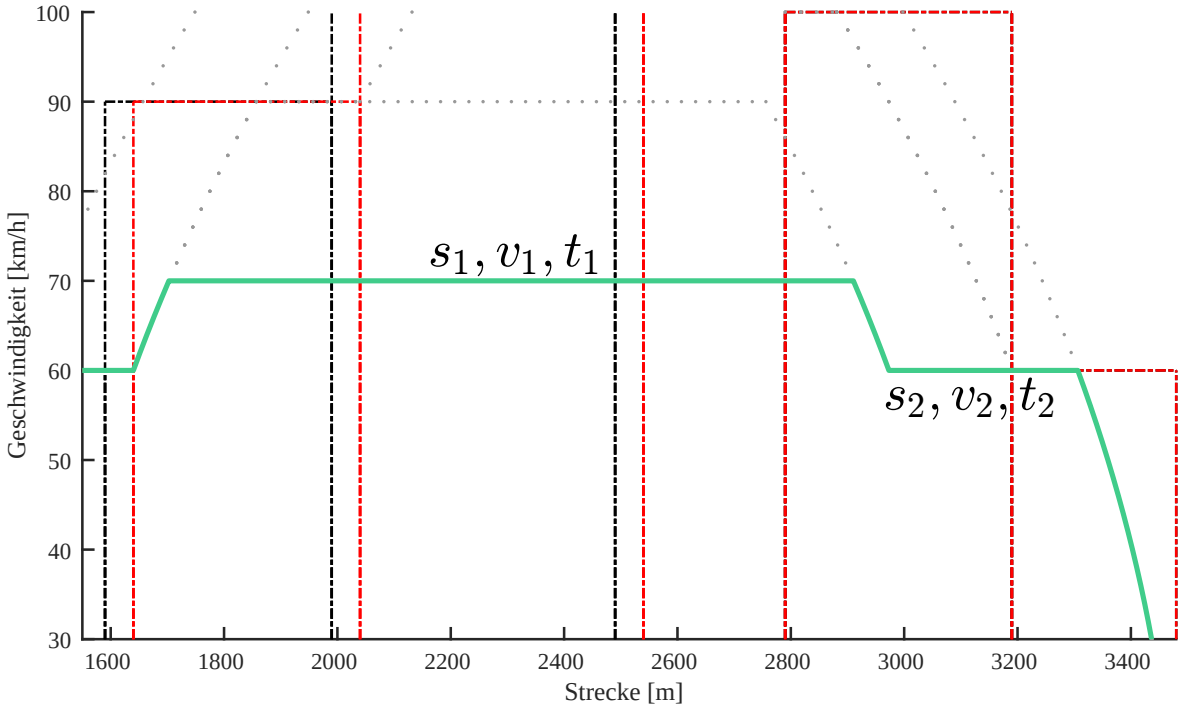

Abbildung 15: Fahrtverlauf vor der Anpassung der exakten Ankunftszeit (Quelle: Eigene Darstellung)

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 *$useSpeedFineTuning* kann festgelegt werden, ob das Fahrzeug versuchen soll, eine exakte Ankunftszeit zu erreichen. Wenn diese Funktion aktiviert ist und der Eintrag *possible* aus dem Array *$returnSpeedDecrease true* ist, wird für den letzten *$keyPoint* aus dem *$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 vor-

v_1	70 km/h
v_2	60 km/h
s_1	1207,67 m
s_2	333,33 m
s_{ges}	1541 m
t_1	62,11 s
t_2	20 s
t_Δ	3,31 s
t_{ges}	85,42 s

Tabelle 11: Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung (vor der Anpassung der exakten Ankunftszeit)

$$t_3 = \frac{1541\ m - \frac{60\ km/h}{3{,}6} \cdot 85,42\ s}{\frac{70\ km/h}{3{,}6} - \frac{60\ km/h}{3{,}6}}$$

$$t_3 = 42,26\ s$$

zeitiger eingeleitet werden soll, damit eine pünktliche Ankunft am Ziel zu ermöglicht werden kann, dargestellt und in Tabelle 11 sind die exakten Werte des exemplarischen Fahrtverlaufs aufgelistet. In diesem konkreten Beispiel würde das Fahrzeug 3,31 s (t_Δ) 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. 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.

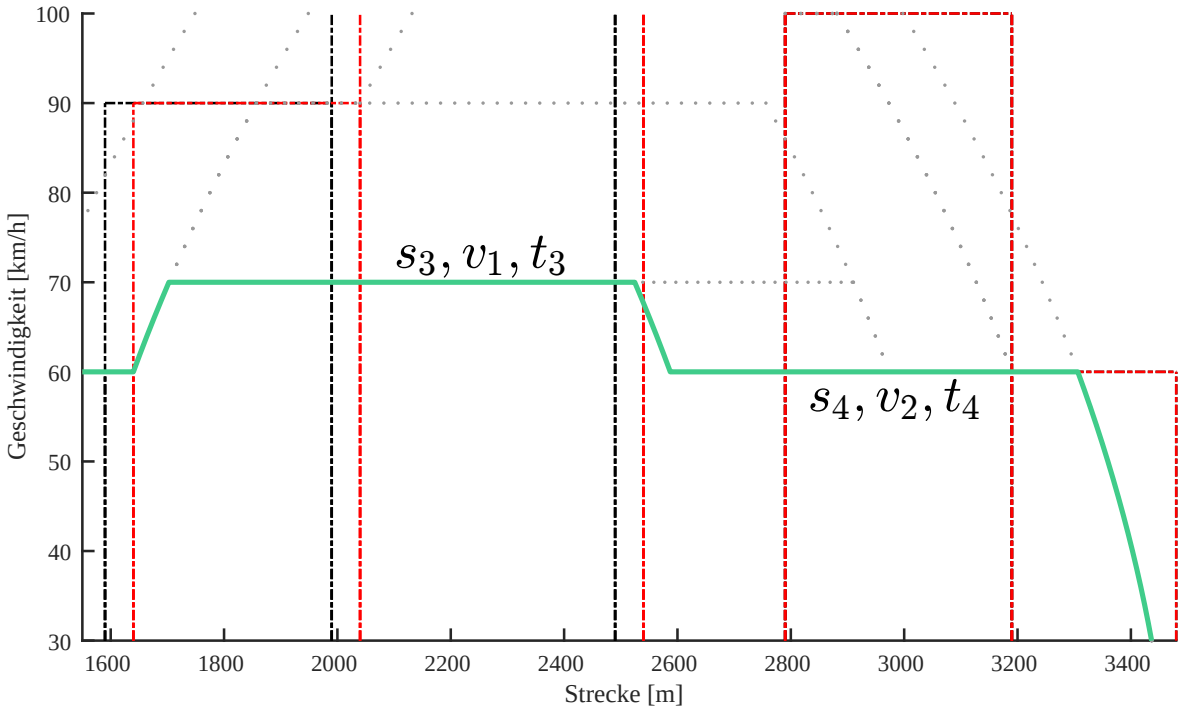

Abbildung 16: Fahrtverlauf nach der Anpassung der exakten Ankunftszeit (Quelle: Eigene Darstellung)

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 der exakten Ankunftszeit)

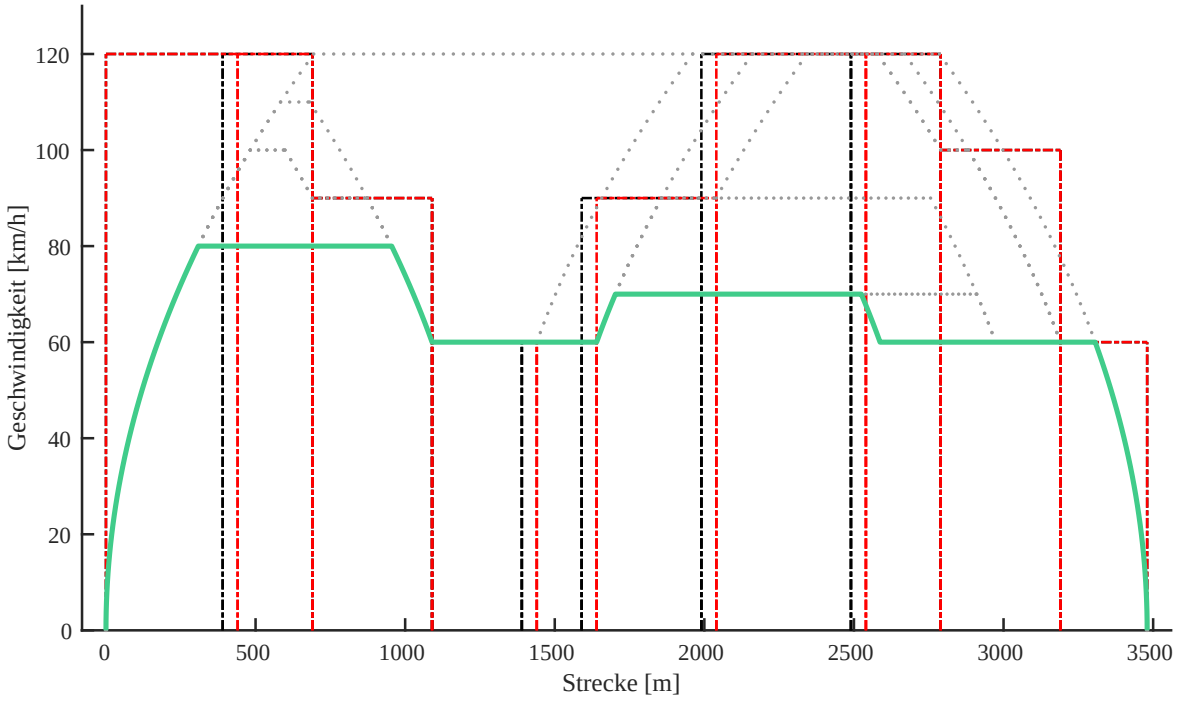

Abbildung 17: Ergebnis der Fahrtverlaufsermittlung (Quelle: Eigene Darstellung)

4.9 Einleitung einer Gefahrenbremsung

Eine Gefahrenbremsung wird eingeleitet, sobald ein Fahrzeug bei einer sofortigen Verzögerung ein auf Halt stehendes Signal überfahren, in einem Infra-Abschnitt die zulässige Höchstgeschwindigkeit überschreiten oder an dem nächsten planmäßigen Halt nicht rechtzeitig zum Stehen kommen würde. Bei einer Gefahrenbremsung wird mit einer Notbremsverzögerung von 2 m/s^2 abgebremst. Dieser Wert kann in der Datei *global_variables.php* über die Variable *$globalNotverzoegerung* 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: $v \geq 10\ km/h$ – nach der Geschwindigkeit von 10 km/h direkt die Geschwindigkeit von 0 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 $v < 10\ km/h$
3. Fahrzeug hat bei der Gefahrenstelle eine Geschwindigkeit von $v \geq 10\ km/h$

Für die Überprüfung, ob das Fahrzeug mit der Notbremsverzögerung vor der Gefah-

renstelle zum Stehen kommt, wird mittels der Funktion *getBrakeDistance*() (*functions.php*) der Bremsweg ($s_{Bremsweg}$) berechnet und mit der Distanz zur Gefahrenstelle ($s_{Gefahrenstelle}$) verglichen. Sollte für den Bremsweg gelten: $s_{Bremsweg} \leq s_{Gefahrenstelle}$, wird das Fahrzeug die Gefahrenbremsung einleiten und in 2 km/h-Schritten auf 0 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 $v \geq 10\ km/h$ haben, bremst das Fahrzeug in 2 km/h-Schritten auf 10 km/h ab und bekommt nach der Übermittlung der 10 km/h direkt 0 km/h übergeben. In dem Fall, dass das Fahrzeug an der Gefahrenstelle langsamer als 10 km/h ist, bremst das Fahrzeug wie im 1. Fall in 2 km/h-Schritten auf 0 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 und gewartet werden, bis die Fahrzeugsteuerung das Entfernen registriert. Im Anschluss kann das Fahrzeug wieder neu positioniert werden.

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

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 (XAB), hat die Fahrtrichtung 1 und die Fahrstraße ist so eingestellt, dass das Fahrzeug bis zum Ausfahrsignal im Bahnhof Zoo fahren 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^2.

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-

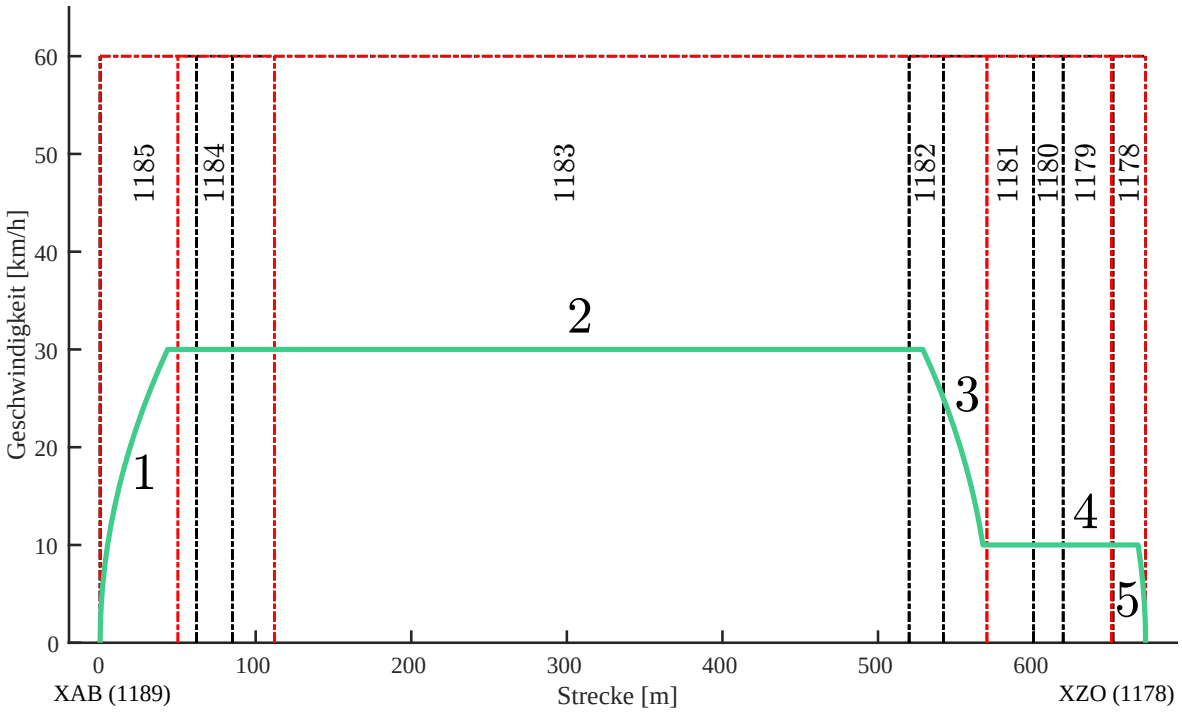

Abbildung 18: Fahrtverlauf am Beispiel von der Fahrt von XAB nach XZO (Quelle: Eigene Darstellung)

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

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

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

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

$$t_{ges} = t_{gleichförmigeBewegungen} + t_{gleichmässigBeschleunigteBewegungen}$$

$$s_{ges} = s_{gleichförmigeBewegungen} + s_{gleichmässigBeschleunigteBewegungen}$$

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

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

$$t_{gleichmässigBeschleunigteBewegungen} = t_1 + t_3 + t_5$$

$$s_{gleichmässigBeschleunigteBewegungen} = s_1 + s_3 + s_5$$

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
$\sum$	—	—	—	672 m[1]	115 s

[1] Die Werte in der Strecken-Spalte sind auf zwei Nachkommastellen gerundet und würden durch das Aufsummieren der Strecken von Abschnitt 1 – 5 eine Gesamtstrecke von 671,99 m ergeben. Die angegebenen 672 m entsprechen der Summe der Abschnitte 1 – 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:

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

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

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

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

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

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

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

$$t_{gleichmässigBeschleunigteBewegungen} = \frac{125}{6}\,s$$

$$s_{gleichmässigBeschleunigteBewegungen} = \frac{3125}{36}\, m$$

Und für die gleichförmigen Bewegungen gilt dementsprechend:

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

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

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

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

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

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

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

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

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

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

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

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

In Summe ergibt das:

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

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

Wie an den errechneten Werten zu erkennen ist, wurde die Mindestzeit von 20 s auf den Beharrungsfahrten (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 Anhang 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  // Fahrtverlaufsberechnung 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:

a = Bremsverzoegerung $[m/s^2]$
v = Geschwindigkeit $[m/s]$
s = Strecke $[m]$
t = Zeit $[s]$

7.1 Formeln für gleichmäßig beschleunigte Bewegungen

Bei einer gleichmäßig beschleunigten Bewegung gilt:[9]

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

Für die Bestimmung der Geschwindigkeit in Abhängigkeit der Zeit, muss die Beschleunigung $a(t)$ nach der Zeit t integriert werden.[10]

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

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.[11] 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 allgemeingültig sind. Für die Berechnung des Beschleuniguns- und Abbremsverhalten der Fahrzeuge ist es notwendig zu wissen, welche Strecke ein Fahrzeug zurücklegen muss, um von einer Startgeschwindigkeit v_0 auf

[9] Richard & Sander (2011, S. 22)
[10] Richard & Sander (2011, S. 20)
[11] Richard & Sander (2011, S. 20)

eine Zielgeschwindigkeit v_1 zu beschleunigen bzw. abzubremsen. Dafür wird die Gleichung für die Geschwindigkeit $v(t)$ nach $t(v)$ umgestellt und und in die Gleichung $s(t)$ eingesetzt. Daraus ergibt sich folgende Gleichung für die Strecke in Abhängigkeit von der Geschwindigkeit:

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

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

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 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 allgemeingültige 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 m/s auf v_1 und von 0 m/s auf v_0 zu beschleunigen. Für die gesuchte Strecke gilt dann:

$$s(v_0, v_1) = |s(v_1) - s(v_0)| \tag{8}$$

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

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 $verzoegerung) {
3   return abs(0.5 * ((pow($v_0/3.6,2) - pow($v_1/3.6, 2))/($verzoegerung)));
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 km/h auf v_0 bzw. v_1 zu beschleunigen und aus der Differenz wird die benötigte Zeit berechnet.

