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, 21. 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.

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

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

ToDo-Liste

- Aurelia korrigieren
- E-Mail wegen Prüfung
- Kapitel zur Fehlerbehandlung
- Start des Programms

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Code-Beispiele

Abkürzungsverzeichnis

EBuEf Eisenbahn-Betriebs- und Experimentierfeld

Infra-Abschnitt Infrastrukturabschnitt

Glossar

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

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

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

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

Realzeit Die Realzeit beschreibt die Zeit des Rechners/Servers, auf dem die 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 Dokumentierung werden in Kapitel 2 die Grundlagen, die Ausgangssituation, die Herangehensweise und die Ziele beschrieben. Die Funktionsweise der Fahrzeugsteuerung wird in Kapitel 3 in chronologischer Form beschrieben, wobei im Kapitel 4 die Ermittlung des Fahrtverlaufs im Detail beschrieben wird. Damit die Allgemeingültigkeit der 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 mit Hilfe der in Kapitel 7 hergeleiteten Formeln auf die Richtigkeit überprüft.

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

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

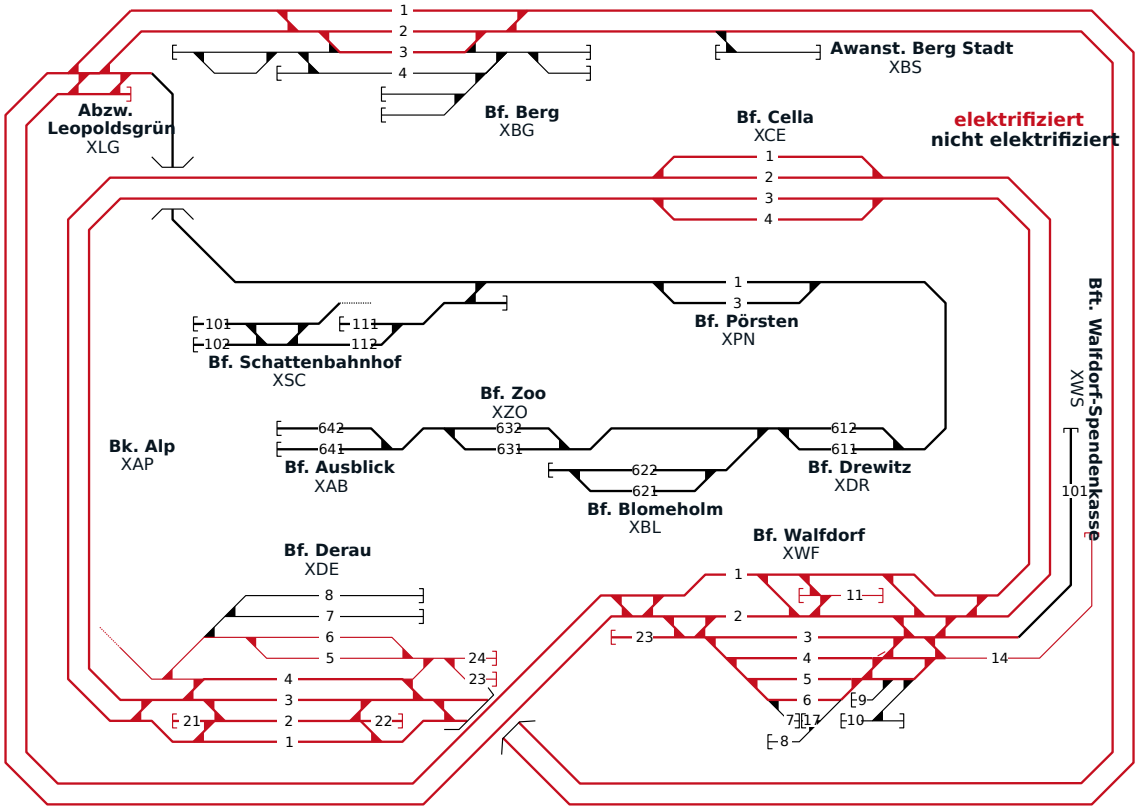

Abbildung 1: Schienennetz des EBuEfs (Quelle: www.ebuef.de/das-betriebsfeld/stellwerke; Letzter Zugriff am: 4. September 2021)

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 ist das eingleisige Netz in schwarz dargestellt und die zweigleisige Hauptstrecke in rot. Das eingleisige Netz ist in Infrastrukturabschnitte (Infra-Abschnitte) – welche mit Blockstrecken vergleichbar sind und eine Zugfolge im festen Raumabstand ermöglichen – eingeteilt.[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 essentiell 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 Geschwindigkeit zu fahren als die maximal zulässige Geschwindigkeit, wird die Geschwindigkeit möglichst am Ende des Fahrtverlaufs reduziert. Dadurch hat das Fahrzeug für den Fall einer Fahrstraßenänderung oder Reduzierung der zulässigen Höchstgeschwindigkeit einen größtmöglichen Zeitpuffer.

2.4 Fahrdynamik

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

- Anfahren
- Beharrungsfahrt
- Auslauf
- Bremsen

Beim Anfahren ist die Antriebskraft größer als die Summe der Widerstandskräfte, wodurch das Fahrzeug beschleunigt und in der Beharrungsfahrt entspricht die Antriebskraft der Summe der Widerstandskräfte, wodurch die Geschwindigkeit des Fahrzeugs konstant bleibt. Für die Reduzierung der Geschwindigkeit kann entweder die Antriebskraft gleich null sein oder eine Bremskraft aufgewendet werden.[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 der Widerstandskräften abhängig ist, wird ebenfalls nicht berücksichtigt.

2.5 Aufbau des Projekts

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

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

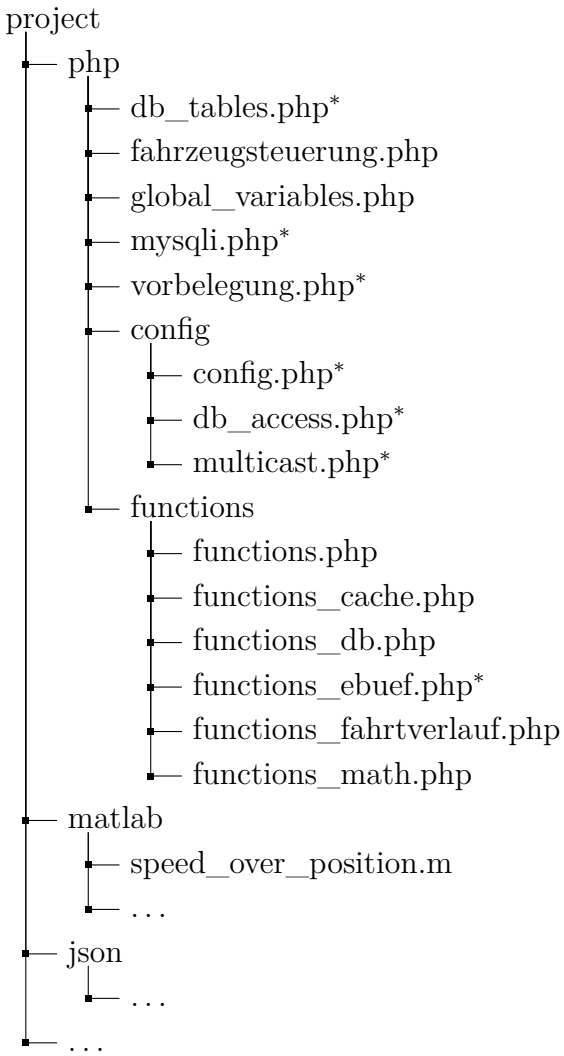

Abbildung 2: Aufbau der Dateistrukturen

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

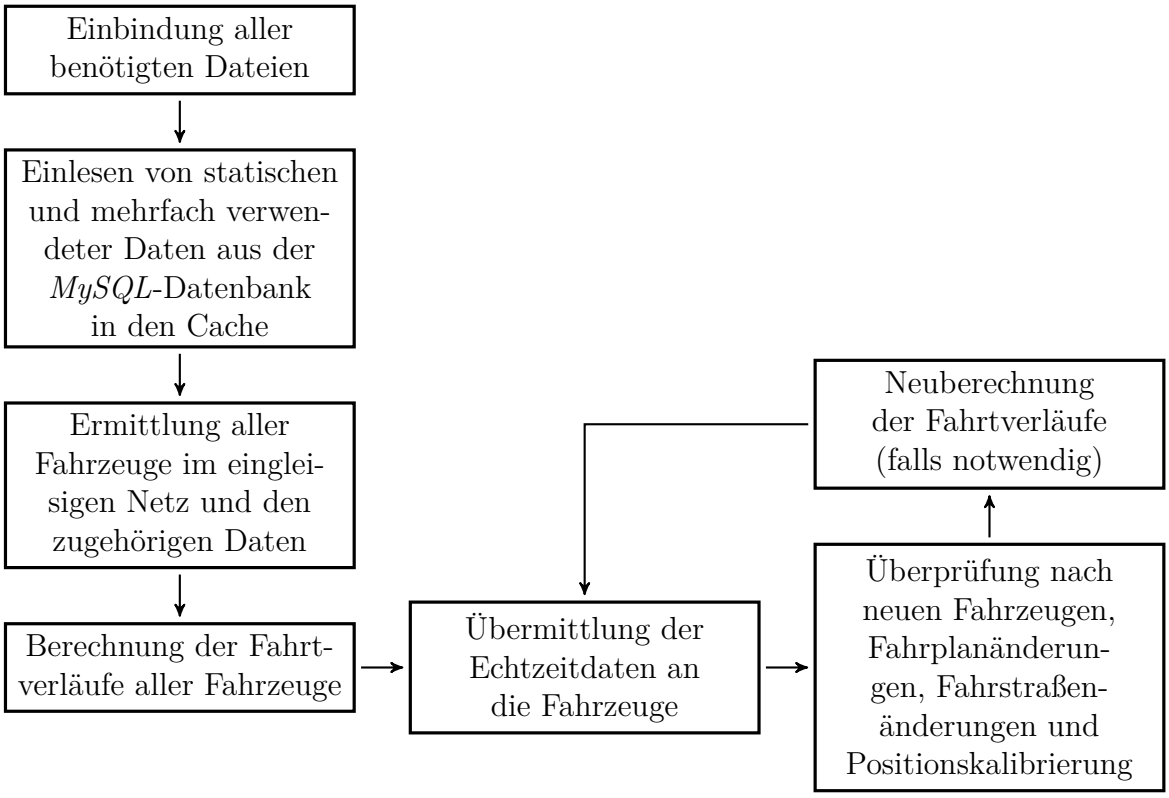

Abbildung 3: Ablauf der Fahrzeugsteuerung

3 Ablauf der Fahrzeugsteuerung

Damit die Fahrzeugsteuerung 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 wird.

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

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

start bzw. bei der ersten Verwendung in den Cache geladen (Code-Beispiel 1). 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);
$cacheAdresseToID = array();  // Filled with data in getAllTrains()
$cacheIDToAdresse = array();  // Filled with data in getAllTrains()
```

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

3.2 Ermittlung der Session-Daten

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

```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)*

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. Es wird davon ausgegangen, dass das Fahrzeug sich direkt vor dem Signal befindet, wodurch die relative Position der Infra-Abschnittslänge entspricht.

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

Bezeichnung	Funktion
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*

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

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

Bezeichnung	Funktion
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*

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 *$nextBetriebsstelleIndex* abgespeichert. Sollte unter den nächsten Betriebsstellen keine dabei sein, auf die diese Kriterien zutreffen, wird in einer zweiten *for*-Schleife nach den selben Kriterien (außer dem des fahrplanmäßigen Halts) nach einer Betriebsstelle gesucht und sobald eine Betriebsstelle gefunden wurde, wird die Schleife abgebrochen und der Index der Betriebsstelle unter der Variablen *$nextBetriebsstelleIndex* abgespeichert. Sollte eine nächste Betriebsstelle für das Fahrzeug existieren wird in einer dritten *for*-Schleife überprüft, ob zwischen der aktuellen Position und der nächsten Betriebsstelle eine Betriebsstelle ist, bei der das Fahrzeug einen Wendeauftrag bekommt. Sollte eine solche Betriebsstelle existieren, wird diese unter der Variablen *$nextBetriebsstelleIndex* abgespeichert. In dem Fall, dass keine nächste Betriebsstelle ermittelt werden konnte und das Fahrzeug aktuell eine Geschwindigkeit hat, für die gilt: $v > 0\ km/h$, wird eine Gefahrenbremsung eingeleitet (siehe Kapitel 4.8).

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

zur Zeit ermittelt werden.

Für die Ermittlung der Ankunftszeit muss neben dem zugehörigen Eintrag *ankunft_soll_timestamp* der Betriebsstelle die Verspätung berücksichtigt werden. Aus diesem Grund wird im ersten Schritt die zuletzt angefahren Betriebsstelle unter der Variablen *$prevBetriebsstelle* abgespeichert. Sollte die nächste Betriebsstelle der erste fahrplanmäßige Halt sein (Ankunftszeit nicht definiert), so wird als Start- und Zielzeit (*$startTime* und *$endTime*) die aktuelle Simulationszeit verwendet. Wenn die nächste Betriebsstelle nicht dem ersten fahrplanmäßigen Halt entspricht, wird als Zielzeit die Ankunftszeit der Betriebsstelle festgelegt und als Startzeit die Abfahrtszeit der vorherigen Betriebsstelle (*$prevBetriebsstelle*) plus die eingetragene Verspätung der vorherigen Betriebsstelle. Sollte es zu dem Zeitpunkt der Berechnung keine vorherige Betriebsstelle geben (*$prevBetriebsstelle* == *null*), so wird als Startzeit die aktuelle Simulationszeit gewählt. Im zweiten Schritt wird überprüft, ob die Startzeit kleiner als die aktuelle Simulationszeit ist und wenn das der Fall ist, wird die Startzeit gleich der Simulationszeit gesetzt. Im dritten Schritt wird die Startzeit gleich der frühstmöglichen 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 frühstmögliche 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 ein Wendeauftrag nach dem Erreichen des Ziel-Infra-Abschnitt dem Fahrzeug übermittelt werden soll, wird von dem Signalbegriff abgeleitet. Die Start- und Zielzeit entsprechen der aktuellen Simulationszeit, bzw. der *earliest_possible_start_time*. Sollte keinem der nächsten Infra-Abschnitte aus dem *last_get_naechste_abschnitte*-Array ein Signal zugeordnet sein und die aktuelle Geschwindigkeit des Fahrzeugs ist größer als 0 km/h sein, so wird eine Gefahrenbremsung eingeleitet. Andernfalls wird die Funktion an dieser Stelle abgebrochen und es wird wieder versucht einen Fahrtverlauf zu berechnen, wenn sich die Fahrstraße geändert hat.

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

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

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

3.5 Übermittlung der Echtzeitdaten an die Fahrzeuge

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

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

Bezeichnung	Funktion
live_position (Float)	absolute Position (kann weg...)
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

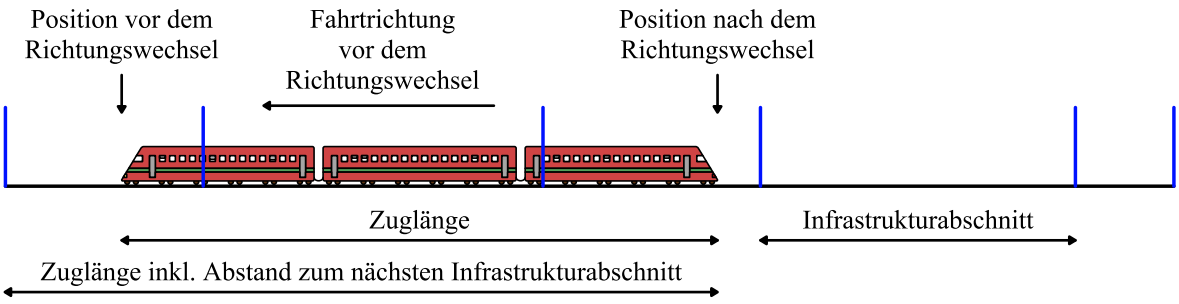

Abbildung 4: Eigene Darstellung der Positionsbestimmung bei einem Richtungswechsel

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

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

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

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

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

Tabelle 5: Verhalten eines Fahrzeugs nach dem Erreichen des Ziels

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

Wenn die letzten Echtzeitdaten eines Fahrzeugs übermittelt wurden (*live_target_reached == true*) und das Fahrzeug dementsprechend zum Stehen gekommen ist, wird überprüft, wie sich das Fahrzeug als nächstes verhalten soll. Dafür wird zwischen vier Fällen (siehe Tabelle 5) unterschieden. Für die Überprüfung, ob sich der Fahrplan eines Fahrzeugs geändert hat, wird über die Funktion *getFahrzeugZugIds*() (*functions_ebuef.php*) die aktuelle Zug-ID abgefragt und mit der vorherigen verglichen. In dem **1. Fall** (alte und neue Zug-ID haben beide den Wert *null*) werden dem Fahrzeug keine neue Daten übergeben und ein neuer Fahrtverlauf wird versucht zu berechnen, sobald die Fahrstraße sich verändert hat. In dem **2.** und **4. Fall** wird die neue Zug-ID dem Fahrzeug übergeben, der Eintrag *operates_on_timetable* auf *true* gesetzt und die Funktionen *getFahrplanAndPositionForOneTrain*() (*functions.php*), *addStopsectionsForTimetable*() (*functions.php*), *calculateNextSections*() (*functions_fahrtverlauf.php*), *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 Überprüfung nach einer Änderung der Fahrstraße

Für die Überprüfung, ob sich die Fahrstraße der Züge verändert hat, wird in regelmäßigen Abständen die Fahrstraße der Fahrzeuge ermittelt und mit der aktuell hinterlegten Fahrstraße verglichen. Das Intervall, in dem diese Überprüfung stattfindet kann über die Variable *$timeCheckFahrstrasseInterval* (*fahrzeugsteuerung.php*) festgelegt werden und ist standardgemäß auf 3 Sekunden festgelegt. Bei der Ermittlung und dem Vergleich der Fahrstraße wird für jedes Fahrzeug die Funktion *compareTwoNaechsteAbschnitte*() (*functions.php*) aufgerufen. Innerhalb dieser Funktion wird die in Kapitel 3.3 erläutere Funktion *calculateNextSections*() (*functions.php*) aufgerufen, mit dem Unterschied, dass die ermittelten nächsten Infra-Abschnitte inkl. der Längen und zulässigen Höchstgeschwindigkeiten nicht dem Fahrzeug hinterlegt werden, sondern lokal in der Funktion gespeichert. Damit die ermittelten Daten für ein Fahrzeug berechnet werden, aber nicht dem Fahrzeug hinterlegt werden, kann der Parameter *$writeResultToTrain* der Funktion *calculateNextSections*() (*functions.php*) (standardgemäß auf *true* gesetzt) auf *false* gesetzt werden. Sollte sich die Fahrstraße geändert haben, wird mit der Funktion *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 genau Fahrzeugsteuerung ist die aktuelle Position der Züge essenziell und muss während der Fahrt kalibriert werden, damit Ungenauigkeiten ausgeglichen werden können. Dafür werden die Daten aus der *MySQL*-Tabelle *fahrzeuge_abschnitte* benötigt, welche durch die Abschnittsüberwachung ermittelt werden. Die Abschnittsüberwachung schreibt für jedes Fahrzeug den aktuellen Infra-Abschnitt in die Datenbank, sobald der Zugkopf den Abschnitt befährt inklusive der aktuellen Zeit (Realzeit). Für jedes Fahrzeug, welches durch die Übermittlung der Echtzeitdaten in einen neuen Infra-Abschnitt einfährt und seit der Einfahrt in den Abschnitt die Geschwindigkeit nicht verändert hat, wird die aktuelle Position neu ermittelt. Würde sich das Fahrzeug in einem Infra-Abschnitt befinden und hätte seit der Einfahrt die Geschwindigkeit angepasst, könnte mit der Fahrzeugsteuerung die Position nicht neu berechnet werden, da nicht bekannt ist, welche Strecke das Fahrzeug seit der Einfahrt zurückgelegt hat. Aus diesem Grund wird, sobald das Fahrzeug nach den Echtzeitdaten einen neuen Abschnitt befährt und aktuell nicht die Geschwindigkeit anpasst (*live_is_speed_change* == *false*) dem Eintrag *calibrate_section_one* der aktuelle Infra-Abschnitt hinzugefügt und dem Eintrag *calibrate_section_two* wird ebenfalls der aktuelle Infra-Abschnitt

```php
// Kalibriert die Position des Fahrzeugs neu anhand der Daten in der Tabelle
// 'fahrzeuge_abschnitte'
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*)

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

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

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

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

relative Position = Geschwindigkeit $\cdot$ 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 befinden auch alle Fahrzeuge, die nach dem Programmstart hinzugefügt werden. Für alle Fahrzeuge, die beim Start des Programms erkannt werden, wird in dem Array *$allTrainsOnTheTrack* die zugehörige Adresse gespeichert (*findTrainsOnTheTracks*()) (*functions.php*). Für die Überprüfung, ob Fahrzeuge entfernt wurden oder neu hinzugekommen sind, wird die Funktion *updateAllTrainsOnTheTrack*() (*functions.php*) verwendet. Diese Funktion wird – wie die Neukalibrierung in Kapitel 3.7 – alle 3 Sekunden ausgeführt. Bei dem Aufruf der Funktion werden alle Fahrzeuge geladen, denen in der *fma*-Tabelle aus der Datenbank ein Infra-Abschnitt zugeordnet ist und mit dem Array *$allTrainsOnTheTrack* verglichen. Fahrzeugadressen, die nicht in dem Array hinterlegt sind, werden in dem Rückgabe-Array unter dem Eintrag *new* zurückgegeben und alle Fahrzeugadressen, die in dem Array enthalten sind, aber bei dem Aufruf der Funktion keinem Infra-Abschnitt zugeordnet sind, werden in dem Rückgabe-Array unter dem Eintrag *removed* zurückgegeben. Nach dem Aufruf der Funktion, werden für alle neuen Fahrzeuge die Funktion *prepareTrainForRide*() (*functions.php*), *addStopsectionsForTimetable*() (*functions.php*), *calculateNextSections*() (*functions.php*), *checkIfTrainReachedHaltepunkt*() (*functions.php*), *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 Fehlerbehebung von Fahrzeugen

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

```php
// Gibt für alle Fahrzeuge die vorhanden Fehlermeldungen 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*)

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

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

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

Tabelle 6: Verhalten eines Fahrzeugs nach dem Erreichen des Ziels

4 Berechnung des Fahrtverlaufs

Der Fahrtverlauf eines Fahrzeuges wird bei der Berechnung auf zwei verschiedenen Arten gespeichert. Einmal in so genannten *$keyPoints*, welche in einem Array die Start- und Zielgeschwindigkeit (*speed_0* und *speed_1*), die Start- und Endposition (*position_0* und *position_1*) und die Start- und Endzeit (*time_0* und *time_1*) der einzelnen Beschleunigungen bzw. Verzögerungen abspeichern. Für die Überprüfung, ob ein Fahrzeug die zulässige Höchstgeschwindigkeit in einem Infra-Abschnitt überschreitet, für die spätere Übermittlung der Echtzeitdaten an das Fahrzeug und die exakte Positionsbestimmung, werden mittels der *$keyPoints* für jede Geschwindigkeitsänderungen (und bei Beharrungsfahrten in 1 Meter Abständen) unter anderem die aktuelle relative Position innerhalb eines Infra-Abschnitts (*$trainRelativePosition*), der Infra-Abschnitt (*$trainSection*), die aktuelle Zeit (*$trainTimeChange*) und die aktuelle Geschwindigkeit (*$trainSpeedChange*) in gespeichert.

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.

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

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 Endposition der einzelnen 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. Als exemplarisch gewählte Zugdaten wurden die in Tabelle 9 definierten Daten verwendet.