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

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

```
// Ermittelt die Distanz für Brems- und Verzögerungsvorgänge
function getBrakeTime (float $v_0, float $v_1, float $verzoegerung) {
  return abs((($v_1/3.6)/$verzoegerung) - (($v_0/3.6)/$verzoegerung));
}
```

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).

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

```
// Ermittelt die Geschwindigkeit, die ein Fahrzeug in einem Bremsvorgang
// nach einer gegebenen Distanz hat.
function getTargetBrakeSpeedWithDistanceAndStartSpeed (float $distance, float
    ↪ $verzoegerung, int $speed) {
  return sqrt((-2 * $verzoegerung * $distance) + (pow(($speed / 3.6), 2)))*3.6;
}
```

Code-Beispiel 11: *getTargetBrakeSpeedWithDistanceAndStartSpeed*() (*functions_math.php*)

7.2 Formeln für gleichförmige Bewegungen

Bei einer gleichförmigen Bewegung gilt der Grundsatz:[12]

$$v(t) = v \tag{12}$$

Für die Berechnung der Strecke gilt wie bei der gleichmäßig beschleunigten Bewegung:[13]

$$s(t) = \int v(t)\, dt \tag{13}$$

$$s(t) = v \cdot t + s_0 \tag{14}$$

[12] Richard & Sander (2011, S. 22)
[13] Richard & Sander (2011, S. 20)

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

$$s(t) = v \cdot t \tag{15}$$

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

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:

$$t_{ges} = t_1 + t_2 \tag{16}$$

$$s_{ges} = s_1 + s_2 \tag{17}$$

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

$$s_{ges} = v_1 \cdot t_1 + v_2 \cdot t_2 \tag{18}$$

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

$$t_1 = \frac{s_{ges} - v_2 \cdot t_{ges}}{v_1 - v_2} \tag{19}$$

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

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 vereinzelten 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 Ausfahrsignal 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.[14]

[14] Büker et al. (2020, S. 37)

Literatur

Büker, T., Hennig, E. & Schotten, S. (2020). Kapazitäsberechnung 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 (Letzter Zugriff am: 21. September 2021)

Ebuef: Eisenbahn-Betriebs- und Experimentierfeld Berlin. (2021). 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). (RailCommunity – 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)

A Anhang

A.1 fahrzeugsteuerung.php

```php
<?php

// Liest alle benötigten Dateien ein
require 'config/multicast.php';
require 'functions/vorbelegung.php';
require 'functions/functions.php';
require 'functions/functions_cache.php';
require 'functions/functions_db.php';
require 'functions/functions_math.php';
require 'functions/functions_ebuef.php';
require 'functions/functions_fahrtverlauf.php';
require 'global_variables.php';

// Zeitzone setzen
date_default_timezone_set('Europe/Berlin');

// PHP-Fehlermeldungen
error_reporting(1);

// Globale Variablen
global $useRecalibration;

// Fahrzeugfehlermeldungen definieren
$trainErrors = array();
$trainErrors[0] = 'Fahrtrichtung des Fahrzeugs musste geändert werden und die
    ↪ Positionsbestimmung war nicht möglich.';
$trainErrors[1] = 'In der Datenbank ist für das Fahrzeug keine Zuglänge
    ↪ angegeben.';
$trainErrors[2] = 'In der Datenbank ist für das Fahrzeug keine v_max angegeben.
    ↪ ';
$trainErrors[3] = 'Das Fahrzeug musste eine Gefahrenbremsung durchführen.';

// Statische Daten einlesen
$cacheInfranachbarn = createCacheInfranachbarn();
$cacheInfradaten = createCacheInfradaten();
```

```php
$cacheSignaldaten = createCacheSignaldaten();
$cacheInfraLaenge = createcacheInfraLaenge();
$cacheHaltepunkte = createCacheHaltepunkte();
$cacheZwischenhaltepunkte = createChacheZwischenhaltepunkte();
$cacheInfraToGbt = createCacheInfraToGbt();
$cacheGbtToInfra = createCacheGbtToInfra();
$cacheFmaToInfra = createCacheFmaToInfra();
$cacheInfraToFma = array_flip($cacheFmaToInfra);
$cacheFahrplanSession = createCacheFahrplanSession();
$cacheSignalIDToBetriebsstelle = createCacheToBetriebsstelle();
$cacheFahrzeugeAbschnitte = createCacheFahrzeugeAbschnitte();
$cacheIDTDecoder = createCacheDecoderToAdresse();
$cacheDecoderToID = array_flip($cacheIDTDecoder);
// Werden in der Funktion getAllTrains() initialisiert
$cacheAdresseToID = array();
$cacheIDToAdresse = array();

// Variablendeklaration
$allTrainsOnTheTrack = array();
$allTrains = array();
$allUsedTrains = array();
$allTimes = array();
$lastMaxSpeedForInfraAndDir = array();

// Real- und Simulationszeit ermitteln
$simulationStartTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->
    ↪ sim_startzeit, 'simulationszeit', null, array('outputtyp' => 'h:i:s')),
    ↪ 'simulationszeit', null, array('inputtyp' => 'h:i:s'));
$simulationEndTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->
    ↪ sim_endzeit, 'simulationszeit', null, array('outputtyp' => 'h:i:s')), '
    ↪ simulationszeit', null, array('inputtyp' => 'h:i:s'));
$simulationDuration = $cacheFahrplanSession->sim_endzeit -
    ↪ $cacheFahrplanSession->sim_startzeit;
$realStartTime = time();
$realEndTime = $realStartTime + $simulationDuration;
$timeDifference = $simulationStartTimeToday - $realStartTime;

// Startmeldung
startMessage();
```

```
67
68  // Ermittlung aller Fahrzeuge
69  $allTrains = getAllTrains();
70
71  // Ermittlung der Fahrzeuge im eingleisigen Netz
72  findTrainsOnTheTracks();
73
74  // Ermittlung der Fahrpläne der Fahrzeuge
75  addStopsectionsForTimetable();
76
77  // Überprüfung, ob die Fahrzeuge bereits
78  // an einer Betriebsstelle des Fahrplans stehen
79  checkIfTrainReachedHaltepunkt();
80
81  // Überprüfung, ob die Fahrtrichtung der Fahrzeuge mit dem
82  // Fahrplan übereinstimmt. Falls die Richtung nicht übereinstimmt,
83  // wird die Fahrtrichtung der Fahrzeuge geändert
84  checkIfStartDirectionIsCorrect();
85  consoleAllTrainsPositionAndFahrplan();
86  showErrors();
87
88  // Ermittlung der Fahrstraßen aller Fahrzeuge
89  calculateNextSections();
90
91  // Überprüfung, ob die Fahrstraße für die Fahrzeuge mit Fahrplan
92  // richtig eingestellt ist
93  checkIfFahrstrasseIsCorrrect();
94
95  // Ermittlung der Fahrtverläufe aller Fahrzeuge
96  calculateFahrtverlauf();
97
98  // Übermittlung der Echtzeitdaten an die Fahrzeuge
99  // $timeCheckFahrstrasseInterval => Überprüfung von Fahrstraßenänderungen
100 // $timeCheckAllTrainErrorsInterval
101 // => Ausgabe der aktuellen Positionen und Fahrplänen
102 // $timeCheckCalibrationInterval => Neukalibrierung der POsition
103 $timeCheckFahrstrasseInterval = 3;
104 $timeCheckFahrstrasse = $timeCheckFahrstrasseInterval + microtime(true);
105 $timeCheckAllTrainStatusInterval = 30;
```

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

```
            ↪ Geschwindigkeit auf ', $timeValue[0]['live_speed'], " km/h
            ↪ angepasst.\n";
        }
      } else {
        if (isset($allUsedTrains[$id]['calibrate_section_one'])) {
          if ($allUsedTrains[$id]['calibrate_section_one'] != $timeValue[0]['
              ↪ live_section']) {
            $allUsedTrains[$id]['calibrate_section_two'] = $timeValue[0]['
                ↪ live_section'];
          }
        }
        $allUsedTrains[$id]['calibrate_section_one'] = $timeValue[0]['
            ↪ live_section'];
      }

      // Aktualisierung der Position im $allUsedTrains-Array
      $allUsedTrains[$id]['current_position'] = $timeValue[0]['
          ↪ live_relative_position'];
      $allUsedTrains[$id]['current_speed'] = $timeValue[0]['live_speed'];
      $allUsedTrains[$id]['current_section'] = $timeValue[0]['live_section'];

      // Überprüfung, ob die Fahrtrichtung geändert werden muss
      if ($timeValue[0]['wendet']) {
        changeDirection($timeValue[0]['id']);
      }

      // Überprüfung, ob das Fahrzeug eine Betriebsstelle erreicht hat
      if (isset($timeValue[0]['live_all_targets_reached'])) {
        $allUsedTrains[$id]['next_betriebsstellen_data'][$timeValue[0]['
            ↪ live_all_targets_reached']]['angekommen'] = true;
        echo 'Der Zug mit der Adresse ', $timeIndex, ' hat den Halt ',
            ↪ $allUsedTrains[$id]['next_betriebsstellen_data'][$timeValue[0][
            ↪ 'live_all_targets_reached']]['betriebstelle'], " erreicht.\n";
      }

      // Überprüfung, ob ein (neuer) Fahrplan für das Fahrzeug
      // vorliegt, wenn das ermittelte Ziel erreicht wurde
      if ($timeValue[0]['live_target_reached']) {

```

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

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

```
239         }
240       }
241       $timeCheckCalibration = $timeCheckCalibration +
              ↪ $timeCheckCalibrationInterval;
242     }
243   }
244
245   // Überprüfung, ob die Fahrstraße der einzelnen Fahrzeuge sich geändert hat
246   if (microtime(true) > $timeCheckFahrstrasse) {
247     foreach ($allUsedTrains as $trainID => $trainValue) {
248       compareTwoNaechsteAbschnitte($trainID);
249     }
250
251     $returnUpdate = updateAllTrainsOnTheTrack();
252     $newTrains = $returnUpdate['new'];
253     $removeTrains = $returnUpdate['removed'];
254
255     if (sizeof($newTrains) > 0) {
256       echo "Neu hinzugefügte Züge: \n";
257       foreach ($newTrains as $newTrain) {
258         $id = $cacheDecoderToID[$newTrain];
259         echo "\tID:\t", $id, "\tAdresse:\t", $newTrain;
260       }
261       echo "\n";
262     }
263
264     foreach ($newTrains as $newTrain) {
265       $id = $cacheDecoderToID[$newTrain];
266       prepareTrainForRide($newTrain);
267       addStopsectionsForTimetable($id);
268       checkIfTrainReachedHaltepunkt($id);
269       checkIfStartDirectionIsCorrect($id);
270       consoleAllTrainsPositionAndFahrplan($id);
271       calculateNextSections($id);
272       checkIfFahrstrasseIsCorrrect($id);
273       calculateFahrtverlauf($id);
274     }
275
276     if (sizeof($removeTrains) > 0) {
```

```
277       echo "Entfernte Züge:\n";
278
279       foreach ($removeTrains as $removeTrain) {
280         $id = $cacheDecoderToID[$removeTrain];
281         unset($allUsedTrains[$id]);
282         echo "\tID:\t", $id, "\tAdresse:\t", $removeTrain;
283       }
284
285       echo "\n";
286     }
287     $timeCheckFahrstrasse = $timeCheckFahrstrasse +
          ↪ $timeCheckFahrstrasseInterval;
288   }
289
290   // Ausgabe der aktuellen Positionen, Fahrplänen
291   // und Fehlermeldungen aller Fahrzeuge
292   if (microtime(true) > $timeCheckAllTrainStatus) {
293     consoleAllTrainsPositionAndFahrplan();
294     showFahrplan();
295     showErrors();
296     $timeCheckAllTrainStatus = $timeCheckAllTrainStatus +
          ↪ $timeCheckAllTrainStatusInterval;
297   }
298
299   sleep($sleeptime);
300 }
```

A.2 functions.php

```php
<?php
// Zeigt beim Starten der Fahrzeugsteuerung eine Startmeldung im Terminal an,
// in der Informationen zur Session angezeigt werden.
function startMessage() {
  global $simulationStartTimeToday;
  global $simulationEndTimeToday;
  global $simulationDuration;
  global $realStartTime;
  global $realEndTime;
  global $cacheFahrplanSession;

  $realStartTimeAsHHMMSS = getUhrzeit($realStartTime, 'simulationszeit', null,
      ↪ array('outputtyp' => 'h:i:s'));
  $simulationEndTimeAsHHMMSS = getUhrzeit($simulationEndTimeToday, '
      ↪ simulationszeit', null, array('outputtyp' => 'h:i:s'));
  $simulationDurationAsHHMMSS = toStd($simulationDuration);
  $realEndTimeAsHHMMSS = getUhrzeit($realEndTime, 'simulationszeit', null,
      ↪ array('outputtyp' => 'h:i:s'));
  $simulationStartTimeAsHHMMSS = getUhrzeit($simulationStartTimeToday, '
      ↪ simulationszeit', null, array('outputtyp' => 'h:i:s'));
  $hashtagLine = "
      ↪ ##########################################################################\
      ↪ n";
  $emptyLine = "#\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t#\n";

  echo $hashtagLine;
  echo $emptyLine;
  echo "#\t\t\t Start der automatischen Zugbeeinflussung\t\t\t\t#\n";
  echo "#\t\tim Eisenbahnbetriebs- und Experimentierfeld (EBuEf) \t\t#\n";
  echo "#\t\t\t\t\t\t der TU Berlin\t\t\t\t\t\t\t#\n";
  echo "#\t\t\t\t\t im eingleisigen Netz \t\t\t\t\t\t#\n";
  echo $emptyLine;
  echo "#\t\t\t\t\t\t\t____\t\t\t\t\t\t\t\t\t#\n";
  echo "#\t\t\t\t\t\t\t|DD|____T_\t\t\t\t\t\t\t\t#\n";
  echo "#\t\t\t\t\t\t\t|_ |_____|<\t\t\t\t\t\t\t\t#\n";
  echo "#\t\t\t\t\t\t\t @-@-@-oo\\\t\t\t\t\t\t\t\t#\n";
  echo "#\t\t\t\t\t==========================\t\t\t\t#\n";
  echo $emptyLine;
```

```
33    echo "#\t Start der Simulation: \t\t\t\t\t\t\t\t\t\t\t#\n";
34    echo "#\t\t Simulationszeit: \t\t\t", $simulationStartTimeAsHHMMSS, "\t\t\t\t
          ↪ \t\t#\n";
35    echo "#\t\t Realzeit: \t\t\t\t\t", $realStartTimeAsHHMMSS, "\t\t\t\t\t\t#\n";
36    echo $emptyLine;
37    echo "#\t Ende der Simulation: \t\t\t\t\t\t\t\t\t\t\t#\n";
38    echo "#\t\t Simulationszeit: \t\t\t", $simulationEndTimeAsHHMMSS, "\t\t\t\t\t
          ↪ \t#\n";
39    echo "#\t\t Realzeit: \t\t\t\t\t", $realEndTimeAsHHMMSS, "\t\t\t\t\t\t#\n";
40    echo $emptyLine;
41    echo "#\t Dauer der Simulation: \t\t\t", $simulationDurationAsHHMMSS, "\t\t\t
          ↪ \t\t\t#\n";
42    echo $emptyLine;
43    echo "#\t Fahrplanname: \t\t\t\t\t", $cacheFahrplanSession->name, "\t\t#\n";
44    echo "#\t Sessionkey: \t\t\t\t\t", $cacheFahrplanSession->sessionkey, "\t\t\t
          ↪ \t\t#\n";
45    echo $emptyLine;
46    echo $hashtagLine, "\n\n";
47  }
48
49  // Konvertiert Sekunden in das Format hh:mm:ss
50  function toStd(float $sekunden) {
51
52    $stunden = floor($sekunden / 3600);
53    $minuten = floor(($sekunden - ($stunden * 3600)) / 60);
54    $sekunden = round($sekunden - ($stunden * 3600) - ($minuten * 60));
55
56    if ($stunden <= 9) {
57      $strStunden = '0'. $stunden;
58    } else {
59      $strStunden = $stunden;
60    }
61
62    if ($minuten <= 9) {
63      $strMinuten = '0'. $minuten;
64    } else {
65      $strMinuten = $minuten;
66    }
67
```

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

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

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

```
174       }
175     }
176     $allUsedTrains[$trainID]['next_betriebsstellen_data'] = array_values(
            ↪ $allUsedTrains[$trainID]['next_betriebsstellen_data']);
177   } else {
178     $allUsedTrains[$trainID]['next_betriebsstellen_data'] = array();
179   }
180
181   foreach ($allUsedTrains[$trainID]['next_betriebsstellen_data'] as
          ↪ $betriebsstelleKey => $betriebsstelleValue) {
182     if ($allUsedTrains[$trainID]['next_betriebsstellen_data'][
            ↪ $betriebsstelleKey]['zeiten']['abfahrt_soll'] != null) {
183       $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
              ↪ ]['zeiten']['abfahrt_soll_timestamp'] = getUhrzeit(
              ↪ $betriebsstelleValue['zeiten']['abfahrt_soll'], 'simulationszeit',
              ↪ null, array('inputtyp' => 'h:i:s'));
184     } else {
185       $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
              ↪ ]['zeiten']['abfahrt_soll_timestamp'] = null;
186     }
187     if ($allUsedTrains[$trainID]['next_betriebsstellen_data'][
            ↪ $betriebsstelleKey]['zeiten']['ankunft_soll'] != null) {
188       $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
              ↪ ]['zeiten']['ankunft_soll_timestamp'] = getUhrzeit(
              ↪ $betriebsstelleValue['zeiten']['ankunft_soll'], 'simulationszeit',
              ↪ null, array('inputtyp' => 'h:i:s'));
189     } else {
190       $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
              ↪ ]['zeiten']['ankunft_soll_timestamp'] = null;
191     }
192     $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey]['
            ↪ zeiten']['verspaetung'] = 0;
193   }
194 }
195
196 // Positionsermittlung einer Zuges, wenn das Fahrzeug mehrere
197 // Infrastrukturabschnitte belegt.
198 function getFrontPosition(array $infra, int $dir) {
199
```

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

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

```
258
259     $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey]['
            ↪ zeiten']['verspaetung'] = 0;
260   }
261 }
262
263 // Gibt in der Konsole für alle Züge (oder nur einen,
264 // wenn eine ID übergeben wird) die aktuellen Daten
265 // (Adresse, ID, Zug ID, Position, Fahrplan vorhanden,
266 // Fehler vorhanden und die Fahrtrichtung) aus.
267 function consoleAllTrainsPositionAndFahrplan($id = false) {
268
269   global $allUsedTrains;
270
271   $checkAllTrains = true;
272
273   if ($id != false) {
274     $checkAllTrains = false;
275   } else {
276     echo "Alle vorhandenen Züge:\n\n";
277   }
278
279   foreach ($allUsedTrains as $train) {
280     if ($checkAllTrains || $train['id'] == $id) {
281       $fahrplan = null;
282       $error = null;
283       $zugId = null;
284       if ($train['operates_on_timetable']) {
285         $fahrplan = 'ja';
286       } else {
287         $fahrplan = 'nein';
288       }
289
290       if (sizeof($train['error']) != 0) {
291         $error = 'ja';
292       } else {
293         $error = 'nein';
294       }
295
```

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

```
331
332         if (!isset($train['zug_id'])) {
333           $zugId = '-----';
334         } else {
335           $zugId = $train['zug_id'];
336         }
337
338         $nextStations = '';
339         $lastStation = '';
340
341         foreach ($train['next_betriebsstellen_data'] as $bs) {
342           if (!$bs['angekommen']) {
343             $nextStations = $nextStations . $bs['betriebstelle'] . ' ';
344
345           } else {
346             $lastStation = $bs['betriebstelle'];
347           }
348         }
349
350         if ($lastStation == '') {
351           $lastStation = '---';
352         }
353
354         echo 'Zug ID: ', $train['id'], ' (Adresse: ', $train['adresse'], ', Zug
               ↪ ID: ', $zugId, ")\t Letzte Station: ", $lastStation, " \tNächste
               ↪ Stationen: ", $nextStations, "\n";
355       }
356     }
357   }
358   echo "\n";
359 }
360
361 // Über prüft für alle Fahrzeuge die nach Fahrplan fahren (oder nur für ein
362 // Fahrzeug, wenn eine ID übergeben wird), ob die Fahrtrichtung mit dem
363 // Fahrplan übereinstimmt, und ob diese geändert werden muss. Wenn die
364 // Fahrtrichtung geändert werden muss, wird die Funktion changeDirection()
365 // aufgerufen
366 function checkIfStartDirectionIsCorrect($id = false) {
367
```

```
368   global $allUsedTrains;
369
370   $checkAllTrains = true;
371
372   if ($id != false) {
373     $checkAllTrains = false;
374     echo "Für den Fall, dass die Fahrtrichtung der Züge nicht mit dem Fahrplan
          ↪ übereinstimmt, wird die Richtung verändert:\n\n";
375   } else {
376     echo "Für den Fall, dass die Fahrtrichtung des Zuges nicht mit dem Fahrplan
          ↪  übereinstimmt, wird die Richtung verändert:\n\n";
377   }
378
379   foreach ($allUsedTrains as $train) {
380     if ($checkAllTrains || $train['id'] == $id) {
381       if ($train['operates_on_timetable']) {
382         $endLoop = 0;
383         for ($i = 0; $i < sizeof($train['next_betriebsstellen_data']); $i++) {
384           if ($train['next_betriebsstellen_data'][$i]['angekommen']) {
385             $endLoop = $i;
386           }
387         }
388
389         if ($train['dir'] != $train['next_betriebsstellen_data'][$endLoop]['
              ↪ zeiten']['fahrtrichtung'][1]) {
390           changeDirection($train['id']);
391         }
392       }
393     }
394   }
395   echo "\n";
396 }
397
398 // Ändert die Fahrtrichtung eines Zuges, wenn das möglich ist. Sollte
399 // das Fahrzeug seine Richtung ändern müssen und ist dies nicht möglich,
400 // so wird dem Fahrzeug eine Fehlermeldung (Fehlerstatus = 0) hinzugefügt.
401 function changeDirection (int $id) {
402
403   global $allUsedTrains;
```

```
404   global $cacheInfraLaenge;
405   global $timeDifference;
406   global $allTrains;
407
408   $section = $allUsedTrains[$id]['current_section'];
409   $position = $allUsedTrains[$id]['current_position'];
410   $direction = $allUsedTrains[$id]['dir'];
411   $length = $allUsedTrains[$id]['zuglaenge'];
412   $newTrainLength = $length + ($cacheInfraLaenge[$section] - $position);
413   $newDirection = null;
414   $newSection = null;
415   $cumLength = 0;
416
417   if ($direction == 0) {
418     $newDirection = 1;
419   } else {
420     $newDirection = 0;
421   }
422
423   $newPosition = null;
424   $nextSections = getNaechsteAbschnitte($section, $newDirection);
425   $currentData = array(0 => array('laenge' => $cacheInfraLaenge[$section], '
          ↪ infra_id' => $section));
426   $mergedData = array_merge($currentData, $nextSections);
427
428   foreach ($mergedData as $sectionValue) {
429     $cumLength += $sectionValue['laenge'];
430
431     if ($newTrainLength <= $cumLength) {
432       $newSection = $sectionValue['infra_id'];
433       $newPosition = $cacheInfraLaenge[$newSection] - ($cumLength -
            ↪ $newTrainLength);
434       break;
435     }
436   }
437
438   if ($newPosition == null) {
439     echo 'Die Richtung des Zugs mit der ID ', $id, " lässt sich nicht ändern,
          ↪ weil das Zugende auf einem auf Halt stehenden Signal steht.\n";
```

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

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

```
508             $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
                    ↪ $betriebsstelleKey]['haltepunkte'] = array(
                    ↪ $cacheZwischenhaltepunkte[$betriebsstelleValue['
                    ↪ betriebstelle']]);
509           } else {
510             $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
                    ↪ $betriebsstelleKey]['haltepunkte'] = array();
511           }
512         }
513       }
514     }
515   }
516 }
517
518 // Ermittelt für alle Fahrzeuge (wenn keine ID übergeben wird) oder für ein
519 // Fahrzeug (wenn eine ID übergeben wird) die Fahrstraße inkl. der Längen,
520 // der zulässigen Höchstgeschwindigkeiten und der IDs der nächsten Abschnitte.
521 //
522 // Die Ergebnisse können direkt im Array $usedTrains gespeichert werden
523 // ($writeResultToTrain = true) oder als return zurückgegeben werden
524 // ($writeResultToTrain = false), so dass sie verglichen werden können
525 // mit den vorherigen Daten verglichen werden können.
526 function calculateNextSections($id = false, $writeResultToTrain = true) {
527
528   global $allUsedTrains;
529   global $cacheInfraLaenge;
530   global $globalSpeedInCurrentSection;
531   global $lastMaxSpeedForInfraAndDir;
532
533   $checkAllTrains = true;
534
535   if ($id != false) {
536     $checkAllTrains = false;
537   }
538
539   foreach ($allUsedTrains as $trainIndex => $trainValue) {
540     if (($checkAllTrains || $trainValue['id'] == $id) && sizeof($trainValue['
            ↪ error']) == 0) {
541
```

```
542       $dir = $trainValue['dir'];
543       $currentSectionComp = $trainValue['current_section'];
544       $signal = getSignalForSectionAndDirection($currentSectionComp, $dir);
545       $nextSectionsComp = array();
546       $nextVMaxComp = array();
547       $nextLengthsComp = array();
548       $nextSignalbegriff = null;
549
550       if ($signal != null) {
551         $nextSignalbegriff = getSignalbegriff($signal);
552         $nextSignalbegriff = $nextSignalbegriff[array_key_last(
              ↪ $nextSignalbegriff)]['geschwindigkeit'];
553         if ($nextSignalbegriff == -25) {
554           $nextSignalbegriff = 25;
555         } else if ($nextSignalbegriff <= 0) {
556           $nextSignalbegriff = 0;
557         }
558       } else {
559         $nextSignalbegriff = null;
560       }
561
562       $return = getNaechsteAbschnitte($currentSectionComp, $dir);
563       $allUsedTrains[$trainIndex]['last_get_naechste_abschnitte'] = $return;
564
565       if (isset($lastMaxSpeedForInfraAndDir[$trainValue['dir']][$trainValue['
            ↪ current_section']])) {
566         $currentVMax = $lastMaxSpeedForInfraAndDir[$trainValue['dir']][
              ↪ $trainValue['current_section']];
567       } else {
568         $currentVMax = $globalSpeedInCurrentSection;
569       }
570
571       array_push($nextSectionsComp, $currentSectionComp);
572       array_push($nextVMaxComp, $currentVMax);
573       array_push($nextLengthsComp, $cacheInfraLaenge[$currentSectionComp]);
574
575       if (isset($nextSignalbegriff)) {
576         $currentVMax = $nextSignalbegriff;
577       }
```

```
578
579       if ($currentVMax == 0) {
580         if ($writeResultToTrain) {
581           $allUsedTrains[$trainIndex]['next_sections'] = $nextSectionsComp;
582           $allUsedTrains[$trainIndex]['next_lenghts'] = $nextLengthsComp;
583           $allUsedTrains[$trainIndex]['next_v_max'] = $nextVMaxComp;
584         } else {
585           return array($nextSectionsComp, $nextLengthsComp, $nextVMaxComp);
586         }
587       } else {
588         foreach ($return as $section) {
589           array_push($nextSectionsComp, $section['infra_id']);
590           array_push($nextVMaxComp, $currentVMax);
591           array_push($nextLengthsComp, $cacheInfraLaenge[$section['infra_id']]);
592           $lastMaxSpeedForInfraAndDir[intval($trainValue['dir'])][intval(
                ↪ $section['infra_id'])] = intval($currentVMax);
593           if ($section['signal_id'] != null) {
594             $signal = $section['signal_id'];
595             $nextSignalbegriff = getSignalbegriff($signal);
596             $nextSignalbegriff = $nextSignalbegriff[array_key_last(
                  ↪ $nextSignalbegriff)]['geschwindigkeit'];
597             if ($nextSignalbegriff == -25) {
598               $currentVMax = 25;
599             } else if ($nextSignalbegriff < 0) {
600               $currentVMax = 0;
601             } else {
602               $currentVMax = $nextSignalbegriff;
603             }
604           }
605         }
606         if ($writeResultToTrain) {
607           $allUsedTrains[$trainIndex]['next_sections'] = $nextSectionsComp;
608           $allUsedTrains[$trainIndex]['next_lenghts'] = $nextLengthsComp;
609           $allUsedTrains[$trainIndex]['next_v_max'] = $nextVMaxComp;
610         } else {
611           return array($nextSectionsComp, $nextLengthsComp, $nextVMaxComp);
612         }
613       }
614     }
```

```
615   }
616 }
617
618 // Prüft für alle Fahrzeuge (falls keine ID übergeben wird)
619 // oder für ein Fahrzeug (falls eine ID übergeben wird),
620 // ob das Fahrzeug bereits am ersten fahrplanmäßigen
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)
654 // oder für ein Fahrzeug (falls eine ID übergeben wird),
655 // ob die Fahrstraße aktuell richtig eingestellt ist,
656 // sodass die nächste Betriebsstelle laut Fahrplan
657 // erreicht werden kann.
658 //
659 // Für Züge ohne Fahrplan ist der Fahrweg immer korrekt.
660 function checkIfFahrstrasseIsCorrrect($id = false) {
661
662   global $allUsedTrains;
663
664   $checkAllTrains = true;
665
666   if ($id != false) {
667     $checkAllTrains = false;
668   }
669
670   foreach ($allUsedTrains as $trainIndex => $trainValue) {
671     if (($checkAllTrains || $trainValue['id'] == $id) && sizeof($trainValue['
          ↪ error']) == 0) {
672       if ($trainValue['operates_on_timetable']) {
673         $allUsedTrains[$trainIndex]['fahrstrasse_is_correct'] = false;
674         foreach ($trainValue['next_betriebsstellen_data'] as $stopIndex =>
              ↪ $stopValue) {
675           if (!$stopValue['angekommen']) {
676             $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$stopIndex
                  ↪ ]['is_on_fahrstrasse'] = false;
677             $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$stopIndex
                  ↪ ]['used_haltepunkt'] = array();
678             $indexSection = 0;
679             for ($i = 0; $i < sizeof($trainValue['next_sections']); $i++) {
680               if ($stopValue['haltepunkte'] != null) {
681                 if (in_array($trainValue['next_sections'][$i], $stopValue['
                      ↪ haltepunkte'])) {
682                   if ($i >= $indexSection) {
```

```
683                     $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
                            ↪ $stopIndex]['is_on_fahrstrasse'] = true;
684                     $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
                            ↪ $stopIndex]['used_haltepunkt'] = $trainValue['
                            ↪ next_sections'][$i];
685                     $allUsedTrains[$trainIndex]['fahrstrasse_is_correct'] = true;
686                     $i = sizeof($trainValue['next_sections']);
687                     $indexSection = $i;
688                   }
689                 }
690               }
691             }
692           } else {
693             $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$stopIndex
                    ↪ ]['is_on_fahrstrasse'] = true;
694           }
695         }
696       } else {
697         $allUsedTrains[$trainIndex]['fahrstrasse_is_correct'] = true;
698       }
699     }
700   }
701 }
702
703 // Berechnet die Beschleunigungs- und Bremskurven für alle Züge (wenn keine ID
704 // übergeben wird) oder für einen Zug (wenn eine ID übergeben wird). Für Züge -
705 // die nach Fahrplan fahren - für alle Betriebsstellen, die auf der aktuell
706 // eingestellten Strecke liegen und für Züge ohne Fahrplan bis zum nächsten
707 // roten Signal.
708 function calculateFahrtverlauf($id = false, $recalibrate = false) {
709
710   global $allUsedTrains;
711   global $cacheInfraLaenge;
712   global $timeDifference;
713   global $globalFirstHaltMinTime;
714
715   $checkAllTrains = true;
716
717   if ($id != false) {
```

```
718     $checkAllTrains = false;
719   }
720
721   foreach ($allUsedTrains as $trainIndex => $trainValue) {
722     $allPossibleStops = array();
723     for($i = 0; $i < sizeof($trainValue['next_betriebsstellen_data']); $i++) {
724       if ($trainValue['next_betriebsstellen_data'][$i]['fahrplanhalt']) {
725         array_push($allPossibleStops, $i);
726       }
727     }
728     if (sizeof($trainValue['error']) == 0 && $trainValue['
          ↪ fahrstrasse_is_correct']) {
729       if ($checkAllTrains || $trainValue['id'] == $id) {
730         if ($trainValue['operates_on_timetable']) {
731           $nextBetriebsstelleIndex = null;
732           $allreachedInfras = array();
733           $wendet = false;
734           for ($i = 0; $i < sizeof($trainValue['next_betriebsstellen_data']); $i
                ↪ ++) {
735             if (!$trainValue['next_betriebsstellen_data'][$i]['angekommen'] &&
                  ↪ $trainValue['next_betriebsstellen_data'][$i]['
                  ↪ is_on_fahrstrasse'] && $trainValue['next_betriebsstellen_data
                  ↪ '][$i]['fahrplanhalt']) {
736               $nextBetriebsstelleIndex = $i;
737               $allUsedTrains[$trainIndex]['next_bs'] = $i;
738               break;
739             }
740           }
741           if (!isset($nextBetriebsstelleIndex)) {
742             for ($i = 0; $i < sizeof($trainValue['next_betriebsstellen_data']);
                  ↪ $i++) {
743               if (!$trainValue['next_betriebsstellen_data'][$i]['angekommen'] &&
                    ↪  $trainValue['next_betriebsstellen_data'][$i]['
                    ↪ is_on_fahrstrasse']) {
744                 $nextBetriebsstelleIndex = $i;
745                 break;
746               }
747             }
748           }
```

```
749             if (isset($nextBetriebsstelleIndex)) {
750               if ($allUsedTrains[$trainIndex]['next_bs'] != $trainValue['
                      ↪ next_betriebsstellen_data'][$nextBetriebsstelleIndex]['
                      ↪ betriebstelle'] || $recalibrate) {
751                 $allUsedTrains[$trainIndex]['next_bs'] = $trainValue['
                        ↪ next_betriebsstellen_data'][$nextBetriebsstelleIndex]['
                        ↪ betriebstelle'];
752                 if (intval($trainValue['next_betriebsstellen_data'][
                        ↪ $nextBetriebsstelleIndex]['zeiten']['wendet']) == 1) {
753                   $wendet = true;
754                 }
755                 for ($i = 0; $i < sizeof($trainValue['next_betriebsstellen_data'])
                        ↪ ; $i++) {
756                   if (!$trainValue['next_betriebsstellen_data'][$i]['angekommen']
                          ↪ && $trainValue['next_betriebsstellen_data'][$i]['
                          ↪ is_on_fahrstrasse'] && $i <= $nextBetriebsstelleIndex) {
757                     array_push($allreachedInfras, array('index' => $i, 'infra' =>
                            ↪ $trainValue['next_betriebsstellen_data'][$i]['
                            ↪ used_haltepunkt']));
758                   }
759                 }
760                 $targetSection = $trainValue['next_betriebsstellen_data'][
                        ↪ $nextBetriebsstelleIndex]['used_haltepunkt'];
761                 $targetPosition = $cacheInfraLaenge[$targetSection];
762                 $startTime = null;
763                 $endTime = null;
764                 $prevBetriebsstelle = null;
765                 for ($i = 0; $i < sizeof($trainValue['next_betriebsstellen_data'])
                        ↪ ; $i++) {
766                   if ($trainValue['next_betriebsstellen_data'][$i]['angekommen'])
                          ↪ {
767                     $prevBetriebsstelle = $i;
768                     break;
769                   }
770                 }
771                 if ($nextBetriebsstelleIndex == 0) {
772                   $startTime = microtime(true) + $timeDifference;
773                   $endTime = $startTime;
774                 } else {
```

```
775                 $endTime = $trainValue['next_betriebsstellen_data'][
                        ↪ $nextBetriebsstelleIndex]['zeiten']['
                        ↪ ankunft_soll_timestamp'];
776                 if (isset($prevBetriebsstelle)) {
777                   if ($trainValue['next_betriebsstellen_data'][
                          ↪ $prevBetriebsstelle]['zeiten']['verspaetung'] > 0) {
778                     $startTime = $trainValue['next_betriebsstellen_data'][
                            ↪ $prevBetriebsstelle]['zeiten']['abfahrt_soll_timestamp
                            ↪ '] + $trainValue['next_betriebsstellen_data'][
                            ↪ $nextBetriebsstelleIndex - 1]['zeiten']['verspaetung'
                            ↪ ];
779                   } else {
780                     $startTime = $trainValue['next_betriebsstellen_data'][
                            ↪ $prevBetriebsstelle]['zeiten']['abfahrt_soll_timestamp
                            ↪ '];
781                   }
782                 } else {
783                   $startTime = microtime(true) + $timeDifference;
784                 }
785               }
786               $reachedBetriebsstele = true;
787
788               if ($startTime < microtime(true) + $timeDifference) {
789                 $startTime = microtime(true) + $timeDifference;
790               }
791
792               if (isset($trainValue['earliest_possible_start_time'])) {
793                 if ($startTime < $trainValue['earliest_possible_start_time']) {
794                   $startTime = $trainValue['earliest_possible_start_time'];
795                 }
796               }
797
798               $verapetung = updateNextSpeed($trainValue, $startTime, $endTime,
                      ↪ $targetSection, $targetPosition, $reachedBetriebsstele,
                      ↪ $nextBetriebsstelleIndex, $wendet, false, $allreachedInfras
                      ↪ );
799
800               if ($nextBetriebsstelleIndex != 0) {
```

```
801                 $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
                        ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] =
                        ↪ $verapetung;
802                 $trainValue['next_betriebsstellen_data'][
                        ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] =
                        ↪ $verapetung;
803               } else {
804                 $end = $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
                        ↪ $nextBetriebsstelleIndex]['zeiten']['
                        ↪ abfahrt_soll_timestamp'];
805                 $start = $startTime;
806                 if ($start + $verapetung + $globalFirstHaltMinTime < $end) {
807                   $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
                          ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] = 0;
808                   $trainValue['next_betriebsstellen_data'][
                          ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] = 0;
809                 } else {
810                   $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
                          ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] =
                          ↪ $start + $verapetung + $globalFirstHaltMinTime - $end;
811                   $trainValue['next_betriebsstellen_data'][
                          ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] =
                          ↪ $start + $verapetung + $globalFirstHaltMinTime - $end;
812                 }
813               }
814             }
815           } else {
816             if ($trainValue['current_speed'] > 0) {
817               emergencyBreak($trainValue['id']);
818             }
819           }
820         } else {
821           $startTime = microtime(true) + $timeDifference;
822           if (isset($trainValue['earliest_possible_start_time'])) {
823             if ($startTime < $trainValue['earliest_possible_start_time']) {
824               $startTime = $trainValue['earliest_possible_start_time'];
825             }
826           }
827           $endTime = $startTime;
```

```
828           $targetSection = null;
829           $targetPosition = null;
830           $reachedBetriebsstele = true;
831           $wendet = false;
832           $signalId = null;
833           for ($i = 0; $i < sizeof($trainValue['last_get_naechste_abschnitte']);
                  ↪  $i++) {
834             if (isset($trainValue['last_get_naechste_abschnitte'][$i]['signal_id
                    ↪ '])) {
835               $signalId = $trainValue['last_get_naechste_abschnitte'][$i]['
                      ↪ signal_id'];
836               $targetSection = $trainValue['last_get_naechste_abschnitte'][$i]['
                      ↪ infra_id'];
837               $targetPosition = $cacheInfraLaenge[$targetSection];
838             }
839           }
840           if (!isset($signalId)) {
841             if ($trainValue['current_speed'] != 0) {
842               emergencyBreak($trainValue['id']);
843             }
844           } else {
845             $signal = getSignalbegriff($signalId)[0]['geschwindigkeit'];
846
847             if ($signal > -25 && $signal < 0) {
848               $wendet = true;
849             }
850
851             updateNextSpeed($trainValue, $startTime, $endTime, $targetSection,
                    ↪ $targetPosition, $reachedBetriebsstele, $signalId, $wendet,
                    ↪ true, array());
852           }
853         }
854       }
855     } else {
856       if ($trainValue['current_speed'] != 0) {
857         emergencyBreak($trainValue['id']);
858       }
859     }
860   }
```

```
861 }
862
863 // Vergleicht für ein Fahrzeug die zuletzt ermittelte
864 // Fahrstraße mit der aktuellen Fahrstraße und berechnet
865 // den Fahrtverlauf neu, wenn das nötig ist.
866 function compareTwoNaechsteAbschnitte(int $id) {
867
868   global $allUsedTrains;
869   global $allTimes;
870
871   if (sizeof($allUsedTrains[$id]['error']) == 0) {
872     $newSections = calculateNextSections($id, false);
873     $newNextSection = $newSections[0];
874     $newNextLenghts = $newSections[1];
875     $newNextVMax = $newSections[2];
876     $oldNextSections = $allUsedTrains[$id]['next_sections'];
877     $oldLenghts = $allUsedTrains[$id]['next_lenghts'];
878     $oldNextVMax = $allUsedTrains[$id]['next_v_max'];
879     $currentSectionOld = $allUsedTrains[$id]['current_section'];
880     $keyCurrentSection = array_search($currentSectionOld, $oldNextSections);
881     $keyLatestSection = array_key_last($oldNextSections);
882     $dataIsIdentical = true;
883     $numberOfSection = $keyLatestSection - $keyCurrentSection + 1;
884     $compareNextSections = array();
885     $compareNextLenghts = array();
886     $compareNextVMax = array();
887
888     for($i = $keyCurrentSection; $i <= $keyLatestSection; $i++) {
889       array_push($compareNextSections, $oldNextSections[$i]);
890       array_push($compareNextLenghts, $oldLenghts[$i]);
891       array_push($compareNextVMax, $oldNextVMax[$i]);
892     }
893
894     if (sizeof($newNextSection) != ($numberOfSection)) {
895       $dataIsIdentical = false;
896     } else {
897       for ($i = 0; $i < $keyLatestSection - $keyCurrentSection; $i++) {
898         if ($newNextSection[$i] != $compareNextSections[$i] || $newNextLenghts[
              ↪ $i] != $compareNextLenghts[$i] || $newNextVMax[$i] !=
```

```
            ↪ $compareNextVMax[$i]) {
899           $dataIsIdentical = false;
900           break;
901         }
902       }
903     }
904
905     if (!$dataIsIdentical) {
906       echo 'Die Fahrstraße des Zuges mit der ID: ', $id, " hat sich geändert.\n
              ↪ ";
907       calculateNextSections($id);
908       $adresse = $allUsedTrains[$id]['adresse'];
909       $allTimes[$adresse] = array();
910       checkIfFahrstrasseIsCorrrect($id);
911       calculateFahrtverlauf($id);
912     }
913   }
914 }
```