Die zuvor ermittelten nächsten Infra-Abschnitte inklusive derer Längen und zulässigen Höchstgeschwindigkeit müssen für die Berechnung des Fahrtverlaufs angepasst werden, da ein Fahrzeug erst beschleunigen darf, wenn das komplette Fahrzeug in den Infra-Abschnitt eingefahren ist. In Darstellung 6 sind die Infra-Abschnitte dargestellt, wie sie von der Fahrzeugsteuerung ermittelt wurden. Dabei werden alle Infra-Abschnitte, die das Fahrzeug bereits durchfahren hat oder hinter dem Ziel-Infra-Abschnitt liegen nicht dargestellt. Zudem wird in dem aktuellen Infra-Abschnitt die relative Position von der Länge abgezogen und der Ziel-Infra-Abschnitt wird nur

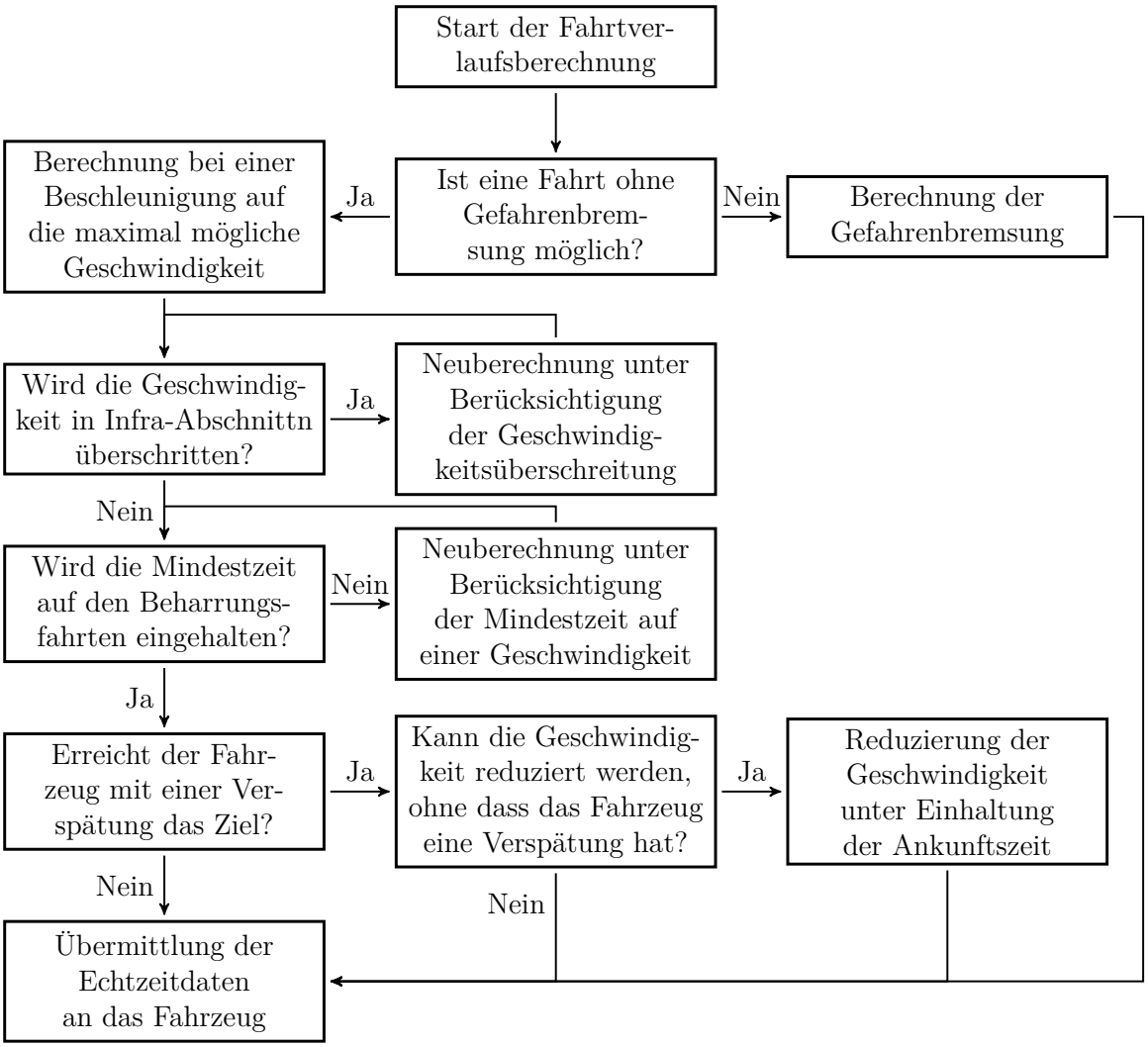

Abbildung 5: Ablaufplan der Fahrtverlaufsberechnung

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

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

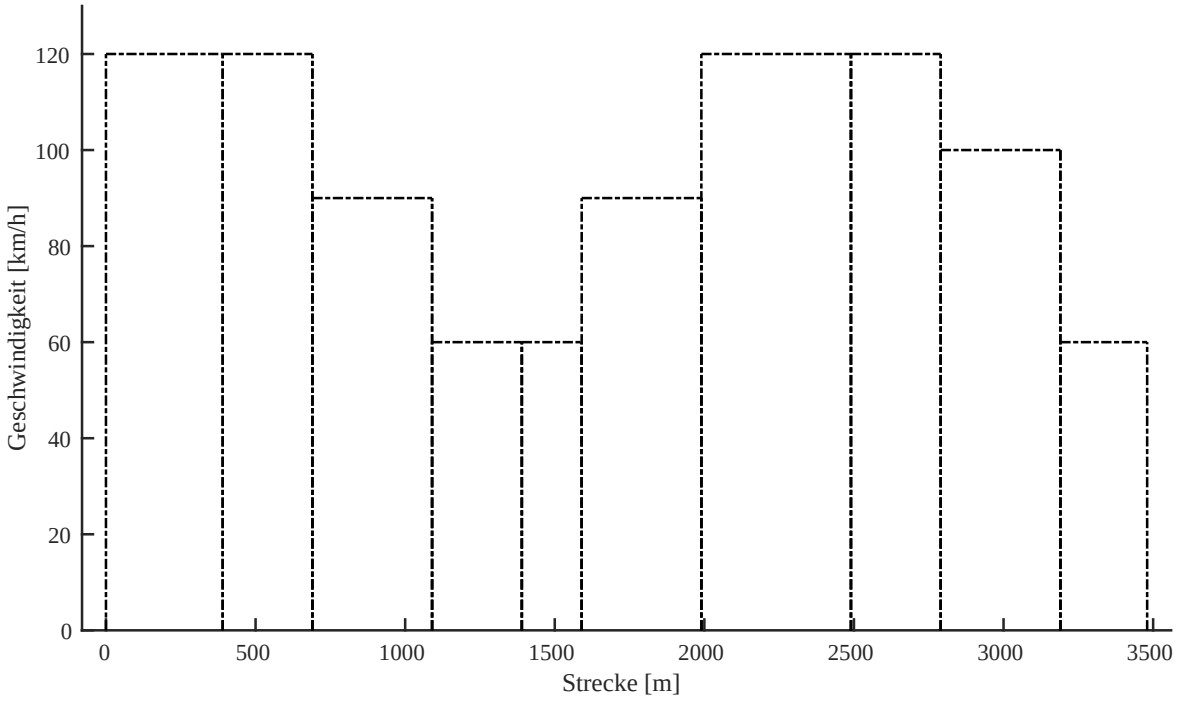

Abbildung 6: Infra-Abschnitte und die zugehörige Höchstgeschwindigkeit

bis zur relativen Zielposition abgebildet. Dementsprechend ist der erste Infra-Abschnitt in der Darstellung 6 der Infra-Abschnitt mit der ID 1001. Dieser hat aufgrund der aktuellen relativen Position des Fahrzeugs eine Länge von 290 m. Und der letzte Infra-Abschnitt ist der Infra-Abschnitt mit der ID 1010 und einer Länge von 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 sich befindet. Sobald der Ziel-Infra-Abschnitt erreicht wurde, wird die Schleife abgebrochen. Die neu ermittelten Infra-Abschnitte werden in den Arrays *$next_lengths_mod* und *$next_v_max_mod* abgespeichert (analog zu den Arrays *$next_lengths* und *$next_v_max*).

Durch diesen Algorithmus kann es dazu kommen, dass sich die Anzahl der Infra-Abschnitte verändert hat, wodurch die Infra-Abschnitte nicht mehr eindeutig mit der Infrastruktur-

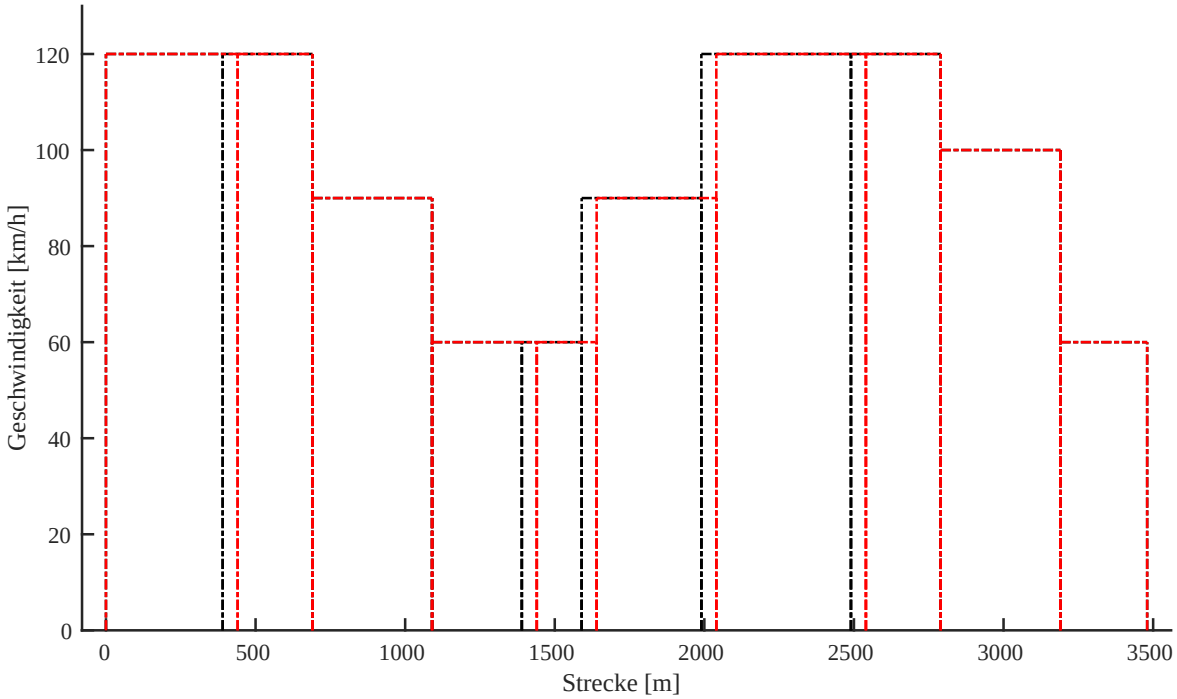

Abbildung 7: Infra-Abschnitte und die zugehörige Höchstgeschwindigkeit unter Berücksichtigung der Fahrzeuglänge

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

4.2 Berechnung bei einer Beschleunigung auf die maximal mögliche Geschwindigkeit

Im ersten Schritt der Fahrtverlaufsberechnung wird die Distanz zwischen der aktuellen Position und der Ziel-Position mittels *$cumulativeSectionLengthStart*, *$cumulativeSectionLengthEnd*, *$indexCurrentSection* und *$indexTargetSection* berechnet. Für die Distanz und die Startgeschwindigkeit wird mit Hilfe der Funktion *getVMaxBetweenTwoPoints*() (*functions_fahrtverlauf.php*) (Code-Beispiel 5) die maximale Geschwindigkeit ermittelt, auf die das Fahrzeug beschleunigen kann, um bis zum Ziel rechtzeitig bremsen zu können. Dabei wird in 10 km/h-Schritten iteriert und der maximale Wert

zurückgegeben. Innerhalb der Funktion wir die Funktion *getBrakeDistance*() (*functions_math.php*) (Code-Beispiel 8) 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 *$keyPoints* erzeugt werden. Mithilfe der Funktion *createTrainChanges*() (*functions_fahrtverlauf.php*) wird aus diesen beiden *$keyPoints* für jede Geschwindigkeitsveränderung die aktuelle absolute Position und Geschwindigkeit ermittelt. An den Positionen, an denen das Fahrzeug eine konstante Geschwindigkeit hat, wird in 1 Meter Abständen die absolute Position und die Geschwindigkeit gespeichert. Die ermittelten Daten werden in den Arrays *$trainPositionChange* und *$trainSpeedChange* gespeichert und sind in der Darstellung 8 abgebildet.

4.3 Überprüfung des Fahrtverlaufs nach Geschwindigkeitsüberschreitungen

Für die Überprüfung, ob es bei einem Fahrtverlauf zu einer Überschreitung der zulässigen Höchstgeschwindigkeit kommt, wird nach jeder Berechnung die Funktion *checkIfTrainIsToFastInCertainSections*() (*functions_fahrtverlauf.php*) (Code-Beispiel 6) 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*).