A.3 functions_fahrtverlauf.php

```php
<?php

// Berechnet den Fahrtverlauf eines Fahrzeugs
function updateNextSpeed (array $train, float $startTime, float $endTime, int
    ↪ $targetSectionPara, int $targetPositionPara, bool $reachedBetriebsstelle
    ↪ , string $indexReachedBetriebsstelle, bool $wendet, bool $freieFahrt,
    ↪ array $allreachedInfras) {

  global $useSpeedFineTuning;
  global $next_sections;
  global $next_lengths;
  global $next_v_max;
  global $allTimes;
  global $verzoegerung;
  global $notverzoegerung;
  global $currentSection;
  global $currentPosition;
  global $currentSpeed;
  global $targetSpeed;
  global $targetSection;
  global $targetPosition;
  global $targetTime;
  global $indexCurrentSection;
  global $indexTargetSection;
  global $distanceToNextStop;
  global $trainSpeedChange;
  global $trainPositionChange;
  global $trainTimeChange;
  global $cumulativeSectionLengthEnd;
  global $cumulativeSectionLengthStart;
  global $keyPoints;
  global $allUsedTrains;
  global $globalIndexBetriebsstelleFreieFahrt;
  global $cacheSignalIDToBetriebsstelle;
  global $useMinTimeOnSpeed;
  global $slowDownIfTooEarly;
  global $globalFloatingPointNumbersRoundingError;

```

```
36  $emptyArray = array();
37  $keyPoints = $emptyArray;
38  $cumulativeSectionLengthStart = $emptyArray;
39  $cumulativeSectionLengthEnd = $emptyArray;
40  $next_sections = $train['next_sections'];
41  $next_lengths = $train['next_lenghts'];
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]['betriebstelle'];
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,
77  // die größer als die zulässige Höchstgeschwindigkeit des
78  // Fahrzeugs ist, wird die Geschwindigkeit des
79  // 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,
      ↪ $indexTargetSection, $currentPosition, $targetPosition, $next_lengths)
      ↪ ;
102 $cumulativeSectionLengthStart = $returnCumulativeSections[0];
103 $cumulativeSectionLengthEnd = $returnCumulativeSections[1];
104 $cumLengthEnd = array();
105 $cumLengthStart = array();
106 $sum = 0;
107
108 foreach ($next_lengths as $index => $value) {
```

```
109     if ($index >= $indexCurrentSection) {
110       $cumLengthStart[$index] = $sum;
111       $sum += $value;
112       $cumLengthEnd[$index] = $sum;
113     }
114   }
115
116   // Ermittlung der Distanz bis zum Ziel
117   $distanceToNextStop = $cumulativeSectionLengthEnd[$indexTargetSection];
118   if (getBrakeDistance($currentSpeed, $targetSpeed, $verzoegerung)>
        ↪ $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;
```

```
147     $next_lengths_mod[$indexTargetSectionMod] = $targetPosition;
148   } else {
149     $startPosition = 0;
150     $indexStartPosition = null;
151     $indexEndPosition = null;
152 
153     do {
154       $reachedTargetSection = false;
155 
156       for ($j = $indexCurrentSection; $j <= $indexTargetSection; $j++) {
157         if ($startPosition >= $cumLengthStart[$j] && $startPosition <
              ↪ $cumLengthEnd[$j]) {
158           $indexStartPosition = $j;
159         }
160       }
161 
162       $endPosition = $cumLengthEnd[$indexStartPosition] + $train_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 <
                ↪ $cumLengthEnd[$j]) {
172             $indexEndPosition = $j;
173           }
174         }
175       }
176 
177       for ($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 
```

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

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

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

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

```
316             $localKeyPoints[$possibleSpeedRange['first_key_point_index'] + 1]['
                    ↪ speed_0'] = $i;
317             $localKeyPoints[$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']);
318             $localKeyPoints[$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));
319             $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
320             $newCalculatedTime = $localKeyPoints[array_key_last($localKeyPoints)
                    ↪ ]['time_1'];
321
322             if ($i == 10) {
323               if ($newCalculatedTime > $maxTimeToNextStop) {
324                 $localKeyPoints[$possibleSpeedRange['first_key_point_index']]['
                        ↪ speed_1'] = $i + 10;
325                 $localKeyPoints[$possibleSpeedRange['first_key_point_index'] +
                        ↪ 1]['speed_0'] = $i + 10;
326                 $localKeyPoints[$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'
                        ↪ ]);
327                 $localKeyPoints[$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));
328               }
329
330               $finish = true;
331               $newKeyPoints = $localKeyPoints;
332               break;
```

```
333             }
334             if (($newCalculatedTime - $startTime) > $maxTimeToNextStop) {
335               if ($i == $possibleSpeedRange['max_speed']) {
336                 $localKeyPoints = $keyPointsPreviousStep;
337                 $localKeyPoints = deleteDoubledKeyPoints($localKeyPoints);
338                 $keyPoints = $localKeyPoints;
339                 $finish = true;
340                 break;
341               }
342               $localKeyPoints[$possibleSpeedRange['first_key_point_index']]['
                      ↪ speed_1'] = $i + 10;
343               $localKeyPoints[$possibleSpeedRange['first_key_point_index'] + 1][
                      ↪ 'speed_0'] = $i + 10;
344               $localKeyPoints[$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'
                      ↪ ]);
345               $localKeyPoints[$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));
346               $newKeyPoints = $localKeyPoints;
347               $finish = true;
348               $keyPoints = $localKeyPoints;
349
350               break;
351             }
352             if ($i == $possibleSpeedRange['min_speed']) {
353               $newKeyPoints = $localKeyPoints;
354               $newKeyPoints = deleteDoubledKeyPoints($newKeyPoints);
355               $keyPoints = $newKeyPoints;
356               break;
357             }
358             $newKeyPoints = $localKeyPoints;
359           }
```

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

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

```
422     $trainBetriebsstelleName[$key] = $targetBetriebsstelle;
423     if (array_key_last($trainPositionChange) != $key) {
424       $trainTargetReached[$key] = false;
425       $trainWendet[$key] = false;
426     } else {
427       if ($wendet) {
428         $trainWendet[$key] = true;
429       } else {
430         $trainWendet[$key] = false;
431       }
432
433       if ($reachedBetriebsstelle) {
434         $trainTargetReached[$key] = true;
435       } else {
436         $trainTargetReached[$key] = false;
437       }
438     }
439   }
440
441   for($i = sizeof($trainSection) - 1; $i >= 0; $i--) {
442     if (in_array($trainSection[$i], $allreachedInfrasID) && !in_array(
          ↪ $trainSection[$i], $allreachedInfrasUsed)) {
443       array_push($allreachedInfrasUsed, $trainSection[$i]);
444       $Infraindex = array_search($trainSection[$i], $allreachedInfrasID);
445       $allReachedTargets[$i] = $allreachedInfrasIndex[$Infraindex];
446     } else {
447       $allReachedTargets[$i] = null;
448     }
449   }
450   ksort($allReachedTargets);
451   $returnArray = array();
452   $adress = $train['adresse'];
453   $trainID = array();
454   $id = $train['id'];
455
456   foreach ($trainPositionChange as $key => $value) {
457     $trainID[$key] = $id;
458   }
459
```

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

```
      ↪ $globalFloatingPointNumbersRoundingError)) {
    array_push($v_max, $i);
  }
}

if (sizeof($v_max) == 0) {
  if ($v_0 == 0 && $v_1 == 0 && $distance > 0) {
    echo 'Der zug müsste langsamer als 10 km/h fahren, um das Ziel zu
      ↪ erreichen.';
  } else {
    emergencyBreak($id);
  }
} else {
  if ($v_0 == $v_1 && max($v_max) < $v_0) {
    $v_max = array($v_0);
  }
}

return max($v_max);
}

// Erstellt einen $keyPoint
function createKeyPoint (float $position_0, float $position_1, int $speed_0,
  ↪ int $speed_1) {
  return array('position_0' => $position_0, 'position_1' => $position_1, '
    ↪ speed_0' => $speed_0, 'speed_1' => $speed_1);
}

// Ermittelt aus den $keyPoint die Echtzeitdaten
// (nur Geschwindigkeit und Position)
function convertKeyPointsToTrainChangeArray (array $keyPoints) {

  global $verzoegerung;

  $trainSpeedChangeReturn = array();
  $trainPositionChnageReturn = array();
  array_push($trainPositionChnageReturn, $keyPoints[0]['position_0']);
  array_push($trainSpeedChangeReturn, $keyPoints[0]['speed_0']);

```

```
525   for ($i = 0; $i <= (sizeof($keyPoints) - 2); $i++) {
526     if ($keyPoints[$i]['speed_0'] < $keyPoints[$i]['speed_1']) {
527       for ($j = $keyPoints[$i]['speed_0']; $j < $keyPoints[$i]['speed_1']; $j =
          ↪  $j + 2) {
528         array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn)
            ↪ + getBrakeDistance($j, ($j + 2), $verzoegerung)));
529         array_push($trainSpeedChangeReturn, ($j + 2));
530       }
531     } elseif ($keyPoints[$i]['speed_0'] > $keyPoints[$i]['speed_1']) {
532       for ($j = $keyPoints[$i]['speed_0']; $j > $keyPoints[$i]['speed_1']; $j =
          ↪  $j - 2) {
533         array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn)
            ↪ + getBrakeDistance($j, ($j - 2), $verzoegerung)));
534         array_push($trainSpeedChangeReturn, ($j - 2));
535       }
536     }
537     array_push($trainPositionChnageReturn, $keyPoints[$i + 1]['position_0']);
538     array_push($trainSpeedChangeReturn, $keyPoints[$i + 1]['speed_0']);
539   }
540
541   if (end($keyPoints)['speed_0'] < end($keyPoints)['speed_1']) {
542     for ($j = end($keyPoints)['speed_0']; $j < end($keyPoints)['speed_1']; $j =
        ↪  $j + 2) {
543       array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
          ↪  getBrakeDistance($j, ($j + 2), $verzoegerung)));
544       array_push($trainSpeedChangeReturn, ($j + 2));
545     }
546   } else if (end($keyPoints)['speed_0'] > end($keyPoints)['speed_1']) {
547     for ($j = end($keyPoints)['speed_0']; $j > end($keyPoints)['speed_1']; $j =
        ↪  $j - 2) {
548       array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
          ↪  getBrakeDistance($j, ($j - 2), $verzoegerung)));
549       array_push($trainSpeedChangeReturn, ($j - 2));
550     }
551   }
552
553   return array($trainPositionChnageReturn, $trainSpeedChangeReturn);
554 }
555
```

```
556 // Wandelt die Daten der Infra-Abschnitte und der Iterationsschritte der
557 // Fahrtverlaufsberechnung in JSON-Dateien um, damit die Fahrtverläufe
558 // visuell dargestellt werden können.
559 function safeTrainChangeToJSONFile(int $indexCurrentSection, int
        ↪ $indexTargetSection, int $indexCurrentSectionMod, int
        ↪ $indexTargetSectionMod, array $speedOverPositionAllIterations) {
560
561   global $trainPositionChange;
562   global $trainSpeedChange;
563   global $next_v_max;
564   global $cumulativeSectionLengthEnd;
565   global $next_v_max_mod;
566   global $cumulativeSectionLengthEndMod;
567
568   $speedOverPosition = array_map('toArr', $trainPositionChange,
        ↪ $trainSpeedChange);
569   $speedOverPosition = json_encode($speedOverPosition);
570   $fp = fopen('../json/speedOverPosition.json', 'w');
571   fwrite($fp, $speedOverPosition);
572   fclose($fp);
573
574   $v_maxFromUsedSections = array();
575
576   for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
577     array_push($v_maxFromUsedSections, $next_v_max[$i]);
578   }
579
580   $VMaxOverCumulativeSections = array_map('toArr', $cumulativeSectionLengthEnd,
        ↪  $v_maxFromUsedSections);
581   $VMaxOverPositionsJSon = json_encode($VMaxOverCumulativeSections);
582   $fp = fopen('../json/VMaxOverCumulativeSections.json', 'w');
583   fwrite($fp, $VMaxOverPositionsJSon);
584   fclose($fp);
585
586   $v_maxFromUsedSections = array();
587
588   for ($i = $indexCurrentSectionMod; $i <= $indexTargetSectionMod; $i++) {
589     array_push($v_maxFromUsedSections, $next_v_max_mod[$i]);
590   }
```

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

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

```
660         array_push($doubledIndex, $i);
661       }
662     }
663
664     foreach ($doubledIndex as $index) {
665       unset($temporaryKeyPoints[$index]);
666     }
667
668     $temporaryKeyPoints = array_values($temporaryKeyPoints);
669   } while ($foundDoubledKeyPoints);
670
671   return $temporaryKeyPoints;
672 }
673
674 // Ermittelt die Zeiten der $keyPoint ausgehend vom ersten $keyPoint. Mit dem
675 // Parameter $inputKeyPoints können $keyPoints übergeben werden, bei den die
676 // Zeit ermittelt wird. Wenn keine $keyPoints übergeben werden, werden die
677 // globalen $keyPoints verwendet. Mit dem Parameter $skippingKeys können
678 // $keyPoints übersprungen werden. Das auslassen von $keyPoints ist für die
679 // Funktion postponeSubsection() relevant.
680 function calculateTimeFromKeyPoints($inputKeyPoints = null, $skippingKeys =
      ↪ null) {
681
682   global $keyPoints;
683   global $verzoegerung;
684
685   if ($inputKeyPoints == null) {
686     $localKeyPoints = $keyPoints;
687   } else {
688     $localKeyPoints = $inputKeyPoints;
689   }
690
691   $keys = array_keys($localKeyPoints);
692
693   if ($skippingKeys != null) {
694     foreach ($skippingKeys as $skip) {
695       unset($keys[array_search($skip, $keys)]);
696     }
697   }
```