4.4 Neuberechnung unter Berücksichtigung der Geschwindigkeitsüberschreitung

In dem Fall, dass es zu einer Geschwindigkeitsüberschreitung gekommen ist, wird der Fahrtverlauf neu berechnet. Als Grundlage dafür dienen die *failed_sections* aus der *checkIfTrainIsToFastInCertainSections*()-Funktion (*functions_fahrtverlauf.php*) (Code-

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

  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*)

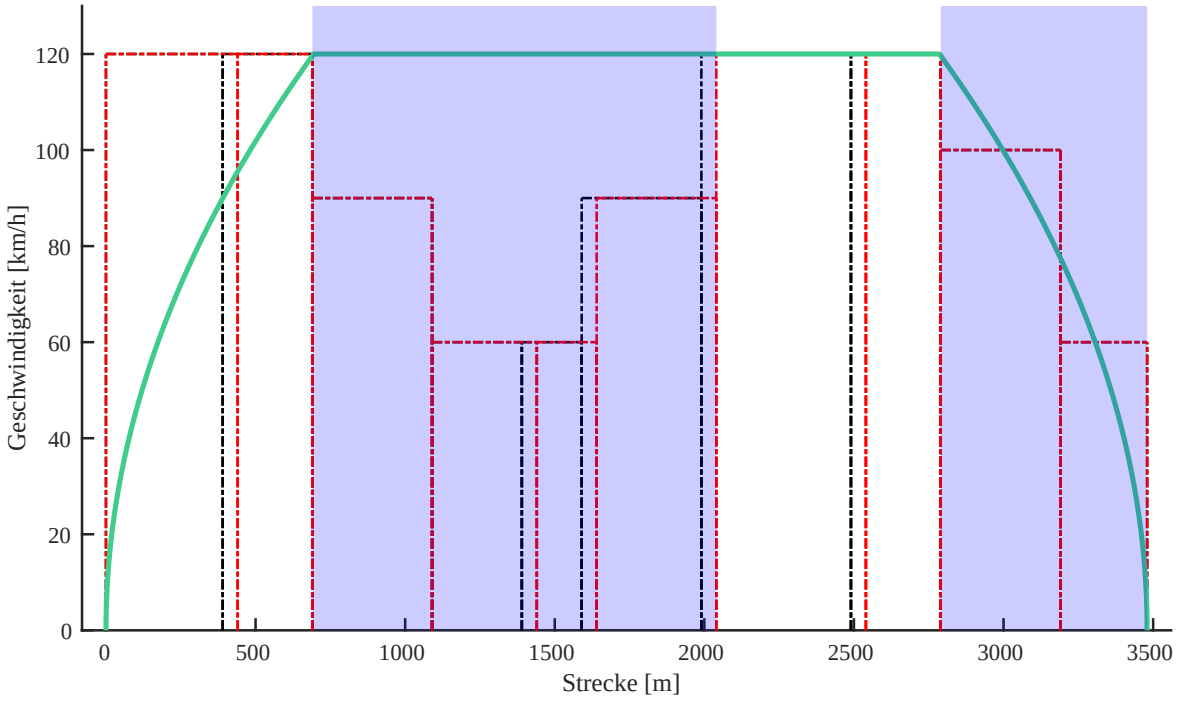

Abbildung 8: Fahrtverlaufsberechnung (1. Iterationsschritt)

Beispiel 6). Die Funktion *recalculateKeyPoints*() (*functions_fahrtverlauf.php*) vergleicht dabei immer zwei benachbarte *$keyPoints* und berechnet in dem Fall einer Geschwindigkeitsüberschreitung mit der Funktion *checkBetweenTwoKeyPoints*() (*functions_fahrtverlauf.php*) diese neu. In dem Fall, dass zwischen zwei benachbarten *$keyPoints* die zulässige Höchstgeschwindigkeit überschritten wird, wird die absolute Start- und End-Position dieser Geschwindigkeitsüberschreitung gespeichert.