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

```
730   $returnTrainRelativePosition = array();
731   $returnTrainSection = array();
732   $returnIsSpeedChange = array();
733
734   // Ermittelt für alle bis auf den letzten $keyPoint die Echtzeitdaten der
735   // Zeit, Geschwindigkeit und Position
736   for ($i = 0; $i < sizeof($keyPoints); $i++) {
737     array_push($returnTrainTimeChange, $keyPoints[$i]['time_0']);
738     array_push($returnTrainSpeedChange, $keyPoints[$i]['speed_0']);
739     array_push($returnTrainPositionChange, $keyPoints[$i]['position_0']);
740     array_push($returnIsSpeedChange, true);
741
742     $itDir = ($keyPoints[$i]['speed_0'] < $keyPoints[$i]['speed_1']) ? 2 : -2;
743
744     for ($j = ($keyPoints[$i]['speed_0'] + $itDir); $j <= $keyPoints[$i]['
          ↪ speed_1']; $j = $j + $itDir) {
745       array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
            ↪  getBrakeDistance(($j - $itDir), $j, $verzoegerung)));
746       array_push($returnTrainSpeedChange, $j);
747       array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
            ↪ getBrakeTime(($j - $itDir), $j, $verzoegerung))));
748       array_push($returnIsSpeedChange, true);
749     }
750
751     // Überprüft, ob nach dem $keyPoint eine Beharrungsfahrt stattfindet
752     if ($i != array_key_last($keyPoints)) {
753       // Ermittelt für die Strecke zwischen zwei $keyPoints die Echtzeitdaten
754       // der Zeit, Geschwindigkeit und Position
755       $startPosition = $keyPoints[$i]['position_1'];
756       $endPosition = $keyPoints[$i + 1]['position_0'];
757       $speedToNextKeyPoint = $keyPoints[$i]['speed_1'];
758       $timeForOneTimeInterval = distanceWithSpeedToTime($speedToNextKeyPoint,
            ↪ $globalDistanceUpdateInterval);
759
760       for ($position = $startPosition + $globalDistanceUpdateInterval;
            ↪ $position < $endPosition; $position = $position +
            ↪ $globalDistanceUpdateInterval) {
761         array_push($returnTrainPositionChange, $position);
762         array_push($returnTrainSpeedChange, $speedToNextKeyPoint);
```

```
763         array_push($returnTrainTimeChange, end($returnTrainTimeChange) +
                ↪ $timeForOneTimeInterval);
764         array_push($returnIsSpeedChange, false);
765       }
766     }
767   }
768   array_push($returnTrainPositionChange, $keyPoints[array_key_last($keyPoints)
        ↪ ]['position_1'] - getBrakeDistance($keyPoints[array_key_last(
        ↪ $keyPoints)]['speed_0'],$keyPoints[array_key_last($keyPoints)]['
        ↪ speed_1'],$verzoegerung));
769   array_push($returnTrainSpeedChange, $keyPoints[array_key_last($keyPoints)]['
        ↪ speed_0']);
770   array_push($returnTrainTimeChange, $keyPoints[array_key_last($keyPoints)]['
        ↪ time_0']);
771   array_push($returnIsSpeedChange, true);
772
773   if ($onlyPositionAndSpeed) {
774     return array($returnTrainPositionChange, $returnTrainSpeedChange);
775   } else {
776     // Ermittelt die relativen Positionen innerhalb der Infra-Abschnitte
777     // zu den absoluten Positionen
778     foreach ($returnTrainPositionChange as $absolutPositionKey =>
            ↪ $absolutPositionValue) {
779       foreach ($cumulativeSectionLengthStart as $sectionStartKey =>
              ↪ $sectionStartValue) {
780         if ($absolutPositionValue >= $sectionStartValue && $absolutPositionValue
                ↪  < $cumulativeSectionLengthEnd[$sectionStartKey]) {
781           if ($sectionStartKey == $indexCurrentSection && $sectionStartKey ==
                  ↪ $indexTargetSection) {
782             $returnTrainRelativePosition[$absolutPositionKey] =
                    ↪ $absolutPositionValue + $currentPosition;
783             $returnTrainSection[$absolutPositionKey] = $next_sections[
                    ↪ $sectionStartKey];
784           } else if ($sectionStartKey == $indexCurrentSection) {
785             $returnTrainRelativePosition[$absolutPositionKey] =
                    ↪ $absolutPositionValue + $currentPosition;
786             $returnTrainSection[$absolutPositionKey] = $next_sections[
                    ↪ $sectionStartKey];
787           } else if ($sectionStartKey == $indexTargetSection) {
```

```
788             $returnTrainRelativePosition[$absolutPositionKey] =
                    ↪ $absolutPositionValue - $sectionStartValue;
789             $returnTrainSection[$absolutPositionKey] = $next_sections[
                    ↪ $sectionStartKey];
790           } else {
791             $returnTrainRelativePosition[$absolutPositionKey] =
                    ↪ $absolutPositionValue - $sectionStartValue;
792             $returnTrainSection[$absolutPositionKey] = $next_sections[
                    ↪ $sectionStartKey];
793           }
794           break;
795         } else if ($absolutPositionKey == array_key_last(
                ↪ $returnTrainPositionChange) && abs($absolutPositionValue -
                ↪ floatval($cumulativeSectionLengthEnd[$sectionStartKey])) <
                ↪ $globalFloatingPointNumbersRoundingError) {
796           $returnTrainRelativePosition[$absolutPositionKey] =
                  ↪ $cumulativeSectionLengthEnd[$sectionStartKey] -
                  ↪ $sectionStartValue;
797           $returnTrainSection[$absolutPositionKey] = $next_sections[
                  ↪ $sectionStartKey];
798           break;
799         } else {
800           debugMessage('Einer absoluten Position konnte kein Infra-Abschnitt und
                  ↪  keine relative Position in einem Infra-Abschnitt zugeordnet
                  ↪ werden.');
801         }
802       }
803     }
804
805     return array($returnTrainPositionChange, $returnTrainSpeedChange,
          ↪ $returnTrainTimeChange, $returnTrainRelativePosition,
          ↪ $returnTrainSection, $returnIsSpeedChange);
806   }
807 }
808
809 // Überprüft, ob es zwischen zwei benachbarten $keyPoints
810 // zu einer Geschwindigkeitsüberschreitung kommt.
811 function recalculateKeyPoints(array $tempKeyPoints, int $id) {
812
```

```
813   $returnKeyPoints = array();
814   $numberOfPairs = sizeof($tempKeyPoints) / 2;
815
816   for($j = 0; $j < $numberOfPairs; $j++) {
817     $i = $j * 2;
818     $return = checkBetweenTwoKeyPoints($tempKeyPoints, $i, $id);
819
820     foreach ($return as $keyPoint) {
821       array_push($returnKeyPoints, $keyPoint);
822     }
823   }
824
825   return $returnKeyPoints;
826 }
827
828 // Überprüft, ob zwischen zwei $keyPoints die zulässige Höchstgeschwindigkeit
829 // überschritten wird
830 function checkBetweenTwoKeyPoints(array $temKeyPoints, int $keyPointIndex, int
        ↪ $id) {
831
832   global $trainPositionChange;
833   global $trainSpeedChange;
834   global $cumulativeSectionLengthStartMod;
835   global $cumulativeSectionLengthEndMod;
836   global $next_v_max_mod;
837   global $verzoegerung;
838   global $indexTargetSectionMod;
839
840   $failedSections = array();
841   $groupedFailedSections = array();
842   $returnKeyPoints = array();
843   $failedPositions = array();
844   $failedSpeeds = array();
845
846   foreach ($trainPositionChange as $trainPositionChangeKey =>
        ↪ $trainPositionChangeValue) {
847     if ($trainPositionChangeValue >= $temKeyPoints[$keyPointIndex]['position_0'
          ↪ ] && $trainPositionChangeValue <= $temKeyPoints[$keyPointIndex + 1][
          ↪ 'position_1']) {
```

```
848         foreach ($cumulativeSectionLengthStartMod as
                ↪ $cumulativeSectionLengthStartKey =>
                ↪ $cumulativeSectionLengthStartValue) {
849           if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
850             if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
                    ↪ $cumulativeSectionLengthStartKey - 1]) {
851               array_push($failedSections, ($cumulativeSectionLengthStartKey - 1));
852               array_push($failedSpeeds, $trainSpeedChange[$trainPositionChangeKey
                      ↪ ]);
853               $failedPositions[$trainPositionChangeKey] = $trainPositionChange[
                      ↪ $trainPositionChangeKey];
854             }
855             break;
856           } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
857             if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
858               if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
                      ↪ $cumulativeSectionLengthStartKey]) {
859                 array_push($failedSections, $cumulativeSectionLengthStartKey);
860                 array_push($failedSpeeds, $trainSpeedChange[
                        ↪ $trainPositionChangeKey]);
861                 $failedPositions[$trainPositionChangeKey] = $trainPositionChange[
                        ↪ $trainPositionChangeKey];
862               }
863               break;
864             }
865           }
866         }
867       }
868     }
869
870     // Alle Infra_abschnitte zwischen denn beiden KeyPoints, bei denen die
871     // zulässige Höchstgeschwindigkeit überschritten wird
872     $failedSections = array_unique($failedSections);
873
874     // Wenn es kein Fehler gibt, werden die beiden KeyPoints zurückgegeben und
875     // wen es einen Fehler gibt, wird der erste der beiden KeyPoints im
876     // $returnKeyPoints gespeichert
877     if (sizeof($failedSections) == 0) {
```

```
878     return array($temKeyPoints[$keyPointIndex], $temKeyPoints[$keyPointIndex +
            ↪ 1]);
879   } else {
880     $returnKeyPoints[0]['speed_0'] = $temKeyPoints[$keyPointIndex]['speed_0'];
881     $returnKeyPoints[0]['position_0'] = $temKeyPoints[$keyPointIndex]['
            ↪ position_0'];
882   }
883
884   // Einteilung der benachbarten failedSections in zusammenhängende Gruppen
885   $previous = NULL;
886   $index = 0;
887   foreach($failedSections as $key => $value) {
888     if($value > $previous + 1) {
889       $index++;
890     }
891     $groupedFailedSections[$index][] = $value;
892     $previous = $value;
893   }
894
895   // Iteration über die zusammenhängenden $failedSections
896   foreach ($groupedFailedSections as $groupSectionsIndex => $groupSectionsValue
            ↪ ) {
897     $firstFailedPositionIndex = null;
898     $lastFailedPositionIndex = null;
899     $firstFailedPosition = null;
900     $lastFailedPosition = null;
901     $lastElement = array_key_last($returnKeyPoints);
902     $failedSection = null;
903
904     // Ermittlung der Section mit der kleinsten v_max von allen $failedSections
905     // in der Gruppe
906     if (sizeof($groupSectionsValue) == 1) {
907       $failedSection = $groupSectionsValue[0];
908     } else {
909       $slowestSpeed = 200;
910       for ($i = 0; $i <= (sizeof($groupSectionsValue) - 1); $i++) {
911         if ($next_v_max_mod[$groupSectionsValue[$i]] < $slowestSpeed) {
912           $slowestSpeed = $next_v_max_mod[$groupSectionsValue[$i]];
913           $failedSection = $groupSectionsValue[$i];
```

```
914         }
915       }
916     }
917
918     // Start- und Endposition der $failedSection
919     $failedSectionStart = $cumulativeSectionLengthStartMod[$failedSection];
920     $failedSectionEnd = $cumulativeSectionLengthEndMod[$failedSection];
921
922     // Bestimmung der ersten und letzten Position, in der es in der
        ↪ $failedSection
923     // zu einer Geschwindigkeitsüberschreitung kommt
924     foreach ($failedPositions as $failPositionIndex => $failPositionValue) {
925       if ($failPositionValue > $failedSectionStart && $failPositionValue <
          ↪ $failedSectionEnd) {
926         if ($firstFailedPositionIndex == null) {
927           $firstFailedPositionIndex = $failPositionIndex;
928         }
929         $lastFailedPositionIndex = $failPositionIndex;
930       }
931     }
932
933     // Bestimmung des letzten Punktes, bei dem die Geschwindigkeit noch
934     // nicht zu schnell war
935     //
936     // Wenn der Punkt davor außerhalb der failedSection liegt
937     // => Startpunkt = Anfang der Section
938     // Wenn der Punkt davor innerhalb der failed Section liegt
939     // => Startpunkt = der Punkt davor
940     if ($firstFailedPositionIndex != 0) {
941       if ($trainPositionChange[$firstFailedPositionIndex - 1] <
          ↪ $failedSectionStart) {
942         $firstFailedPosition = $failedSectionStart;
943       } else {
944         $firstFailedPosition = $trainPositionChange[$firstFailedPositionIndex -
            ↪ 1];
945       }
946     } else {
947       $firstFailedPosition = $failedSectionStart;
948     }
```

```
949
950     // Bestimmung der ersten Position, bei dem die Geschwindigkeit nicht
951     // mehr zu hoch war.
952     if ($lastFailedPositionIndex != array_key_last($trainPositionChange)) {
953       if ($trainPositionChange[$lastFailedPositionIndex + 1] >
            ↪ $failedSectionEnd) {
954         $lastFailedPosition = $failedSectionEnd;
955       } else {
956         $lastFailedPosition = $trainPositionChange[$lastFailedPositionIndex +
              ↪ 1];
957       }
958     } else {
959       $lastFailedPosition = $failedSectionEnd;
960     }
961
962     $returnKeyPoints[$lastElement + 1]['position_1'] = $firstFailedPosition;
963     $returnKeyPoints[$lastElement + 1]['speed_1'] = $next_v_max_mod[
          ↪ $failedSection];
964     $returnKeyPoints[$lastElement + 2]['position_0'] = $lastFailedPosition;
965     $returnKeyPoints[$lastElement + 2]['speed_0'] = $next_v_max_mod[
          ↪ $failedSection];
966   }
967
968   // Zielwerte des letzten $keyPoint vom zweiten $keyPoint übernehmen
969   $returnKeyPoints[array_key_last($returnKeyPoints) + 1]['position_1'] =
        ↪ $temKeyPoints[$keyPointIndex + 1]['position_1'];
970   $returnKeyPoints[array_key_last($returnKeyPoints)]['speed_1'] = $temKeyPoints
        ↪ [$keyPointIndex + 1]['speed_1'];
971   $numberOfPairs = sizeof($returnKeyPoints) / 2;
972
973   for($j = 0; $j < $numberOfPairs; $j++) {
974     $i = $j * 2;
975     $distance = $returnKeyPoints[$i + 1]['position_1'] - $returnKeyPoints[$i]['
          ↪ position_0'];
976     $vMax = getVMaxBetweenTwoPoints($distance, $returnKeyPoints[$i]['speed_0'],
          ↪  $returnKeyPoints[$i + 1]['speed_1'], $id);
977     $returnKeyPoints[$i]['speed_1'] = $vMax;
978     $returnKeyPoints[$i]['position_1'] = $returnKeyPoints[$i]['position_0'] +
          ↪ getBrakeDistance($returnKeyPoints[$i]['speed_0'], $vMax,
```

```
        ↪ $verzoegerung);
979     $returnKeyPoints[$i + 1]['speed_0'] = $vMax;
980     $returnKeyPoints[$i + 1]['position_0'] = $returnKeyPoints[$i + 1]['
        ↪ position_1'] - getBrakeDistance($vMax, $returnKeyPoints[$i + 1]['
        ↪ speed_1'], $verzoegerung);
981   }
982
983   return $returnKeyPoints;
984 }
985
986 // Wenn ein Key Point beschleunigt und der nächste Key Point abbremst, wird
987 // die Geschwindigkeit zwischen den beiden KeyPoints als $v_maxBetweenKeyPoints
988 // gespeichert und als $v_minBetweenKeyPoints der größere Wert von
989 // $keyPoints[$i]["speed_0"] und $keyPoints[$i + 1]["speed_1"]
990 function checkIfTheSpeedCanBeDecreased() {
991
992   global $keyPoints;
993   global $returnPossibleSpeed;
994
995   $returnPossibleSpeed = array();
996
997   for ($i = 0; $i < (sizeof($keyPoints) - 1); $i++) {
998     $v_maxBetweenKeyPoints = $keyPoints[$i]['speed_1'];
999     $v_minBetweenKeyPoints = null;
1000
1001    if ($keyPoints[$i]['speed_0'] < $v_maxBetweenKeyPoints && $keyPoints[$i +
        ↪ 1]['speed_1'] < $v_maxBetweenKeyPoints) {
1002      $v_minBetweenKeyPoints = $keyPoints[$i]['speed_0'];
1003      if ($keyPoints[$i + 1]['speed_1'] > $v_minBetweenKeyPoints) {
1004        $v_minBetweenKeyPoints = $keyPoints[$i + 1]['speed_1'];
1005      }
1006    }
1007
1008    if (isset($v_minBetweenKeyPoints)) {
1009      if ($v_minBetweenKeyPoints == 0 && $v_maxBetweenKeyPoints >= 10) {
1010        $v_minBetweenKeyPoints = 10;
1011      } else if ($v_minBetweenKeyPoints == 0 && $v_maxBetweenKeyPoints == 10) {
1012        $v_minBetweenKeyPoints = null;
1013      }
```

```
1014      }
1015
1016      if ($v_minBetweenKeyPoints != null) {
1017        if ($v_minBetweenKeyPoints % 10 != 0) {
1018          $rest = $v_minBetweenKeyPoints % 10;
1019          $v_minBetweenKeyPoints = $v_minBetweenKeyPoints - $rest + 10;
1020        }
1021
1022        array_push($returnPossibleSpeed, array('KeyPoint_index' => $i, 'values'
              => range($v_minBetweenKeyPoints, $v_maxBetweenKeyPoints, 10)));
1023      }
1024    }
1025    if (sizeof($returnPossibleSpeed) > 0) {
1026      return array('possible' => true, 'range' => $returnPossibleSpeed);
1027    } else {
1028      return array('possible' => false, 'range' => array());
1029    }
1030  }
1031
1032  // Wenn in 'global_variables.php' der Variablen $useSpeedFineTuning
1033  // der Wert 'true' zugewiesen ist und das Fahrzeug zu früh an der
1034  // nächsten Betriebsstelle ankommt, wird überprüft, ob durch eine
1035  // vorzeitige Einleitung einer Verzögerung die exakte Ankunftszeit
1036  // eingehalten werden kann.
1037  function speedFineTuning(float $timeDiff, int $index) {
1038
1039    global $keyPoints;
1040    global $verzoegerung;
1041    global $globalTimeOnOneSpeed;
1042    global $useMinTimeOnSpeed;
1043
1044    $speed_0 = $keyPoints[$index]['speed_1'];
1045    $speed_1 = null;
1046    $availableDistance = $keyPoints[$index + 1]['position_0'] - $keyPoints[$index
          ]['position_1'];
1047    $timeBetweenKeyPoints = $keyPoints[$index + 1]['time_0'] - $keyPoints[$index
          ]['time_1'];
1048    $availableTime = $timeBetweenKeyPoints + $timeDiff;
1049
```

```
1050  if ($keyPoints[$index + 1]['speed_1'] != 0) {
1051    $speed_1 = $keyPoints[$index + 1]['speed_1'];
1052    $lengthDifference = calculateDistanceforSpeedFineTuning($keyPoints[$index +
          ↪  1]['speed_0'], $keyPoints[$index + 1]['speed_1'],
          ↪ $availableDistance, $availableTime);
1053
1054    if ($useMinTimeOnSpeed) {
1055      if (distanceWithSpeedToTime($speed_0, $availableDistance -
            ↪ $lengthDifference) > $globalTimeOnOneSpeed &&
            ↪ distanceWithSpeedToTime($speed_1, $lengthDifference) >
            ↪ $globalTimeOnOneSpeed) {
1056        $keyPoints[$index + 1]['position_0'] = $keyPoints[$index + 1]['
              ↪ position_0'] - $lengthDifference;
1057        $keyPoints[$index + 1]['position_1'] = $keyPoints[$index + 1]['
              ↪ position_1'] - $lengthDifference;
1058      }
1059    } else {
1060      $keyPoints[$index + 1]['position_0'] = $keyPoints[$index + 1]['position_0
            ↪ '] - $lengthDifference;
1061      $keyPoints[$index + 1]['position_1'] = $keyPoints[$index + 1]['position_1
            ↪ '] - $lengthDifference;
1062    }
1063  } else if ($keyPoints[$index + 1]['speed_0'] > 10) {
1064    $speed_1 = 10;
1065    $lengthDifference = calculateDistanceforSpeedFineTuning($keyPoints[$index +
          ↪  1]['speed_0'],10, $availableDistance, $availableTime);
1066
1067    if ($useMinTimeOnSpeed) {
1068      if (distanceWithSpeedToTime($speed_0, $availableDistance -
            ↪ $lengthDifference) > $globalTimeOnOneSpeed &&
            ↪ distanceWithSpeedToTime($speed_1, $lengthDifference) >
            ↪ $globalTimeOnOneSpeed) {
1069        $firstKeyPoint = createKeyPoint(($keyPoints[$index + 1]['position_0'] -
              ↪ $lengthDifference),($keyPoints[$index + 1]['position_0'] -
              ↪ $lengthDifference + getBrakeDistance($keyPoints[$index + 1]['
              ↪ speed_0'],10, $verzoegerung)),$keyPoints[$index + 1]['speed_0'
              ↪ ],10);
1070        $secondKeyPoint = createKeyPoint(($keyPoints[$index + 1]['position_1'] -
              ↪  getBrakeDistance(10, 0, $verzoegerung)),$keyPoints[$index + 1]['
```

```
            ↪ position_1'],10,$keyPoints[$index + 1]['speed_1']);
1071        $keyPoints[$index + 1] = $secondKeyPoint;
1072        array_splice( $keyPoints, ($index + 1), 0, array($firstKeyPoint));
1073      }
1074    } else {
1075      $firstKeyPoint = createKeyPoint(($keyPoints[$index + 1]['position_0'] -
            ↪ $lengthDifference),($keyPoints[$index + 1]['position_0'] -
            ↪ $lengthDifference + getBrakeDistance($keyPoints[$index + 1]['
            ↪ speed_0'],10, $verzoegerung)),$keyPoints[$index + 1]['speed_0'],10)
            ↪ ;
1076      $secondKeyPoint = createKeyPoint(($keyPoints[$index + 1]['position_1'] -
            ↪ getBrakeDistance(10, 0, $verzoegerung)),$keyPoints[$index + 1]['
            ↪ position_1'],10,$keyPoints[$index + 1]['speed_1']);
1077      $keyPoints[$index + 1] = $secondKeyPoint;
1078      array_splice( $keyPoints, ($index + 1), 0, array($firstKeyPoint));
1079    }
1080  }
1081 }
1082
1083 // Sucht den KeyPoint der zu maximalen Geschwindigkeit beschleunigt
1084 // Wenn die maximale Geschwindigkeit mehrfach erreciht wird, wird
1085 // der letzte dieser KeyPoints genommen
1086 //
1087 // Zu dem Index wird auch die Speed Range abgespeichert wie bei
1088 // checkIfTheSpeedCanBeDecreased()
1089 function findMaxSpeed(array $speedDecrease) {
1090   $maxSpeed = 0;
1091   $minSpeed = 0;
1092   $keyPointIndex = null;
1093
1094   for ($i = 0; $i < sizeof($speedDecrease['range']); $i++) {
1095     if (max($speedDecrease['range'][$i]['values']) >= $maxSpeed) {
1096       $maxSpeed = max($speedDecrease['range'][$i]['values']);
1097       $minSpeed = min($speedDecrease['range'][$i]['values']);
1098       $keyPointIndex = $speedDecrease['range'][$i]['KeyPoint_index'];
1099     }
1100   }
1101   return array('min_speed' => $minSpeed, 'max_speed' => $maxSpeed, '
          ↪ first_key_point_index' => $keyPointIndex);
```

```
1102 }
1103
1104 // Überprüft beim Start der Fahrtverlaufsberechnung,
1105 // ob es möglich ist einen Fahrtverlauf zu ermitteln
1106 function checkIfItsPossible(int $id) {
1107
1108   global $currentSpeed;
1109   global $distanceToNextStop;
1110   global $verzoegerung;
1111   global $globalTimeOnOneSpeed;
1112   global $errorMinTimeOnSpeed;
1113
1114   $minTimeIsPossible = true;
1115
1116   if ($currentSpeed == 0) {
1117     $distance_0 = getBrakeDistance(0, 10, $verzoegerung);
1118     $distance_1 = getBrakeDistance(10, 0, $verzoegerung);
1119     $time = distanceWithSpeedToTime(10, $distanceToNextStop - $distance_0 -
          ↪ $distance_1);
1120
1121     if ($time < $globalTimeOnOneSpeed) {
1122       $minTimeIsPossible = false;
1123
1124       if ($errorMinTimeOnSpeed) {
1125         emergencyBreak($id);
1126       }
1127
1128       echo "Der Zug schafft es ohne eine Gefahrenbremsung am Ziel anzukommen,
            ↪ kann aber nicht die mind. Zeit einhalten.\n";
1129     }
1130   } else {
1131     if (getBrakeDistance($currentSpeed, 0, $verzoegerung) !=
          ↪ $distanceToNextStop) {
1132       $distance_0 = getBrakeDistance($currentSpeed, 10, $verzoegerung);
1133       $distance_1 = getBrakeDistance(10, 0, $verzoegerung);
1134       $time = distanceWithSpeedToTime(10, $distanceToNextStop - $distance_0 -
            ↪ $distance_1);
1135
1136       if ($time < $globalTimeOnOneSpeed) {
```

```
1137        $minTimeIsPossible = false;
1138
1139        if ($errorMinTimeOnSpeed) {
1140          emergencyBreak($id);
1141        }
1142
1143        echo "Der Zug schafft es, ohne eine Gefahrenbremsung am Ziel anzukommen
                ↪ .\n";
1144      }
1145    }
1146  }
1147  return $minTimeIsPossible;
1148 }
1149
1150 // Überprüft, ob der vorgeschriebene Wert aus der Variablen
1151 // $globalTimeOnOneSpeed eingehalten wird, falls die
1152 // Variable $useMinTimeOnSpeed den Wert 'true' hat
1153 function toShortOnOneSpeed () {
1154
1155   global $keyPoints;
1156   global $verzoegerung;
1157
1158   $localKeyPoints = $keyPoints;
1159   $subsections = createSubsections($localKeyPoints);
1160
1161   while (toShortInSubsection($subsections)) {
1162     $breakesOnly = true;
1163     foreach ($subsections as $sectionKey => $sectionValue) {
1164       if ($sectionValue['failed']) {
1165         if (!$sectionValue['brakes_only']) {
1166           $breakesOnly = false;
1167         }
1168
1169         $return = postponeSubsection($localKeyPoints, $sectionValue);
1170
1171         if (!$return['fail']) {
1172           $localKeyPoints = $return['keyPoints'];
1173         } else {
1174           if (!$sectionValue['brakes_only']) {
```

```
1175            $localKeyPoints[$sectionValue['max_index']]['speed_1'] -= 10;
1176            $localKeyPoints[$sectionValue['max_index'] + 1]['speed_0'] -= 10;
1177            $localKeyPoints[$sectionValue['max_index']]['position_1'] =
                    ↪ $localKeyPoints[$sectionValue['max_index']]['position_0'] +
                    ↪ getBrakeDistance($localKeyPoints[$sectionValue['max_index']][
                    ↪ 'speed_0'], $localKeyPoints[$sectionValue['max_index']]['
                    ↪ speed_1'], $verzoegerung);
1178            $localKeyPoints[$sectionValue['max_index'] + 1]['position_0'] =
                    ↪ $localKeyPoints[$sectionValue['max_index'] + 1]['position_1']
                    ↪  - getBrakeDistance($localKeyPoints[$sectionValue['max_index'
                    ↪ ] + 1]['speed_0'], $localKeyPoints[$sectionValue['max_index']
                    ↪  + 1]['speed_1'], $verzoegerung);
1179            $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1180            $localKeyPoints = deleteDoubledKeyPoints($localKeyPoints);
1181            break;
1182          }
1183        }
1184      }
1185    }
1186    $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1187    $localKeyPoints = array_values($localKeyPoints);
1188    $subsections = createSubsections($localKeyPoints);
1189
1190    if ($breakesOnly) {
1191      break;
1192    }
1193  }
1194  $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1195  $keyPoints = $localKeyPoints;
1196 }
1197
1198 // Überprüft, ob innerhalb einer $subsection Beschleuniguns- und
1199 // Bremsvorgänge später bzw. früher eingeleitet werden können.
1200 function postponeSubsection (array $localKeyPoints, array $subsection) {
1201
1202   global $globalTimeOnOneSpeed;
1203   global $verzoegerung;
1204
1205   $deletedKeyPoints = array();
```

```
$numberOfKeyPoints = sizeof($subsection['indexes']);
$indexMaxSection = array_search($subsection['max_index'], $subsection['
    ↪ indexes']);
$indexLastKeyPoint = array_key_last($subsection['indexes']);

if ($subsection['is_prev_section']) {
  $timeDiff = $localKeyPoints[$subsection['indexes'][0]]['time_0'] -
      ↪ $localKeyPoints[$subsection['indexes'][0] - 1]['time_1'] -
      ↪ $globalTimeOnOneSpeed;
  if ($timeDiff < 0) {
    $positionDiff = abs($timeDiff) * $localKeyPoints[$subsection['indexes'
        ↪ ][0]]['speed_0'] / 3.6;
    if (!($localKeyPoints[$subsection['indexes'][0]]['position_1'] +
        ↪ $positionDiff > $localKeyPoints[$subsection['indexes'][
        ↪ $indexMaxSection + 1]]['position_0'])) {
      $localKeyPoints[$subsection['indexes'][0]]['position_0'] +=
          ↪ $positionDiff;
      $localKeyPoints[$subsection['indexes'][0]]['position_1'] +=
          ↪ $positionDiff;
      if ($localKeyPoints[$subsection['indexes'][0]]['position_1'] >
          ↪ $localKeyPoints[$subsection['indexes'][0] + 1]['position_0']) {
        array_push($deletedKeyPoints, $subsection['indexes'][0] + 1);
        $numberOfKeyPoints -= 1;
        $v_0 = $localKeyPoints[$subsection['indexes'][0]]['speed_0'];
        $v_1 = $localKeyPoints[$subsection['indexes'][0] + 1]['speed_1'];
        $localKeyPoints[$subsection['indexes'][0]]['position_1'] =
            ↪ $localKeyPoints[$subsection['indexes'][0]]['position_0'] +
            ↪ getBrakeDistance($v_0, $v_1, $verzoegerung);
        $localKeyPoints[$subsection['indexes'][0]]['speed_1'] = $v_1;
      }
      $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
          ↪ $deletedKeyPoints);
    }
  }
}

for ($i = 1; $i <= $indexMaxSection; $i++) {
  if (!in_array($subsection['indexes'][$i], $deletedKeyPoints)) {
```

```
1232      $timeDiff = $localKeyPoints[$subsection['indexes'][$i]]['time_0'] -
              ↪ $localKeyPoints[$subsection['indexes'][$i] - 1]['time_1'] -
              ↪ $globalTimeOnOneSpeed;
1233      if ($timeDiff < 0) {
1234        $positionDiff = abs($timeDiff) * $localKeyPoints[$subsection['indexes'][
              ↪ $i]]['speed_0'] / 3.6;
1235        if (!($localKeyPoints[$subsection['indexes'][$i]]['position_1'] +
              ↪ $positionDiff > $localKeyPoints[$subsection['indexes'][
              ↪ $indexMaxSection + 1]]['position_0'])) {
1236          $localKeyPoints[$subsection['indexes'][$i]]['position_0'] +=
                ↪ $positionDiff;
1237          $localKeyPoints[$subsection['indexes'][$i]]['position_1'] +=
                ↪ $positionDiff;
1238          if ($i < $indexMaxSection && $localKeyPoints[$subsection['indexes'][$i
                ↪ ]]['position_1'] > $localKeyPoints[$subsection['indexes'][$i] +
                ↪  1]['position_0']) {
1239            array_push($deletedKeyPoints, ($subsection['indexes'][$i] + 1));
1240            $numberOfKeyPoints -= 1;
1241            $v_0 = $localKeyPoints[$subsection['indexes'][$i]]['speed_0'];
1242            $v_1 = $localKeyPoints[$subsection['indexes'][$i] + 1]['speed_1'];
1243            $localKeyPoints[$subsection['indexes'][$i]]['position_1'] =
                  ↪ $localKeyPoints[$subsection['indexes'][$i]]['position_0'] +
                  ↪ getBrakeDistance($v_0, $v_1, $verzoegerung);
1244            $localKeyPoints[$subsection['indexes'][$i]]['speed_1'] = $v_1;
1245          }
1246          $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
                ↪ $deletedKeyPoints);
1247        }
1248      }
1249    }
1250  }
1251
1252  if ($subsection['is_next_section']) {
1253    $timeDiff = $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint] +
          ↪ 1]['time_0'] - $localKeyPoints[$subsection['indexes'][
          ↪ $indexLastKeyPoint]]['time_1'] - $globalTimeOnOneSpeed;
1254    if ($timeDiff < 0) {
1255      $positionDiff = abs($timeDiff) * $localKeyPoints[$indexLastKeyPoint]['
            ↪ speed_1'] / 3.6;
```

```
1256      if (!($localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]]['
              ↪ position_0'] - $positionDiff < $localKeyPoints[$subsection['indexes
              ↪ '][$indexMaxSection]]['position_0'])) {
1257        $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]]['position_0'
                ↪ ] -= $positionDiff;
1258        $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]]['position_1'
                ↪ ] -= $positionDiff;
1259        if ($localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]]['
                ↪ position_0'] < $localKeyPoints[$subsection['indexes'][
                ↪ $indexLastKeyPoint] - 1]['position_1']) {
1260          array_push($deletedKeyPoints, ($subsection['indexes'][
                  ↪ $indexLastKeyPoint] - 1));
1261          $numberOfKeyPoints -= 1;
1262          $v_0 = $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint] -
                  ↪ 1]['speed_0'];
1263          $v_1 = $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]]['
                  ↪ speed_1'];
1264          $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]]['
                  ↪ position_0'] = $localKeyPoints[$subsection['indexes'][
                  ↪ $indexLastKeyPoint]]['position_1'] - getBrakeDistance($v_0,
                  ↪ $v_1, $verzoegerung);
1265          $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]]['speed_0']
                  ↪  = $v_0;
1266        }
1267        $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
                ↪ $deletedKeyPoints);
1268      }
1269    }
1270  }
1271
1272  for ($i = $indexLastKeyPoint - 1; $i > $indexMaxSection; $i--) {
1273    if (!in_array($i, $deletedKeyPoints)) {
1274      $timeDiff = $localKeyPoints[$subsection['indexes'][$i + 1]]['time_0'] -
              ↪ $localKeyPoints[$subsection['indexes'][$i]]['time_1'] -
              ↪ $globalTimeOnOneSpeed;
1275      if ($timeDiff < 0) {
1276        $positionDiff = abs($timeDiff) * $localKeyPoints[$indexLastKeyPoint]['
                ↪ speed_1'] / 3.6;
```

```
1277        if (!($localKeyPoints[$subsection['indexes'][$i]]['position_0'] -
                ↪ $positionDiff < $localKeyPoints[$subsection['indexes'][
                ↪ $indexMaxSection]]['position_0'])) {
1278          $localKeyPoints[$subsection['indexes'][$i]]['position_0'] -=
                  ↪ $positionDiff;
1279          $localKeyPoints[$subsection['indexes'][$i]]['position_1'] -=
                  ↪ $positionDiff;
1280          if ($i > ($indexMaxSection + 1) && $localKeyPoints[$subsection['
                  ↪ indexes'][$i]]['position_0'] < $localKeyPoints[$subsection['
                  ↪ indexes'][$i] - 1]['position_1']) {
1281            array_push($deletedKeyPoints, ($subsection['indexes'][$i] - 1));
1282            $numberOfKeyPoints -= 1;
1283            $v_0 = $localKeyPoints[$subsection['indexes'][$i] - 1]['speed_0'];
1284            $v_1 = $localKeyPoints[$subsection['indexes'][$i]]['speed_1'];
1285            $localKeyPoints[$subsection['indexes'][$i]]['position_0'] =
                    ↪ $localKeyPoints[$subsection['indexes'][$i]]['position_1'] -
                    ↪ getBrakeDistance($v_0, $v_1, $verzoegerung);
1286            $localKeyPoints[$subsection['indexes'][$i]]['speed_0'] = $v_0;
1287          }
1288          $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
                  ↪ $deletedKeyPoints);
1289        }
1290      }
1291    }
1292  }
1293
1294  $keys = $subsection['indexes'];
1295
1296  foreach ($deletedKeyPoints as $index) {
1297    unset($keys[array_search($index, $keys)]);
1298  }
1299
1300  // Ordnet die Abschnitte zwischen zwei $subsection den $subsections zu
1301  $keys = array_values($keys);
1302  $failed = false;
1303
1304  if ($subsection['is_prev_section']) {
1305    if ($localKeyPoints[$keys[0]]['time_0'] - $localKeyPoints[$keys[0] - 1]['
        ↪ time_1'] < $globalTimeOnOneSpeed) {
```

```
1306        $failed = true;
1307      }
1308    }
1309
1310    if ($subsection['is_next_section']) {
1311      if ($localKeyPoints[end($keys) + 1]['time_0'] - $localKeyPoints[end($keys)
              ↪ ]['time_1'] < $globalTimeOnOneSpeed) {
1312        $failed = true;
1313      }
1314    }
1315
1316    for ($i = 1; $i < sizeof($keys); $i++) {
1317      if ($localKeyPoints[$keys[$i]]['time_0'] - $localKeyPoints[$keys[$i - 1]]['
              ↪ time_1'] < $globalTimeOnOneSpeed) {
1318        $failed = true;
1319        break;
1320      }
1321    }
1322
1323    if ($failed) {
1324      return array('fail' => true, 'keyPoints' => array());
1325    } else {
1326      foreach ($deletedKeyPoints as $index) {
1327        unset($localKeyPoints[$index]);
1328      }
1329
1330      return array('fail' => false, 'keyPoints' => $localKeyPoints);
1331    }
1332  }
1333
1334  // Erstellt mittels der $keyPoints die $subsections
1335  function createSubsections (array $localKeyPoints) {
1336
1337    global $globalTimeOnOneSpeed;
1338
1339    $keyPoints = $localKeyPoints;
1340    $subsections = array();
1341    $subsection = array('max_index' => null, 'indexes' => array(), '
              ↪ is_prev_section' => false, 'is_next_section' => false);
```

```
$maxIndex = null;

for($i = 0; $i < sizeof($keyPoints); $i++) {
  if ($i > 0) {
    if ($keyPoints[$i]['speed_0'] < $keyPoints[$i]['speed_1'] && $keyPoints[
        ↪ $i - 1]['speed_0'] > $keyPoints[$i - 1]['speed_1'] || $i == sizeof(
        ↪ $keyPoints) - 1) {
      if ($i == sizeof($keyPoints) - 1) {
        array_push($subsection['indexes'], $i);
      }

      array_push($subsections, $subsection);
      $subsection['indexes'] = array();
    }
  }

  if ($keyPoints[$i]['speed_0'] < $keyPoints[$i]['speed_1']) {
    $subsection['max_index'] = $i;
  }

  array_push($subsection['indexes'], $i);
}

// Überprüfung der Abschnitte zwischen zwei $subsections
for ($i = 1; $i < sizeof($subsections); $i++) {
  $firstIndex = $subsections[$i]['indexes'][array_key_first($subsections[$i][
      ↪ 'indexes'])];

  if ($keyPoints[$firstIndex]['time_0'] - $keyPoints[$firstIndex - 1]['time_1
      ↪ '] < $globalTimeOnOneSpeed) {
    $subsections[$i]['is_prev_section'] = true;
    $subsections[$i]['failed'] = true;
  } else {
    $subsections[$i]['is_prev_section'] = false;
    $subsections[$i]['failed'] = false;
  }
}

for ($i = sizeof($subsections) - 1; $i >= 0; $i--) {
```

```
    $isFirstSubsection = false;
    $isLastSubsection = false;

    if ($i == 0) {
      $isFirstSubsection = true;
    }

    if ($i == sizeof($subsections) - 1) {
      $isLastSubsection = true;
    }

    if ($subsections[$i]['failed'] || failOnSubsection($keyPoints, $subsections
        ↪ [$i])) {
      $subsections[$i]['failed'] = true;

      if (!$isFirstSubsection) {
        $subsections[$i]['is_prev_section'] = true;
      }

      if (!$isLastSubsection) {
        if (!$subsections[$i + 1]['is_prev_section']) {
          $subsections[$i]['is_next_section'] = true;
        }
      }
    } else {
      $subsections[$i]['failed'] = false;
    }
  }

  for ($i = 0; $i < sizeof($subsections); $i++) {
    if (!isset($subsections[$i]['max_index'])) {
      $subsections[$i]['brakes_only'] = true;
      $subsections[$i]['max_index'] = $subsections[$i]['indexes'][0];
    } else {
      $subsections[$i]['brakes_only'] = false;
    }
  }

  $subsections = array_values($subsections);
```

```
1415
1416   return array_reverse($subsections);
1417 }
1418
1419 // Überprüfung, ob es in einer $subsection zu einer Unterschreitung
1420 // der Mindestzeit kommt
1421 function failOnSubsection(array $keyPoints, array $subsection) {
1422
1423   global $globalTimeOnOneSpeed;
1424
1425   $failed = false;
1426
1427   for ($i = 1; $i < sizeof($subsection['indexes']); $i++) {
1428     if ($keyPoints[$subsection['indexes'][$i]]['time_0'] - $keyPoints[
          ↪ $subsection['indexes'][$i] - 1]['time_1'] < $globalTimeOnOneSpeed) {
1429       $failed = true;
1430       break;
1431     }
1432   }
1433
1434   return $failed;
1435 }
1436
1437 function toShortInSubsection (array $subsections) {
1438
1439   $foundError = false;
1440
1441   foreach ($subsections as $subsection) {
1442     if ($subsection['failed']) {
1443       $foundError = true;
1444       break;
1445     }
1446   }
1447
1448   return $foundError;
1449 }
1450
1451 function createCumulativeSections ($indexCurrentSection, $indexTargetSection,
      ↪ $currentPosition, $targetPosition, $next_lengths) {
```

```php

  $cumLength = array();
  $sum = 0;

  foreach ($next_lengths as $index => $value) {
    if ($index >= $indexCurrentSection) {
      $sum += $value;
      $cumLength[$index] = $sum;
    }
  }
  // Berechnung der kummulierten Start- und Endlängen der Abschnitte
  for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
    if ($indexCurrentSection == $indexTargetSection) {
      $cumulativeSectionLengthStart[$i] = 0;
      $cumulativeSectionLengthEnd[$i] = intval($targetPosition -
          ↪ $currentPosition);
    } else {
      if ($i == $indexCurrentSection) {
        $cumulativeSectionLengthStart[$i] = 0;
        $cumulativeSectionLengthEnd[$i] = intval($cumLength[$i] -
            ↪ $currentPosition);
      } else if ($i == $indexTargetSection) {
        $cumulativeSectionLengthStart[$i] = intval($cumLength[$i - 1] -
            ↪ $currentPosition);
        $cumulativeSectionLengthEnd[$i] = intval($cumLength[$i - 1] +
            ↪ $targetPosition - $currentPosition);
      } else {
        $cumulativeSectionLengthStart[$i] = intval($cumLength[$i - 1] -
            ↪ $currentPosition);
        $cumulativeSectionLengthEnd[$i] = intval($cumLength[$i] -
            ↪ $currentPosition);
      }
    }
  }

  return array($cumulativeSectionLengthStart, $cumulativeSectionLengthEnd);
}

function toArr(){
```

```
1485    return func_get_args();
1486  }
1487
1488  // Ermittelt die Echtzeitdaten für eine Gefahrenbremsung
1489  function emergencyBreak ($id, $distanceToNextStop = 0) {
1490
1491    global $allUsedTrains;
1492    global $timeDifference;
1493    global $allTimes;
1494
1495    $targetSpeed = 0;
1496    $returnArray = array();
1497    $time = microtime(true) + $timeDifference;
1498    $currentSpeed = $allUsedTrains[$id]['current_speed'];
1499    $notverzoegerung = $allUsedTrains[$id]['notverzoegerung'];
1500    $currentSection = $allUsedTrains[$id]['current_section'];
1501
1502    echo 'Der Zug mit der Adresse: ', $allUsedTrains[$id]['adresse'], " leitet
         ↪ jetzt eine Gefahrenbremsung ein.\n";
1503
1504    if (getBrakeDistance($currentSpeed, $targetSpeed, $notverzoegerung) <=
         ↪ $distanceToNextStop) {
1505      for ($i = $currentSpeed; $i >= 0; $i = $i - 2) {
1506        array_push($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));
1507        $time = $time + getBrakeTime($i, $i - 1, $notverzoegerung);
1508      }
1509    } else {
1510      $targetSpeedNotbremsung = getTargetBrakeSpeedWithDistanceAndStartSpeed(
           ↪ $distanceToNextStop, $notverzoegerung, $currentSpeed);
1511      $speedBeforeStop = intval($targetSpeedNotbremsung / 2) * 2;
1512
1513      if ($speedBeforeStop >= 10) {
1514        for ($i = $currentSpeed; $i >= 10; $i = $i - 2) {
```

```
1515        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));
1516        $time = $time + getBrakeTime($i, $i - 1, $notverzoegerung);
1517      }
1518      array_push($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' => $id, 'wendet' => false, '
              ↪ betriebsstelle' => 'Notbremsung', 'live_all_targets_reached' =>
              ↪ null));
1519    } else {
1520      array_push($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));
1521      $time = $time + getBrakeTime($currentSpeed, $currentSpeed - 1,
              ↪ $notverzoegerung);
1522      array_push($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));
1523    }
1524  }
1525
1526  $allTimes[$allUsedTrains[$id]['adresse']] = $returnArray;
1527  array_push($allUsedTrains[$id]['error'], 3);
1528
1529  return 0;
1530 }
```