In dem folgenden Schritt wird wie in dem Abschnitt 4.2 zwischen den Start-Werten des ersten *$keyPoints* und der ersten Geschwindigkeitsüberschreitung die maximale Geschwindigkeit berechnet und zwei neue *$keyPoints* erzeugt. Das gleiche passiert zwischen der Position der letzten Geschwindigkeitsüberschreitung und den End-Werten des zweiten *$keyPoints*. Dadurch wird sichergestellt, dass immer eine gerade Anzahl an *$keyPoints* existiert und somit in jedem Iterationsschritt zwei benachbarte *$keyPoints* verglichen werden können. Nachdem alle *$keyPoint*-Paare überprüft wurden, werden mit Hilfe der *createTrainChanges*()-Funktion (*functions_fahrtverlauf.php*) die Arrays *$trainPositionChange* und *$trainSpeedChange* erzeugt. Der neu berechnete Fahrtverlauf wird erneut der Funktion *checkIfTrainIsToFastInCertainSections*() (*functions_fahrtverlauf.php*) (Code-Beispiel 6) übergeben. Dieser Prozess wird solange durchlaufen, bis es zu keiner Geschwindigkeitsüberschreitung mehr kommt. In den folgenden Abbildungen (Darstellung 9, 10 und 11) werden die Ergebnisse der einzelnen Iterationsschritte

```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 6: *checkIfTrainIsToFastInCertainSections*() (*functions_fahrtverlauf-.php*)

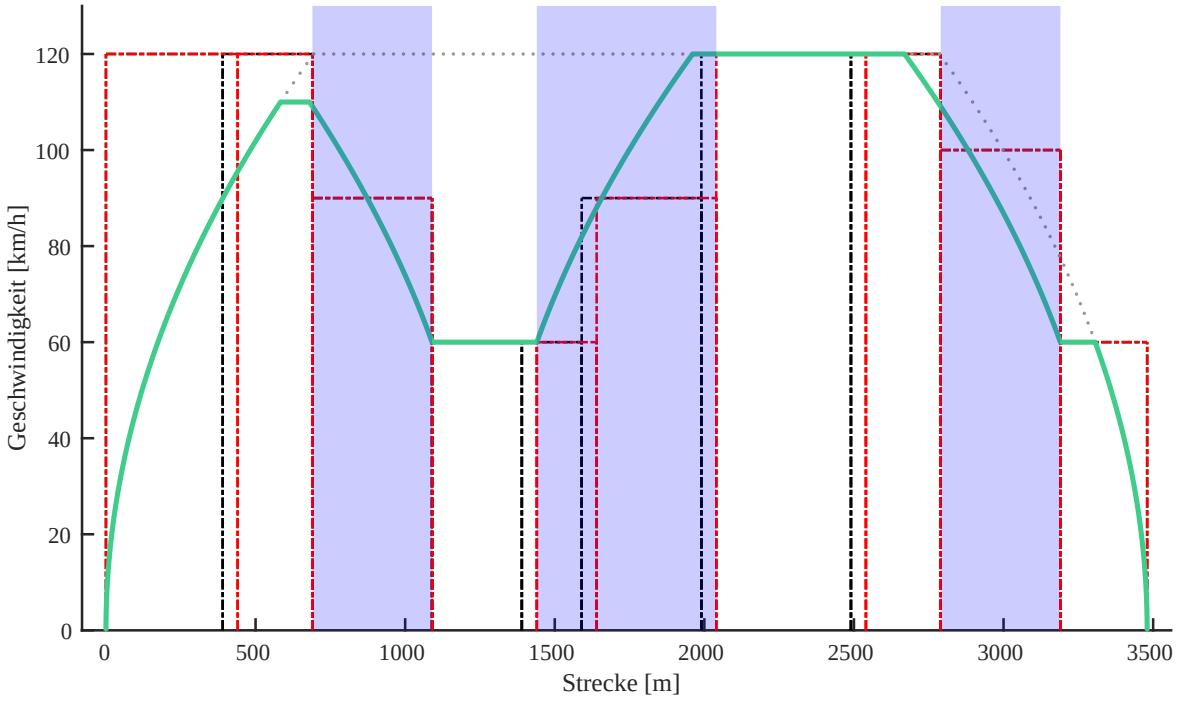

Abbildung 9: Fahrtverlaufsberechnung (2. Iterationsschritt)

visuell abgebildet, wobei die grau gepunkteten Linien die Ergebnisse der vorherigen Iterationsschritte darstellen.

4.5 Einhaltung der Mindestzeit auf einer Beharrungsfahrt

Für eine möglichst realitätsnahe Simulation kann über die Variable *$globalTimeOnOneSpeed* in der Datei *global_variables.php* eine Mindestzeit festgelegt werden, die ein Fahrzeug auf einer Geschwindigkeit mindestens einhalten muss (Beharrungsfahrt). Ebenfalls kann über die Variablen *$useMinTimeOnSpeed* und *$errorMinTimeOnSpeed* festgelegt werden, ob die Funktion aktiviert sein soll und ob es in dem Fall, dass diese Zeit nicht eingehalten werden kann, zu einer Fehlermeldung kommen soll. Im Falle einer Fehlermeldung würde das Fahrzeug nicht losfahren bzw. eine Gefahrenbremsung 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 werden, eine Verzögerung vorzeitiger eingeleitet werden oder auf eine kleinere Geschwindigkeit beschleunigt werden. Dadurch, dass sich eine Verschiebung einer Beschleunigung bzw. Verzögerung auf die nächsten Abschnitte auswirken kann, wird der Fahrtverlauf in *$subsections* unterteilt. Eine *$subsection* beschreibt dabei den Bereich des Fahrtverlaufs, in dem das Fahrzeug zum ersten Mal beschleunigt und zum letzten Mal abbremst. In der Darstellung 12 wurde der exemplari-

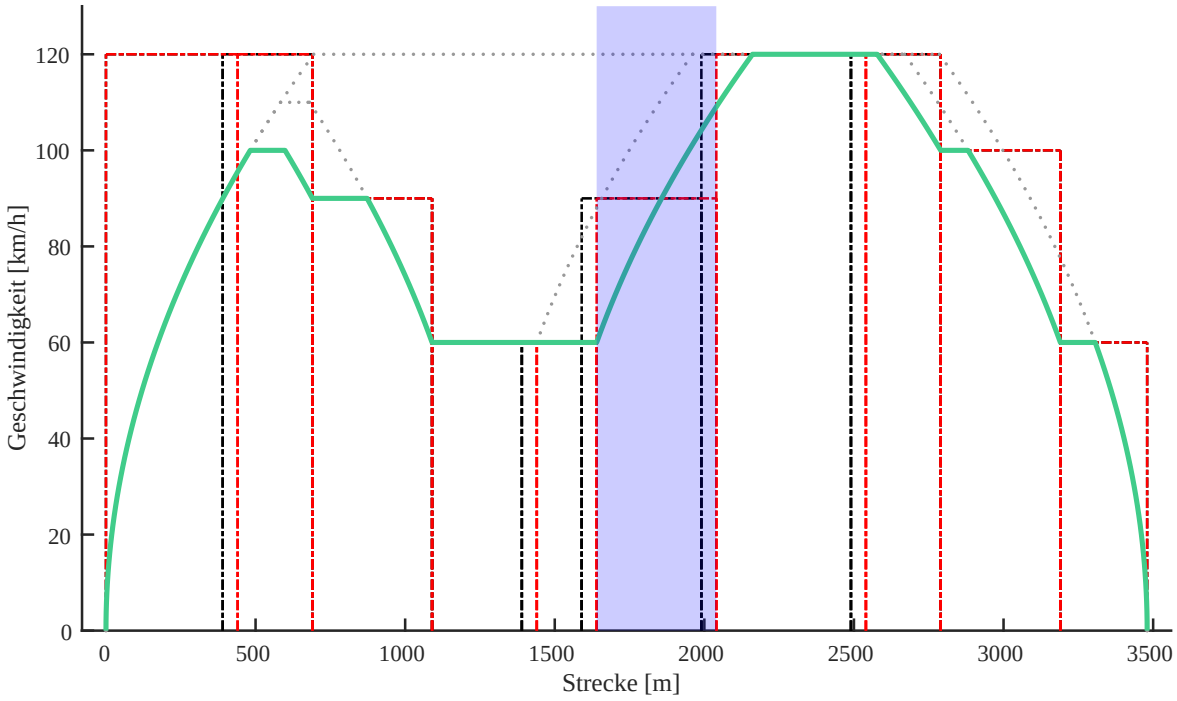

Abbildung 10: Fahrtverlaufsberechnung (3. Iterationsschritt)

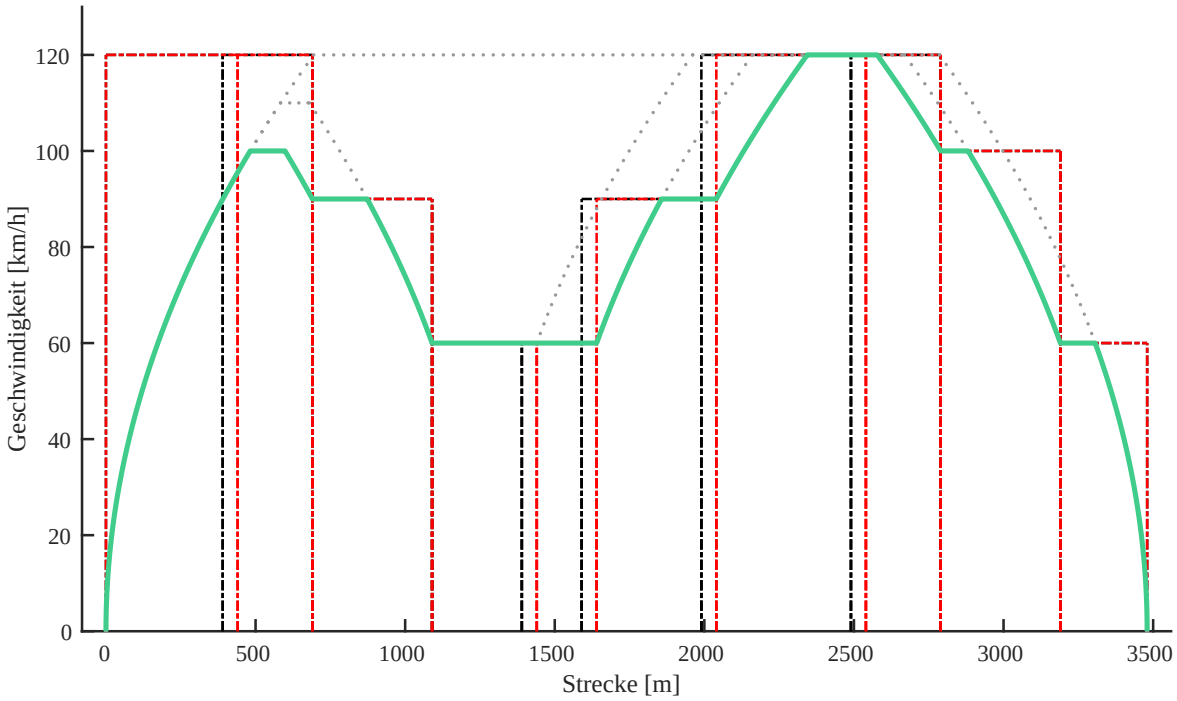

Abbildung 11: Fahrtverlaufsberechnung (4. Iterationsschritt)

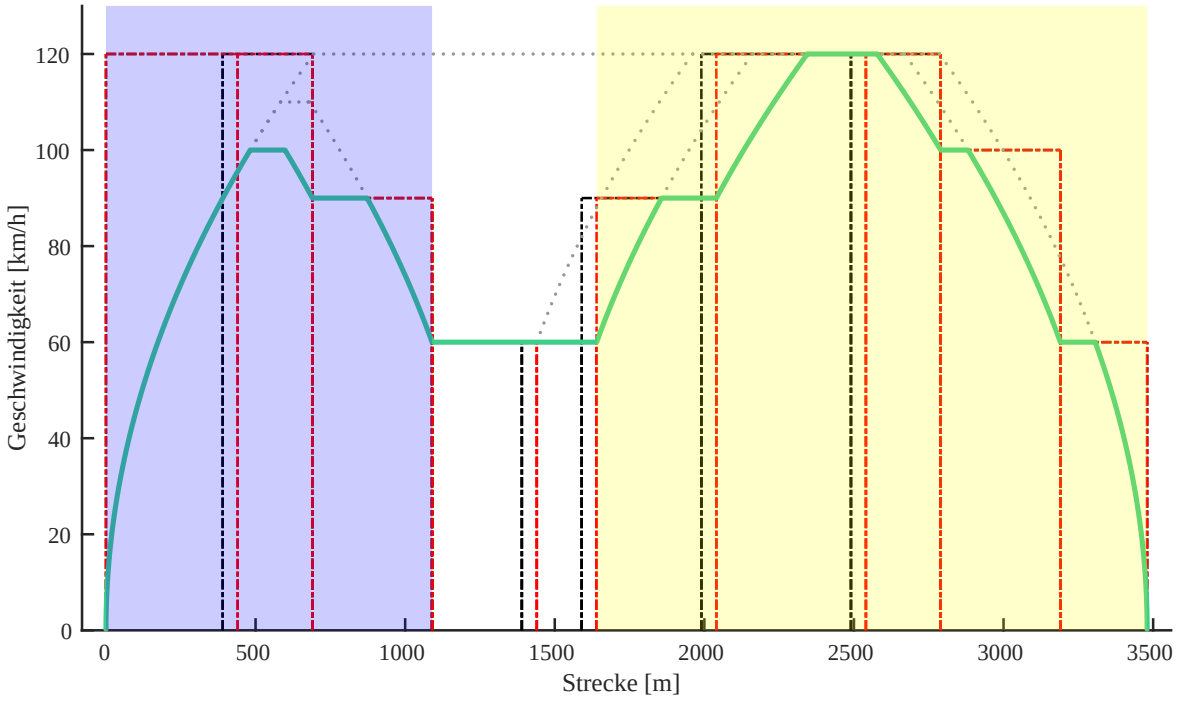

Abbildung 12: Einteilung des Fahrtverlaufs in *$subsections*

sche Fahrtverlauf somit in zwei *$subsection* unterteilt, welche Lila bzw. Gelb hinterlegt sind. Durch diese Einteilung kann verhindert werden, dass es zu Konflikten kommt. Falls die Beschleunigungen bzw. Verzögerungen soweit nach hinten bzw. nach vorne verschoben werden müssen, kann die maximale Geschwindigkeit auf dieser *$subsection* reduziert werden und die zur Verfügung stehende Strecke vergrößert werden. Wie in Darstellung 12 zu erkennen wird hierbei im ersten Schritt der Abschnitt zwischen zwei *$subsections* ausgelassen. Nach der Ermittlung der *$subsections* wird überprüft, ob auf den Abschnitten zwischen den *$subsections* die Mindestzeit eingehalten wird. Wenn das nicht der Fall ist, wird der Abschnitt automatisch der in Fahrtrichtung hinteren *$subsection* zugeordnet. Dadurch wird sichergestellt, dass das Fahrzeug, wenn es an einer Stelle des Fahrtverlaufs die Geschwindigkeit reduziert, dies möglichst spät tut.

Nachdem die *$subsections* mittels der Funktion *createSubsections*() (*func_tions_fahrtverlauf.php*) erstellt wurden und mit der Funktion *array_reverse*() in umgekehrter Reihenfolge in dem Array *$subsection_list* gesammelt wurden, wurde für jede *$subsection* ein Array erzeugt, welches die Variablen aus Tabelle 10 beinhaltet und jede *$subsection* eindeutig beschreibt.

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

Index	Funktion
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

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

Sollte bei einer Verschiebung die *position_1* eines *$keyPoints* hinter *position_0* des folgenden *$keyPoints* liegen (bei einer Beschleunigung), wird der zweite *$keyPoint* gelöscht und die Zielgeschwindigkeit des zweiten *$keyPoints* wird der Zielgeschwindigkeit des ersten *$keyPoints* zugewiesen. Gleiches geschieht bei der Verzögerung in umgekehrter Reihenfolge.

Nach der Verschiebung wird überprüft, ob auf allen konstanten Geschwindigkeit 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 einer Geschwindigkeit 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.

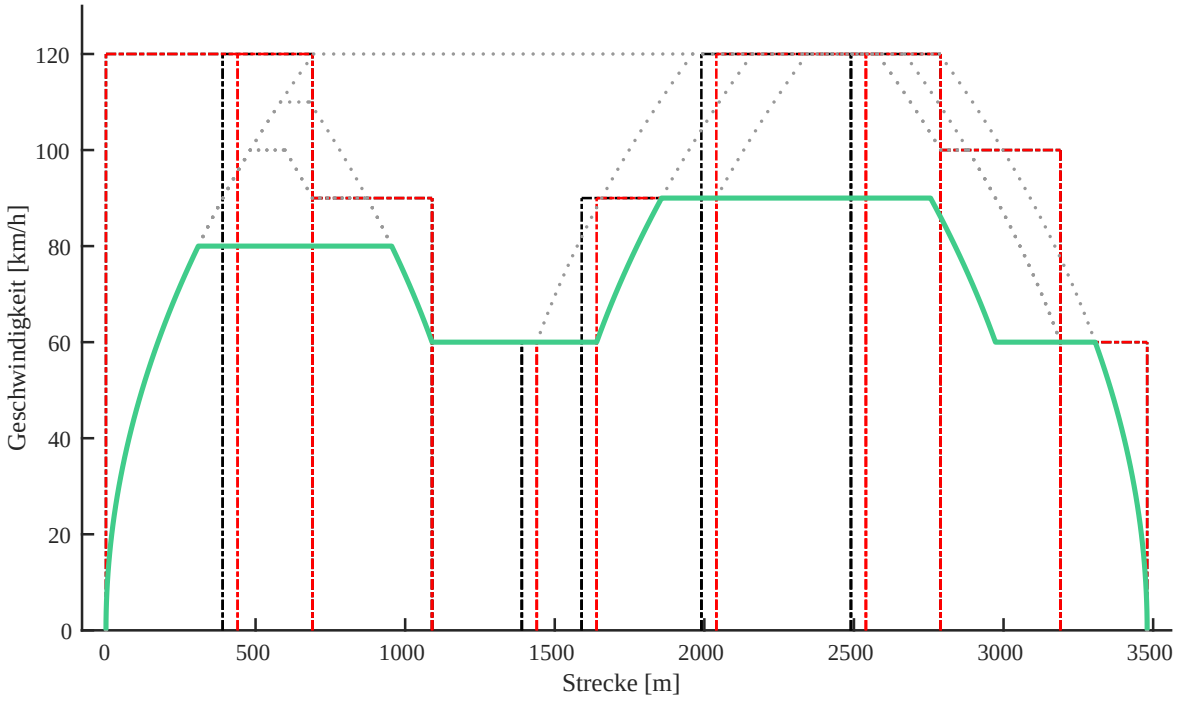

Abbildung 13: Fahrtverlauf unter Einhaltung der Mindestzeit

4.6 Berücksichtigung der Ankunftszeit bei der Berechnung des Fahrtverlaufs

Der berechnete Fahrtverlauf aus den Kapiteln 4.2, 4.3, 4.4 und 4.5 ermittelt die frühstmögliche Ankunftszeit am Ziel. In dem Fall, dass der Zug dadurch mit einer Verspätung am Ziel ankommt, wird der Fahrtverlauf an das Fahrzeug übergeben. Falls das Fahrzeug mit dem Fahrtverlauf zu früh am Ziel ankommen würde, wird überprüft, ob es möglich ist die Geschwindigkeit zu reduzieren, sodass der Zug energieeffizienter fahren kann und ohne Verspätung am Ziel ankommt.

Im ersten Schritt wird mittels der Funktion *checkIfTheSpeedCanBeDecreased*() (*functions_fahrtverlauf.php*) überprüft, ob die Geschwindigkeit reduziert werden kann. Dabei werden alle *$keyPoints* ermittelt, bei denen das Fahrzeug beschleunigt und die beim darauffolgenden *$keyPoint* abbremsen. Für jeden dieser *$keyPoints* werden die möglichen Geschwindigkeiten ermittelt, welche das Fahrzeug zwischen den beiden *$keyPoints* fahren könnte. Für die Berechnung dieser Geschwindigkeiten wird als niedrigste Geschwindigkeit die *speed_0* des ersten *$keyPoints* bzw. *speed_1* des zweiten *$keyPoints* – jenachdem, 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 *$return-*

SpeedDecrease abgespeichert. Der Eintrag *possible* gibt an, ob das Fahrzeug auf dem gesamten Fahrtverlauf die Geschwindigkeit reduzieren könnte und wird als Boolescher Wert (*true/false*) abgespeichert und in dem Array *range* werden alle Indexe der möglichen *$keyPoints* inklusive der ermittelten Geschwindigkeiten abgespeichert.

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

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

4.7 Berücksichtigung der exakten Ankunftszeit bei der Berechnung des Fahrtverlaufs

Die in Kapitel 4.5 errechnete Ankunftszeit, beschreibt die spätmöglichste Ankunftszeit am Ziel, ohne dass das Fahrzeug mit einer Verspätung am Ziel ankommt, wenn bei einer Beschleunigung auf eine geringere Zielgeschwindigkeit beschleunigt wird. Dadurch wird das Fahrzeug im Normalfall nicht exakt pünktlich das Ziel erreichen. Über die Variable *$useSpeedFineTuning* kann festgelegt werden, ob das Fahrzeug eine exakte Ankunftszeit versuchen soll zu erreichen. Wenn diese Funktion aktiviert ist und der Eintrag *possible* aus dem Array *$returnSpeedDecrease true* ist, wird für den 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

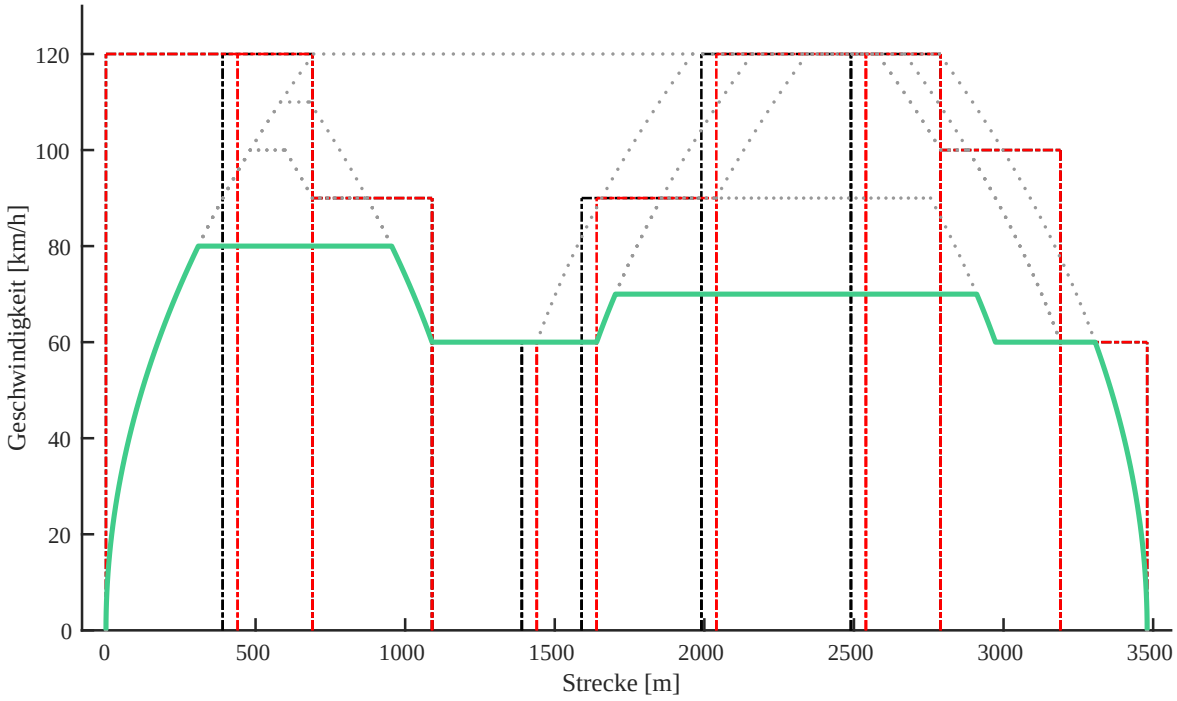

Abbildung 14: Fahrtverlauf mit reduzierter Geschwindigkeit unter Einhaltung der Ankunftszeit

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 *$keyPoints* übergeben bekommt und auf der Gleichung 19 aus Kapitel 7 basiert. In Abbildung 15 werden die Geschwindigkeiten (v_1, v_2), Strecken (s_1, s_2) und Zeiten (t_1, t_2) vor und nach der Verzögerung – welche vorzeitiger eingeleitet werden soll, um eine pünktliche Ankunft am Ziel zu ermöglichen – dargestellt und in Tabelle 11 sind die exakten Werte des exemplarischen Fahrtverlaufs aufgelistet. In diesem konkreten Beispiel würde das Fahrzeug 3,31 s (t_Δ) zu früh an der Betriebsstelle ankommen, wodurch das Fahrzeug für die Zurücklegung der Strecken s_1 und s_2 insgesamt 85,42 s ($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.

$$t_3 = \frac{1541\ m - 16,67\ m/s \cdot 85,42\ s}{19,44\ m/s - 16,67\ m/s}$$
$$t_3 = 42,26\ s$$

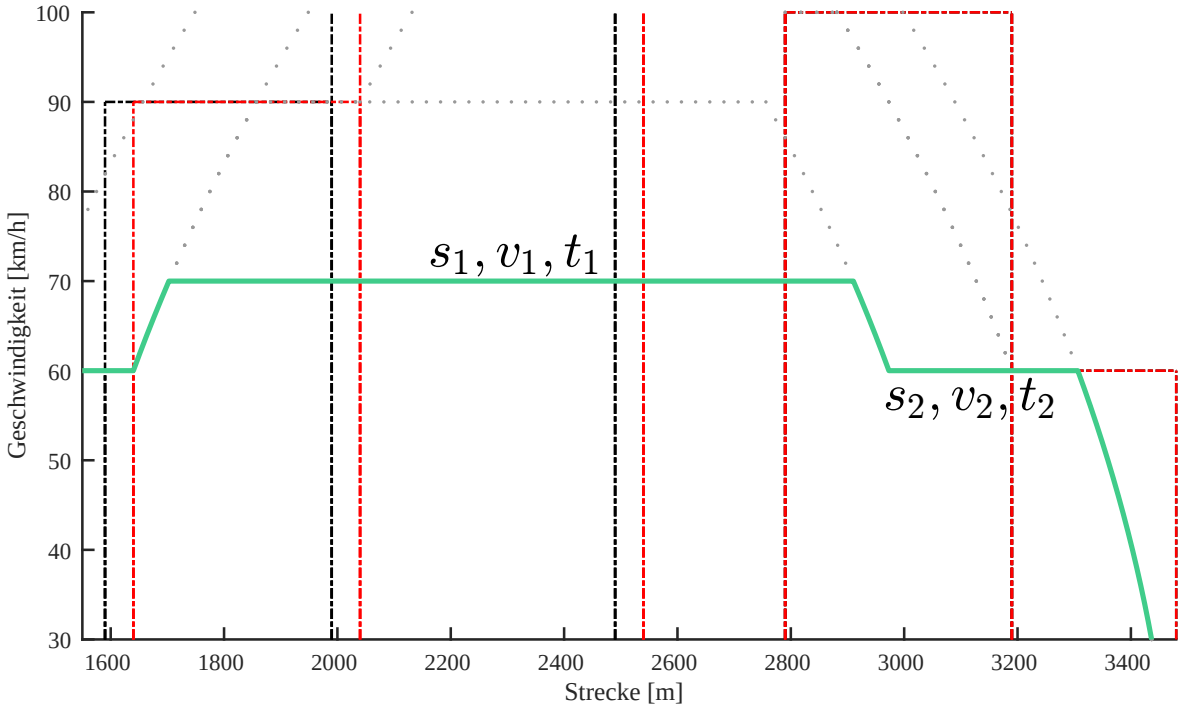

Abbildung 15: Fahrtverlauf vor der Anpassung der exakten Ankunftszeit

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

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

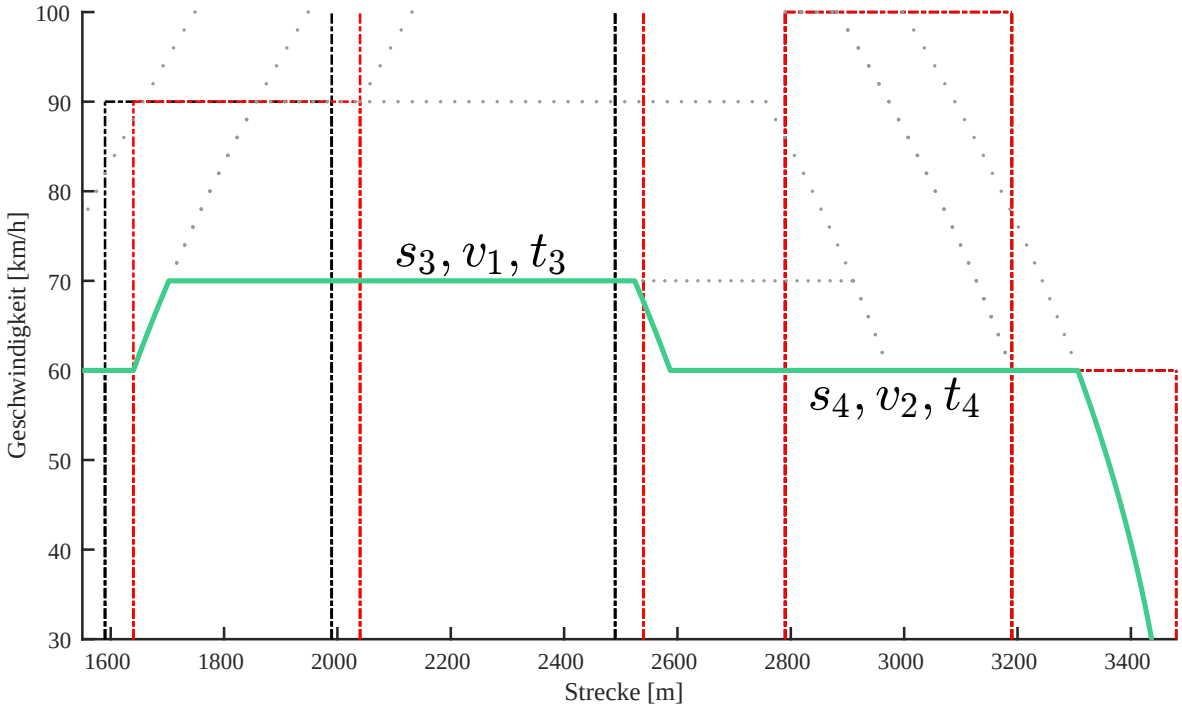

Abbildung 16: Fahrtverlauf nach der Anpassung der exakten Ankunftszeit

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

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

4.8 Einleitung einer Gefahrenbremsung

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

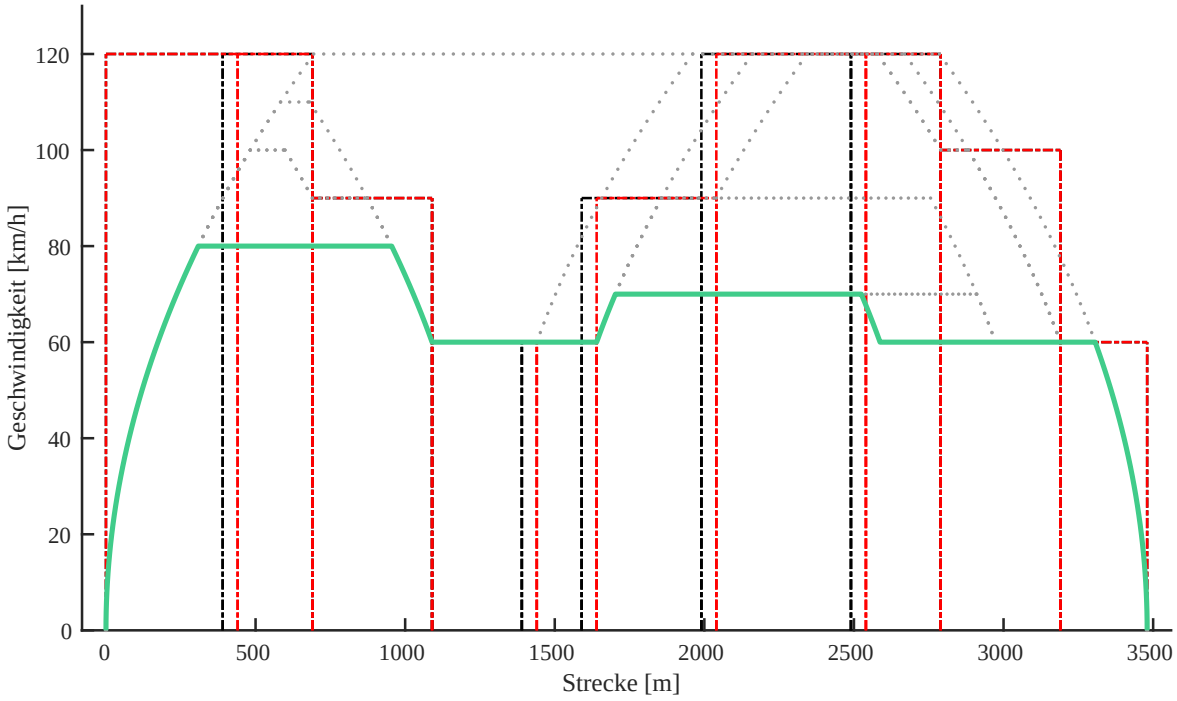

Abbildung 17: Ergebnis der Fahrtverlaufs-Ermittlung

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 Gefahrenstelle 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 Gefahrenstel-

le 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 Fahrtbetrieb aufnehmen kann, muss das Fahrzeug händisch von der Anlage genommen werden, gewartet werden, bis die Fahrzeugsteuerung das Entfernen registriert hat und wieder neu positioniert werden.

$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, hat die Fahrtrichtung 1 und die Fahrstraße ist so eingestellt, dass das Fahrzeug bis zum Ausfahrsignal im Bahnhof Zoo fahren kann und dort im Infra-Abschnitt 1178 zum Stehen kommen kann. Somit beträgt die Strecke bis zum nächsten Halt 672 m und das Fahrzeug hat 115 s zur Verfügung. Die Bremsverzögerung des Fahrzeugs beträgt 0,8 m/s^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-

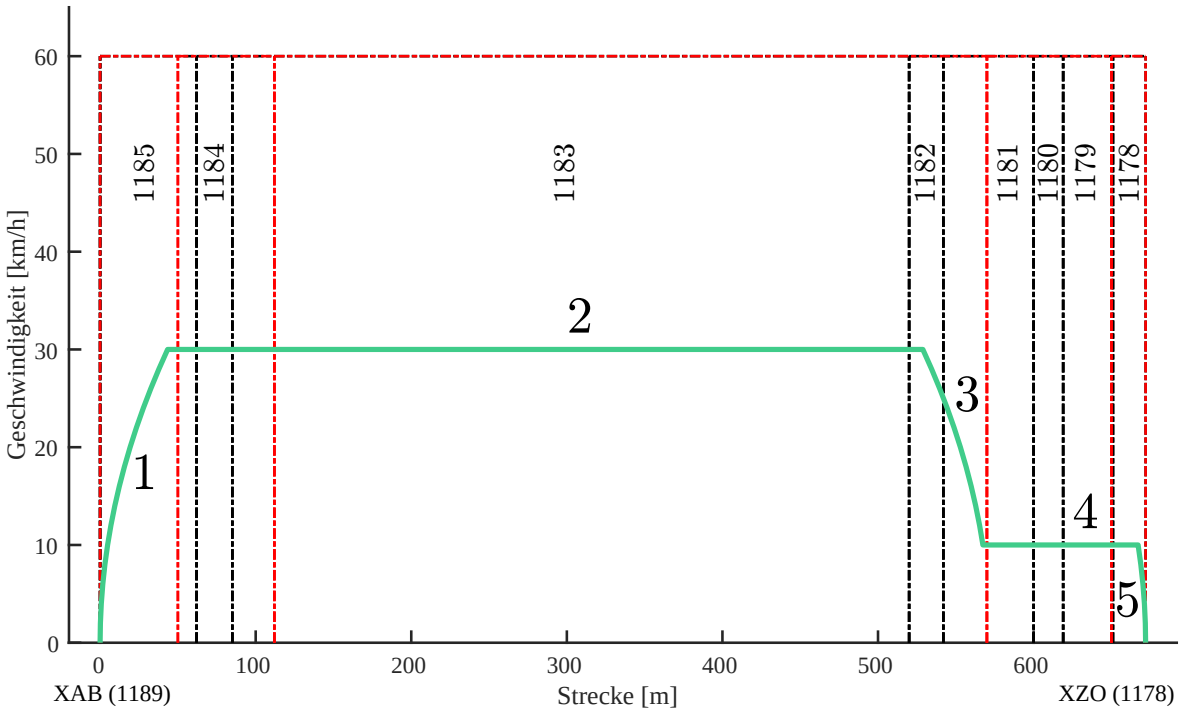

Abbildung 18: Fahrtverlauf für eine Beispielrechnung

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

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

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

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 bis 5 eine Gesamtstrecke von 671,99 m kommen. Die angegebenen 672 m entsprechen der Summe der Abschnitte 1 bis 5, ohne dass die einzelnen Strecken gerundet werden.

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

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

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

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

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

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

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

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

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 einer konstanten Geschwindigkeit (t_2 und t_4) eingehalten und die Werte stimmen mit den Werten aus der Tabelle 14 überein.

6 Visualisierung der Fahrtverläufe

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

Damit die Daten aus der Berechnung des Fahrtverlaufs von MATLAB eingelesen werden können, wurde die Funktion *safeTrainChangeToJSONFile*() (*functions_fahrtverlauf.php*) (Code-Beispiel 7) 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 ist im Anhang (siehe A.8) dieser Arbeit angehängt 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 7: *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 allgemein gültig sind. Für die Berechnung des Beschleuniguns- und Abbremsverhalten der Fahrzeuge ist es notwendig zu wissen,

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

welche Strecke ein Fahrzeug zurücklegen muss, um von einer Startgeschwindigkeit v_0 auf eine Zielgeschwindigkeit v_1 zu beschleunigen bzw. 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 allgemein gültige Gleichung aufzustellen, wird für die Ermittlung der benötigten Strecke bei einer gegebenen Start- und Zielgeschwindigkeit die Strecke berechnet, die das Fahrzeug benötigt um von 0 m/s auf v_1 zu beschleunigen und von 0 m/s auf v_0. 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 dem Programm übernimmt diese Berechnung die Funktion *getBrakeDistance()* (Code-Beispiel 8).