A.4 functions_math.php

```php
<?php

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

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

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

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

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

A.5 functions_cache.php

```php
<?php

// Ermittelt die Längen aller Infra-Abschnitte
function createcacheInfraLaenge() {
  $DB = new DB_MySQL();
  $returnArray = array();
  $infralaenge = $DB->select('SELECT ``.DB_TABLE_INFRAZUSTAND.``.`id`, ``.
      ↪ DB_TABLE_INFRAZUSTAND.``.`laenge` FROM ``.DB_TABLE_INFRAZUSTAND.``
      ↪ WHERE ``.DB_TABLE_INFRAZUSTAND."``.`type` = '".'gleis'."'");
  unset($DB);
  foreach ($infralaenge as $data) {
    if ($data->laenge != null) {
      $returnArray[$data->id] = intval($data->laenge);
    }
  }
  return $returnArray;
}

// Ermittelt die zugehörigen Infra-Abschnitte zu den Betriebsstellen
function createCacheHaltepunkte() {

  $DB = new DB_MySQL();
  $returnArray = array();

  $betriebsstellen = $DB->select('SELECT ``.DB_TABLE_BETRIEBSSTELLEN_DATEN.``.`
      ↪ parent_kuerzel` FROM ``.DB_TABLE_BETRIEBSSTELLEN_DATEN.`` WHERE ``.
      ↪ DB_TABLE_BETRIEBSSTELLEN_DATEN.``.`parent_kuerzel` IS NOT NULL');
  unset($DB);

  foreach ($betriebsstellen as $betriebsstelle) {
    $returnArray[$betriebsstelle->parent_kuerzel][0] = array();
    $returnArray[$betriebsstelle->parent_kuerzel][1] = array();
  }

  foreach ($returnArray as $betriebsstelleKey => $betriebsstelleValue) {
    $DB = new DB_MySQL();
    $name = $betriebsstelleKey;
    $name .= '%';
```