```
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 8: *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 dem Programm übernimmt diese Berechnung die Funktion *getBrakeTime()* (*functions_math.php*) (Code-Beispiel 9).

```
// 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 9: *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.

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

```
// 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 10: *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 11: *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 12: *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 frühstmöglich 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 **??** erläutert werden.

8.2 Was hat nicht geklappt

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.

- Test

8.2.2 Ermittlung der Fahrstraßen

- *getNaechsteAbschnitte*() hat bei bestimmten Betriebsstellen nicht die eingestellte Fahrstraße wiedergegebene

8.2.3 Kalibrierung der Position

- nicht erklärbar, entweder wird der Eintrag in *fahrzeuge_abschnitte* nicht pünktlich eingetragen (falsche Zeit), oder die hinterlegten Längen stimmen nicht

8.3 Potential für die Zukunft

- Moving-Block Verfahren

A Anhang

A.1 fahrzeugsteuerung.php

```php
<?php

// Liest alle benötigten Dateien ein
require 'config/multicast.php';
require '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 Notbremsung 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);
$cacheAdresseToID = array();  // Filled with data in getAllTrains()
$cacheIDToAdresse = array();  // Filled with data in getAllTrains()

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

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

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

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

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

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

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

A.2 functions.php

```php
<?php
ini_set('memory_limit', '1024M');

// 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\t#\n";
  echo "#\t\t\t\t\t\t\t @-@-@-oo\\\t\t\t\t\t\t\t\t\t#\n";
```

```
33    echo "#\t\t\t\t\t==========================\t\t\t\t\t#\n";
34    echo $emptyLine;
35    echo "#\t Start der Simulation: \t\t\t\t\t\t\t\t\t\t\t#\n";
36    echo "#\t\t Simulationszeit: \t\t\t", $simulationStartTimeAsHHMMSS, "\t\t\t\t
          ↪ \t\t#\n";
37    echo "#\t\t Realzeit: \t\t\t\t\t", $realStartTimeAsHHMMSS, "\t\t\t\t\t\t#\n";
38    echo $emptyLine;
39    echo "#\t Ende der Simulation: \t\t\t\t\t\t\t\t\t\t\t#\n";
40    echo "#\t\t Simulationszeit: \t\t\t", $simulationEndTimeAsHHMMSS, "\t\t\t\t\t
          ↪ \t#\n";
41    echo "#\t\t Realzeit: \t\t\t\t\t", $realEndTimeAsHHMMSS, "\t\t\t\t\t\t#\n";
42    echo $emptyLine;
43    echo "#\t Dauer der Simulation: \t\t\t", $simulationDurationAsHHMMSS, "\t\t\t
          ↪ \t\t\t#\n";
44    echo $emptyLine;
45    echo "#\t Fahrplanname: \t\t\t\t\t", $cacheFahrplanSession->name, "\t\t#\n";
46    echo "#\t Sessionkey: \t\t\t\t\t", $cacheFahrplanSession->sessionkey, "\t\t\t
          ↪ \t\t#\n";
47    echo $emptyLine;
48    echo $hashtagLine, "\n\n";
49  }
50
51  // Konvertiert Sekunden in das Format hh:mm:ss
52  function toStd(float $sekunden) {
53
54    $stunden = floor($sekunden / 3600);
55    $minuten = floor(($sekunden - ($stunden * 3600)) / 60);
56    $sekunden = round($sekunden - ($stunden * 3600) - ($minuten * 60));
57
58    if ($stunden <= 9) {
59      $strStunden = "0" . $stunden;
60    } else {
61      $strStunden = $stunden;
62    }
63
64    if ($minuten <= 9) {
65      $strMinuten = "0" . $minuten;
66    } else {
67      $strMinuten = $minuten;
```

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

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

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

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

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

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

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

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

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

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

```
// so wird dem Fahrzeug eine Fehlermeldung (Fehlerstatus = 0) hinzugefügt.
function changeDirection (int $id) {

  global $allUsedTrains;
  global $cacheInfraLaenge;
  global $timeDifference;
  global $allTrains;

  $section = $allUsedTrains[$id]["current_section"];
  $position = $allUsedTrains[$id]["current_position"];
  $direction = $allUsedTrains[$id]["dir"];
  $length = $allUsedTrains[$id]["zuglaenge"];
  $newTrainLength = $length + ($cacheInfraLaenge[$section] - $position);
  $newDirection = null;
  $newSection = null;
  $cumLength = 0;

  if ($direction == 0) {
    $newDirection = 1;
  } else {
    $newDirection = 0;
  }

  $newPosition = null;
  $nextSections = getNaechsteAbschnitte($section, $newDirection);
  $currentData = array(0 => array("laenge" => $cacheInfraLaenge[$section], "
      ↪ infra_id" => $section));
  $mergedData = array_merge($currentData, $nextSections);

  foreach ($mergedData as $sectionValue) {
    $cumLength += $sectionValue["laenge"];

    if ($newTrainLength <= $cumLength) {
      $newSection = $sectionValue["infra_id"];
      $newPosition = $cacheInfraLaenge[$newSection] - ($cumLength -
          ↪ $newTrainLength);
      break;
    }
  }
```

```
439
440   if ($newPosition == null) {
441     echo "Die Richtung des Zugs mit der ID ", $id, " lässt sich nicht ändern,
          ↪ weil das Zugende auf einem auf Halt stehenden Signal steht.\n";
442     echo "\tDie Zuglänge beträgt:\t", $length, " m\n\tDie Distanz zwischen
          ↪ Zugende und dem auf Halt stehenden Signal beträgt:\t", ($cumLength -
          ↪  ($cacheInfraLaenge[$section] - $position)), " m\n\n";
443     array_push($allUsedTrains[$id]["error"], 0);
444   } else {
445     echo "Die Richtung des Zugs mit der ID: ", $id, " wurde auf ",
          ↪ $newDirection, " geändert.\n";
446     $allUsedTrains[$id]["current_section"] = $newSection;
447     $allUsedTrains[$id]["current_position"] = $newPosition;
448     $allUsedTrains[$id]["dir"] = $newDirection;
449     $allUsedTrains[$id]["earliest_possible_start_time"] = FZS_WARTEZEIT_WENDEN
          ↪ + time() + $timeDifference;
450     $allTrains[$id]["dir"] = $newDirection;
451     $DB = new DB_MySQL();
452     $DB->select("UPDATE `".DB_TABLE_FAHRZEUGE."` SET `".DB_TABLE_FAHRZEUGE."`.`
          ↪ dir` = $newDirection WHERE `".DB_TABLE_FAHRZEUGE."`.`id` = $id");
453     unset($DB);
454     sendFahrzeugbefehl($id, -4);
455   }
456 }
457
458 // Gibt für alle Fahrzeuge die vorhanden Fehlermeldungen an.
459 function showErrors() {
460
461   global $allUsedTrains;
462   global $trainErrors;
463
464   $foundError = false;
465   echo "Hier werden für alle Züge mögliche Fehler angezeigt:\n\n";
466
467   foreach ($allUsedTrains as $trainIndex => $trainValue) {
468     if (sizeof($trainValue["error"]) != 0) {
469       $foundError = true;
470       echo "Zug ID: ", $trainValue["id"], "\n";
471       $index = 1;
```

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

```
507             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                    ↪ $betriebsstelleKey]["haltepunkte"] = $cacheHaltepunkte[
                    ↪ $betriebsstelleValue["betriebstelle"]][$trainValue["dir"]];
508           } else if (in_array($betriebsstelleValue["betriebstelle"],
                  ↪ array_keys($cacheZwischenhaltepunkte))) {
509             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                    ↪ $betriebsstelleKey]["haltepunkte"] = array(
                    ↪ $cacheZwischenhaltepunkte[$betriebsstelleValue["
                    ↪ betriebstelle"]]);
510           } else {
511             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                    ↪ $betriebsstelleKey]["haltepunkte"] = array();
512           }
513         }
514       }
515     }
516   }
517 }
518
519 // TODO: KANN GELÖSCHT WERDEN?!?!
520 function initalFirstLiveData($id = false) {
521
522   global $allUsedTrains;
523   global $allTimes;
524
525   $checkAllTrains = true;
526
527   if ($id != false) {
528     $checkAllTrains = false;
529   }
530
531   foreach ($allUsedTrains as $trainIndex => $trainValue) {
532     if (($checkAllTrains || $trainValue["id"] == $id)) {
533       $allTimes[$trainValue["adresse"]] = array();
534     }
535   }
536 }
537
538
```