```
35      $asig = 'ASig';
36      $bksig = 'BkSig';
37      $vsig = 'VSig';
38      $ja = 'ja';
39      if ($betriebsstelleKey == 'XAB' || $betriebsstelleKey == 'XBL') {
40        $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` = '$ja'");
41        unset($DB);
42      } else if ($betriebsstelleKey == 'XTS') {
43        $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` = '$bksig'");
44        unset($DB);
45      } else if ($betriebsstelleKey == 'XLG') {
46        $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` != '$vsig'");
47        unset($DB);
48      } else {
49        $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` = '$asig'");
50        unset($DB);
51      }
52
53      foreach ($haltepunkte as $haltepunkt) {
```

```
54        if ($haltepunkt->wirkrichtung == 0) {
55          array_push($returnArray[$betriebsstelleKey][0], intval($haltepunkt->
                ↪ freimelde_id));
56        } elseif ($haltepunkt->wirkrichtung == 1) {
57          array_push($returnArray[$betriebsstelleKey][1], intval($haltepunkt->
                ↪ freimelde_id));
58        }
59      }
60    }
61    $returnArray['XSC'][1] = array(734, 732, 735, 733, 692);
62    return $returnArray;
63  }
64
65  // Ermittelt die zugehörigen Infra-Abschnitte zu den Zwischen-Betriebsstellen
66  function createChacheZwischenhaltepunkte() {
67    $DB = new DB_MySQL();
68    $allZwischenhalte = array();
69    $returnArray = array();
70    $zwischenhalte = $DB->select('SELECT DISTINCT ``.DB_TABLE_SIGNALE_STANDORTE.'
          ↪ `.`betriebsstelle` FROM ``.DB_TABLE_SIGNALE_STANDORTE.`` WHERE ``.
          ↪ DB_TABLE_SIGNALE_STANDORTE.``.`betriebsstelle` IS NOT NULL');
71    unset($DB);
72    foreach ($zwischenhalte as $halt) {
73      array_push($allZwischenhalte, $halt->betriebsstelle);
74    }
75    foreach ($allZwischenhalte as $halt) {
76      $DB = new DB_MySQL();
77      $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');
78      unset($DB);
79      if (sizeof($zwischenhalte) == 1) {
80        if (sizeof(explode('_', $halt)) == 2) {
81          $returnArray[$halt] = intval($zwischenhalte[0]->freimelde_id);
82        }
83      }
84    }
85    return $returnArray;
```

```
86  }
87
88  // Ermittelt die Zuordnung zwischen den
89  // Infra-Abschnitten und den GBT-Abschnitten
90  function createCacheInfraToGbt () {
91    $DB = new DB_MySQL();
92    $infraArray = array();
93    $returnArray = array();
94    $allInfra = $DB->select('SELECT ``.DB_TABLE_FMA_GBT.``.`infra_id` FROM ``.
          ↪ DB_TABLE_FMA_GBT.`` WHERE ``.DB_TABLE_FMA_GBT.``.`infra_id` IS NOT
          ↪ NULL');
95    unset($DB);
96    foreach ($allInfra as $infra) {
97      array_push($infraArray, intval($infra->infra_id));
98    }
99    foreach ($infraArray as $infra) {
100     $DB = new DB_MySQL();
101     $gbt = $DB->select('SELECT ``.DB_TABLE_FMA_GBT.``.`gbt_id` FROM ``.
            ↪ DB_TABLE_FMA_GBT.`` WHERE ``.DB_TABLE_FMA_GBT."`.`infra_id` = '
            ↪ $infra'")[0]->gbt_id;
102     unset($DB);
103     $returnArray[$infra] = intval($gbt);
104   }
105   return $returnArray;
106 }
107
108 // Ermittelt die Zuordnung zwischen den
109 // GBT-Abschnitten und den Infra-Abschnitten
110 function createCacheGbtToInfra () {
111
112   $DB = new DB_MySQL();
113
114   $returnArray = array();
115
116   $allGbt = $DB->select('SELECT DISTINCT ``.DB_TABLE_FMA_GBT.``.`gbt_id` FROM `
          ↪ '.DB_TABLE_FMA_GBT.`` WHERE ``.DB_TABLE_FMA_GBT.``.`gbt_id` IS NOT
          ↪ NULL');
117   unset($DB);
118
```

```
119   foreach ($allGbt as $gbt) {
120     $DB = new DB_MySQL();
121     $gbt = $gbt->gbt_id;
122     $infras = $DB->select('SELECT ``.DB_TABLE_FMA_GBT.``.`infra_id` FROM ``.
          ↪ DB_TABLE_FMA_GBT.`` WHERE ``.DB_TABLE_FMA_GBT."`.`gbt_id` = '$gbt'")
          ↪ ;
123     unset($DB);
124     $returnArray[$gbt] = array();
125     foreach ($infras as $infra) {
126       array_push($returnArray[$gbt], intval($infra->infra_id));
127     }
128   }
129   return $returnArray;
130 }
131 
132 // Ermittelt die Zuordnung zwischen den
133 // FMA-Abschnitten und den Infra-Abschnitten
134 function createCacheFmaToInfra () {
135   $DB = new DB_MySQL();
136   $returnArray = array();
137   $fmaToInfra = $DB->select('SELECT ``.DB_TABLE_FMA_GBT.``.`infra_id`, ``.
          ↪ DB_TABLE_FMA_GBT.``.`fma_id` FROM ``.DB_TABLE_FMA_GBT.`` WHERE ``.
          ↪ DB_TABLE_FMA_GBT.``.`fma_id` IS NOT NULL');
138   unset($DB);
139   foreach ($fmaToInfra as $value) {
140     $returnArray[intval($value->fma_id)] = intval($value->infra_id);
141   }
142   return $returnArray;
143 }
144 
145 // Ordnet die Signal ID den Betriebsstellen zu
146 function createCacheToBetriebsstelle() {
147   $DB = new DB_MySQL();
148   $returnArray = array();
149   $fmaToInfra = $DB->select('SELECT ``.DB_TABLE_SIGNALE_STANDORTE.``.`id`, ``.
          ↪ DB_TABLE_SIGNALE_STANDORTE.``.`betriebsstelle` FROM ``.
          ↪ DB_TABLE_SIGNALE_STANDORTE.``');
150   unset($DB);
151   foreach ($fmaToInfra as $value) {
```

```
152     $returnArray[intval($value->id)] = $value->betriebsstelle;
153   }
154   return $returnArray;
155 }
156
157 // Ermittelt die Daten aus der Abschnittsüberwachung
158 function createCacheFahrzeugeAbschnitte () {
159   $DB = new DB_MySQL();
160   $returnArray = array();
161   $fahrzeugeAbschnitte = $DB->select('SELECT `'.DB_TABLE_FAHRZEUGE_ABSCHNITTE.'
        ↪ `.`fahrzeug_id`, `'.DB_TABLE_FAHRZEUGE_ABSCHNITTE.'`.`infra_id`, `'.
        ↪ DB_TABLE_FAHRZEUGE_ABSCHNITTE.'`.`unixtimestamp` FROM `'.
        ↪ DB_TABLE_FAHRZEUGE_ABSCHNITTE.'`');
162   unset($DB);
163   foreach ($fahrzeugeAbschnitte as $fahrzeug) {
164     $returnArray[intval($fahrzeug->fahrzeug_id)]['infra_id'] = intval($fahrzeug
          ↪ ->infra_id);
165     $returnArray[intval($fahrzeug->fahrzeug_id)]['unixtimestamp'] = intval(
          ↪ $fahrzeug->unixtimestamp);
166   }
167   return $returnArray;
168 }
169
170 // Ermittelt die Zuordnung zwischen Fahrzeug ID und Decoder-Adresse
171 function createCacheDecoderToAdresse () {
172   $DB = new DB_MySQL();
173   $returnArray = array();
174   $decoderToAdresse = $DB->select('SELECT `'.DB_TABLE_FAHRZEUGE.'`.`id`, `'.
        ↪ DB_TABLE_FAHRZEUGE.'`.`adresse` FROM `'.DB_TABLE_FAHRZEUGE.'`');
175   unset($DB);
176   foreach ($decoderToAdresse as $fahrzeug) {
177     $returnArray[intval($fahrzeug->id)] = intval($fahrzeug->adresse);
178   }
179   return $returnArray;
180 }
181
182 // Ermittelt die Daten der Fahrplansession
183 function createCacheFahrplanSession() {
184   $DB = new DB_MySQL();
```

```
185   $fahrplanData = $DB->select('SELECT * FROM ``.DB_TABLE_FAHRPLAN_SESSION.``
          ↪ WHERE ``.DB_TABLE_FAHRPLAN_SESSION."`.`status` = '".'1'."'");
186   unset($DB);
187
188   return $fahrplanData[0];
189 }
```

A.6 functions_db.php

```php
<?php

// Ermittelt die Daten aller Fahrzeuge.
function getAllTrains () {

  global $cacheAdresseToID;
  global $cacheIDToAdresse;
  global $globalTrainVMax;

  $allAdresses = getAllAdresses();
  $DB = new DB_MySQL();
  $allTrains = array();
  $id = null;

  foreach ($allAdresses as $adress) {
    $train_fahrzeuge = get_object_vars($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]);
    $id = $train_fahrzeuge['id'];
    $train_daten = $DB->select('SELECT ``.DB_TABLE_FAHRZEUGE_DATEN.``.`baureihe
        ↪ ` FROM ``.DB_TABLE_FAHRZEUGE_DATEN.`` WHERE ``.
        ↪ DB_TABLE_FAHRZEUGE_DATEN."`.`id` = $id")[0]->baureihe;
    $train_baureihe = $DB->select('SELECT ``.DB_TABLE_FAHRZEUGE_BAUREIHEN.``.`
        ↪ vmax` FROM ``.DB_TABLE_FAHRZEUGE_BAUREIHEN.`` WHERE ``.
        ↪ DB_TABLE_FAHRZEUGE_BAUREIHEN."`.`nummer` = $train_daten");

    if (sizeof($train_baureihe) != 0) {
      $train_baureihe_return['v_max'] = intval($train_baureihe[0]->vmax);
    } else {
      $train_baureihe_return['v_max'] = $globalTrainVMax;
    }

    $id = intval($train_fahrzeuge['id']);
    $cacheAdresseToID[intval($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
123 // einen Abschnitt und eine 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 $signal;
135 }
136
137 // Kalibriert die Position des Fahrzeugs anhand der Daten in der Tabelle
138 // 'fahrzeuge_abschnitte' neu.
139 function getCalibratedPosition ($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 () {
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

```php
<?php

// Ermittlung der exakten Ankunftszeit
$useSpeedFineTuning = true;
// Mindestzeit für Beharrungsfahrten verwenden
$useMinTimeOnSpeed = true;
// Fehlermeldung, wenn die Mindestzeit für
// Beharrungsfahrten nicht eingehalten werden kann
$errorMinTimeOnSpeed = false;
// Anpassung der Geschwindigkeit, wenn das Fahrzeug
// zu früh an der Ziel-Betriebsstellen ankommen würde
$slowDownIfTooEarly = true;
// Neukalibrierung der Position verrwenden
$useRecalibration = true;

// Bremsverzögerung bei einer Gefahrenbremsung [m/s^2]
$globalNotverzoegerung = 2;
// Maximale Geschwindigkeit, wenn keine vorgegeben ist [km/h]
$globalTrainVMax = 10;
// Maximale Geschwindigkeit im aktuellen Abschnitt [km/h]
$globalSpeedInCurrentSection = 60;
// Mindesthaltezeit am ersten fahrplanmäßigen Halt [s]
$globalFirstHaltMinTime = 20;
// Kulanzbereich beim Rechnen mit float-Werten
$globalFloatingPointNumbersRoundingError = 0.0000000001;
// Mindestzeit für Beharrungsfahrten [s]
$globalTimeOnOneSpeed = 20;
// Distanzintervall für Beharrungsfahrten [m]
$globalDistanceUpdateInterval = 1;
```

A.8 speed_over_position.m

```
%% Liest die Daten der kummulierten Infra-Abschnitte ein

fname = '../json/VMaxOverCumulativeSections.json';
fid = fopen(fname);
raw = fread(fid,inf);
str = char(raw');
fclose(fid);
vmaxOverPosition = jsondecode(str);
vmaxOverPosition_Position = vmaxOverPosition(:,1);
vmaxOverPosition_v_max = vmaxOverPosition(:,2);

%% Liest die Daten der kummulierten Infra-Abschnitte
%% inklusive der Fahrzeuglänge ein

fname = '../json/VMaxOverCumulativeSectionsMod.json';
fid = fopen(fname);
raw = fread(fid,inf);
str = char(raw');
fclose(fid);
vmaxOverPosition_mod = jsondecode(str);
vmaxOverPosition_Position_mod = vmaxOverPosition_mod(:,1);
vmaxOverPosition_v_max_mod = vmaxOverPosition_mod(:,2);

%% Liest die Echtzeitdaten ein

fname = '../json/speedOverPosition.json';
fid = fopen(fname);
raw = fread(fid,inf);
str = char(raw');
fclose(fid);
val = jsondecode(str);

speedOverPosition_x_v1 = val(:,1);
speedOverPosition_y_v1 = val(:,2);

%% Liest die Echtzeitdaten der vorherigen Iterationsschritte ein

fname_it = '../json/speedOverPosition_prevIterations.json';
```

```
fid_it = fopen(fname_it);
raw_it = fread(fid_it,inf);
str_it = char(raw_it');
fclose(fid_it);
val_it = jsondecode(str_it);

%% Plot

hold on
figure(1)

% Plottet die Infra-Abschnitte

p = line([0 0], [0 vmaxOverPosition_v_max(1)],'Linestyle','-.','LineWidth',2,'color','black','DisplayName',['Infra-Abschnitte']);
line([0 vmaxOverPosition_Position(1)], [vmaxOverPosition_v_max(1) vmaxOverPosition_v_max(1)],'Linestyle','-.','LineWidth',2,'color','black', 'HandleVisibility','off');

for i = 1:size(vmaxOverPosition_Position) - 1
   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');
   line([vmaxOverPosition_Position(i) vmaxOverPosition_Position(i)], [0 vmaxOverPosition_v_max(i + 1)],'Linestyle','-.','LineWidth',2,'color','black', 'HandleVisibility','off');
   line([vmaxOverPosition_Position(i) vmaxOverPosition_Position(i)], [0 vmaxOverPosition_v_max(i)],'Linestyle','-.','LineWidth',2,'color','black', 'HandleVisibility','off');
   line([vmaxOverPosition_Position(i + 1) vmaxOverPosition_Position(i + 1)], [0  vmaxOverPosition_v_max(i + 1)],'Linestyle','-.','LineWidth',2,'color','black', 'HandleVisibility','off');
end

% Plottet die kummulierten Infra-Abschnitte inklusive der Fahrzeuglänge
line([0 0], [0 vmaxOverPosition_v_max_mod(1)],'Linestyle','-.','LineWidth',2,'color','red', 'HandleVisibility','off');
```

```
64 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']);
65
66 for i = 1:size(vmaxOverPosition_Position_mod) - 1
67    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');
68    line([vmaxOverPosition_Position_mod(i) vmaxOverPosition_Position_mod(i)], [0
          ↪  vmaxOverPosition_v_max_mod(i + 1)],'Linestyle','-.','LineWidth',2,'
          ↪ color','red', 'HandleVisibility','off');
69    line([vmaxOverPosition_Position_mod(i) vmaxOverPosition_Position_mod(i)], [0
          ↪  vmaxOverPosition_v_max_mod(i)],'Linestyle','-.','LineWidth',2,'color
          ↪ ','red', 'HandleVisibility','off');
70    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');
71 end
72
73 % Plottet alle Iterationsschritte der Echtzeitdaten
74 for i = 1:length(val_it)
75     %plot(val_it{i}(:,1),val_it{i}(:,2),'.','markersize',8,'Color', [0.6 0.6
           ↪ 0.6],'DisplayName',legend_name);
76
77 end
78
79 % Plottet die Echtzeitdaten
80
81 plot(speedOverPosition_x_v1,speedOverPosition_y_v1,'LineWidth',4,'Color', [0.25
       ↪  0.80 0.54],'DisplayName','Fahrtverlauf');
82
83 %% Ausfüllobjekte zur besseren Darstellung
84
85 %fill([0, 0, 1090, 1090, 0], [0, 200, 200, 0, 0], 'b','facealpha',.2,'LineStyle
       ↪ ','none');
86
87 %% Text hinzufügen zur bessern Darstellung
88
```

```
%text(20,17,'1','Interpreter','latex','fontsize', 40);

%% Plot formatieren

p.LineWidth = 2;
box off
fontSize = 18;
xlabel("Strecke [m]", 'FontSize', fontSize);
ylabel("Geschwindigkeit [km/h]", 'FontSize', fontSize);
x0=10;
y0=10;
width=1100;
height=600;
axis([-80 max(vmaxOverPosition_Position)+80 0 max(vmaxOverPosition_v_max)+10]);
axis([-20 max(vmaxOverPosition_Position)+20 0 max(vmaxOverPosition_v_max)+5]);
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'PaperPositionMode','auto');
set(gca, 'FontSize', 18);
set(gca, 'Linewidth', 2);

t = gca;
exportgraphics(t,'SpeedOverPosition.pdf','ContentType','vector');
hold off
```

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Die selbständige und eigenständige Anfertigung versichert an Eides statt:

Berlin, 30. September 2021	
Ort, Datum	Unterschrift