```php
// Ermittelt für alle Fahrzeuge (wenn keine ID übergeben wird) oder für ein
// Fahrzeug (wenn eine ID übergeben wird) die Fahrstraße inkl. der Längen,
// der zulässigen Höchstgeschwindigkeiten und der IDs der nächsten Abschnitte.
//
// Die Ergebnisse können direkt im Array $usedTrains gespeichert werden
// ($writeResultToTrain = true) oder als return zurückgegeben werden
// ($writeResultToTrain = false), so dass sie verglichen werden können
// mit den vorherigen Daten verglichen werden können.
function calculateNextSections($id = false, $writeResultToTrain = true) {

  global $allUsedTrains;
  global $cacheInfraLaenge;
  global $globalSpeedInCurrentSection;
  global $lastMaxSpeedForInfraAndDir;

  $checkAllTrains = true;

  if ($id != false) {
    $checkAllTrains = false;
  }

  foreach ($allUsedTrains as $trainIndex => $trainValue) {
    if (($checkAllTrains || $trainValue["id"] == $id) && sizeof($trainValue["
        ↪ error"]) == 0) {
      $dir = $trainValue["dir"];
      $currentSectionComp = $trainValue["current_section"];
      $signal = getSignalForSectionAndDirection($currentSectionComp, $dir);
      $nextSectionsComp = array();
      $nextVMaxComp = array();
      $nextLengthsComp = array();
      $nextSignalbegriff = null;

      if ($signal != null) {
        $nextSignalbegriff = getSignalbegriff($signal);
        $nextSignalbegriff = $nextSignalbegriff[array_key_last(
            ↪ $nextSignalbegriff)]["geschwindigkeit"];
        if ($nextSignalbegriff == -25) {
          $nextSignalbegriff = 25;
        } else if ($nextSignalbegriff <= 0) {
```

```
576           $nextSignalbegriff = 0;
577         }
578       } else {
579         $nextSignalbegriff = null;
580       }
581
582       $return = getNaechsteAbschnitte($currentSectionComp, $dir);
583       $allUsedTrains[$trainIndex]["last_get_naechste_abschnitte"] = $return;
584
585       if (isset($lastMaxSpeedForInfraAndDir[$trainValue["dir"]][$trainValue["
              ↪ current_section"]])) {
586         $currentVMax = $lastMaxSpeedForInfraAndDir[$trainValue["dir"]][
                ↪ $trainValue["current_section"]];
587       } else {
588         $currentVMax = $globalSpeedInCurrentSection;
589       }
590
591       array_push($nextSectionsComp, $currentSectionComp);
592       array_push($nextVMaxComp, $currentVMax);
593       array_push($nextLengthsComp, $cacheInfraLaenge[$currentSectionComp]);
594
595       if (isset($nextSignalbegriff)) {
596         $currentVMax = $nextSignalbegriff;
597       }
598
599       if ($currentVMax == 0) {
600         if ($writeResultToTrain) {
601           $allUsedTrains[$trainIndex]["next_sections"] = $nextSectionsComp;
602           $allUsedTrains[$trainIndex]["next_lenghts"] = $nextLengthsComp;
603           $allUsedTrains[$trainIndex]["next_v_max"] = $nextVMaxComp;
604         } else {
605           return array($nextSectionsComp, $nextLengthsComp, $nextVMaxComp);
606         }
607       } else {
608         foreach ($return as $section) {
609           array_push($nextSectionsComp, $section["infra_id"]);
610           array_push($nextVMaxComp, $currentVMax);
611           array_push($nextLengthsComp, $cacheInfraLaenge[$section["infra_id"]]);
```

```
612           $lastMaxSpeedForInfraAndDir[intval($trainValue["dir"])][intval(
                  ↪ $section["infra_id"])] = intval($currentVMax);
613           if ($section["signal_id"] != null) {
614             $signal = $section["signal_id"];
615             $nextSignalbegriff = getSignalbegriff($signal);
616             $nextSignalbegriff = $nextSignalbegriff[array_key_last(
                    ↪ $nextSignalbegriff)]["geschwindigkeit"];
617             if ($nextSignalbegriff == -25) {
618               $currentVMax = 25;
619             } else if ($nextSignalbegriff < 0) {
620               $currentVMax = 0;
621             } else {
622               $currentVMax = $nextSignalbegriff;
623             }
624           }
625         }
626         if ($writeResultToTrain) {
627           $allUsedTrains[$trainIndex]["next_sections"] = $nextSectionsComp;
628           $allUsedTrains[$trainIndex]["next_lenghts"] = $nextLengthsComp;
629           $allUsedTrains[$trainIndex]["next_v_max"] = $nextVMaxComp;
630         } else {
631           return array($nextSectionsComp, $nextLengthsComp, $nextVMaxComp);
632         }
633       }
634     }
635   }
636 }
637
638 // Prüft für alle Fahrzeuge (falls keine ID übergeben wird) oder für ein
       ↪ Fahrzeug
639 // (falls eine ID übergeben wird), ob das Fahrzeug bereits am ersten fahrplanmä
       ↪ ßigen
640 // Halt ist oder nicht.
641 function checkIfTrainReachedHaltepunkt ($id = false) {
642
643   global $allUsedTrains;
644   global $cacheInfraToGbt;
645   global $cacheGbtToInfra;
646
```

```
647   $checkAllTrains = true;
648
649   if ($id != false) {
650     $checkAllTrains = false;
651   }
652
653   foreach ($allUsedTrains as $trainIndex => $trainValue) {
654     if ($checkAllTrains || $trainValue["id"] == $id) {
655       $currentInfrasection = $trainValue["current_section"];
656       $currentGbt = $cacheInfraToGbt[$currentInfrasection];
657       $allInfraSections = $cacheGbtToInfra[$currentGbt];
658       for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i++)
          ↪  {
659         if (sizeof(array_intersect($trainValue["next_betriebsstellen_data"][$i][
            ↪ "haltepunkte"], $allInfraSections)) != 0) {
660           $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$i]["
              ↪ angekommen"] = true;
661           for ($j = 0; $j < $i; $j++) {
662             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$j]["
                ↪ angekommen"] = true;
663           }
664         } else {
665           $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$i]["
              ↪ angekommen"] = false;
666         }
667       }
668     }
669   }
670 }
671
672 // Prüft für alle Fahrzeuge (falls keine ID übergeben wird) oder für ein
    ↪ Fahrzeug
673 // (falls eine ID übergeben wird), ob die Fahrstraße aktuell richtig
    ↪ eingestellt
674 // ist, sodass die nächste Betriebsstelle laut Fahrplan erreicht werden kann.
675 //
676 // Für Züge ohne Fahrplan ist der Fahrweg immer korrekt.
677 function checkIfFahrstrasseIsCorrrect($id = false) {
678
```

```
679   global $allUsedTrains;
680
681   $checkAllTrains = true;
682
683   if ($id != false) {
684     $checkAllTrains = false;
685   }
686
687   foreach ($allUsedTrains as $trainIndex => $trainValue) {
688     if (($checkAllTrains || $trainValue["id"] == $id) && sizeof($trainValue["
          ↪ error"]) == 0) {
689       if ($trainValue["operates_on_timetable"]) {
690         $allUsedTrains[$trainIndex]["fahrstrasse_is_correct"] = false;
691         foreach ($trainValue["next_betriebsstellen_data"] as $stopIndex =>
              ↪ $stopValue) {
692           if (!$stopValue["angekommen"]) {
693             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$stopIndex
                  ↪ ]["is_on_fahrstrasse"] = false;
694             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$stopIndex
                  ↪ ]["used_haltepunkt"] = array();
695             $indexSection = 0;
696             for ($i = 0; $i < sizeof($trainValue["next_sections"]); $i++) {
697               if ($stopValue["haltepunkte"] != null) {
698                 if (in_array($trainValue["next_sections"][$i], $stopValue["
                      ↪ haltepunkte"])) {
699                   if ($i >= $indexSection) {
700                     $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                          ↪ $stopIndex]["is_on_fahrstrasse"] = true;
701                     $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                          ↪ $stopIndex]["used_haltepunkt"] = $trainValue["
                          ↪ next_sections"][$i];
702                     $allUsedTrains[$trainIndex]["fahrstrasse_is_correct"] = true;
703                     $i = sizeof($trainValue["next_sections"]);
704                     $indexSection = $i;
705                   }
706                 }
707               }
708             }
709           } else {
```

```
710             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$stopIndex
                    ↪ ]["is_on_fahrstrasse"] = true;
711           }
712         }
713       } else {
714         $allUsedTrains[$trainIndex]["fahrstrasse_is_correct"] = true;
715       }
716     }
717   }
718 }
719
720 // Berechnet die Beschleunigungs- und Bremskurven für alle Züge (wenn keine ID
721 // übergeben wird) oder für einen Zug (wenn eine ID übergeben wird). Für Züge -
722 // die nach Fahrplan fahren - für alle Betriebsstellen, die auf der aktuell
723 // eingestellten Strecke liegen und für Züge ohne Fahrplan bis zum nächsten
724 // roten Signal.
725 function calculateFahrtverlauf($id = false, $recalibrate = false) {
726
727   global $allUsedTrains;
728   global $cacheInfraLaenge;
729   global $timeDifference;
730   global $globalFirstHaltMinTime;
731
732   $checkAllTrains = true;
733
734   if ($id != false) {
735     $checkAllTrains = false;
736   }
737
738   foreach ($allUsedTrains as $trainIndex => $trainValue) {
739     $allPossibleStops = array();
740     for($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i++) {
741       if ($trainValue["next_betriebsstellen_data"][$i]["fahrplanhalt"]) {
742         array_push($allPossibleStops, $i);
743       }
744     }
745     if (sizeof($trainValue["error"]) == 0 && $trainValue["
          ↪ fahrstrasse_is_correct"]) {
746       if ($checkAllTrains || $trainValue["id"] == $id) {
```

```
747         if ($trainValue["operates_on_timetable"]) {
748           $nextBetriebsstelleIndex = null;
749           $allreachedInfras = array();
750           $wendet = false;
751           for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i
                  ↪ ++) {
752             if (!$trainValue["next_betriebsstellen_data"][$i]["angekommen"] &&
                    ↪ $trainValue["next_betriebsstellen_data"][$i]["
                    ↪ is_on_fahrstrasse"] && $trainValue["next_betriebsstellen_data
                    ↪ "][$i]["fahrplanhalt"]) {
753               $nextBetriebsstelleIndex = $i;
754               $allUsedTrains[$trainIndex]["next_bs"] = $i;
755               break;
756             }
757           }
758           if (!isset($nextBetriebsstelleIndex)) {
759             for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]);
                    ↪ $i++) {
760               if (!$trainValue["next_betriebsstellen_data"][$i]["angekommen"] &&
                      ↪  $trainValue["next_betriebsstellen_data"][$i]["
                      ↪ is_on_fahrstrasse"]) {
761                 $nextBetriebsstelleIndex = $i;
762                 break;
763               }
764             }
765           }
766           if (isset($nextBetriebsstelleIndex)) {
767             if ($allUsedTrains[$trainIndex]["next_bs"] != $trainValue["
                    ↪ next_betriebsstellen_data"][$nextBetriebsstelleIndex]["
                    ↪ betriebstelle"] || $recalibrate) {
768               $allUsedTrains[$trainIndex]["next_bs"] = $trainValue["
                      ↪ next_betriebsstellen_data"][$nextBetriebsstelleIndex]["
                      ↪ betriebstelle"];
769               if (intval($trainValue["next_betriebsstellen_data"][
                      ↪ $nextBetriebsstelleIndex]["zeiten"]["wendet"]) == 1) {
770                 $wendet = true;
771               }
772               for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"])
                      ↪ ; $i++) {
```

```
773                 if (!$trainValue["next_betriebsstellen_data"][$i]["angekommen"]
                        ↪ && $trainValue["next_betriebsstellen_data"][$i]["
                        ↪ is_on_fahrstrasse"] && $i <= $nextBetriebsstelleIndex) {
774                   array_push($allreachedInfras, array("index" => $i, "infra" =>
                          ↪ $trainValue["next_betriebsstellen_data"][$i]["
                          ↪ used_haltepunkt"]));
775                 }
776               }
777               $targetSection = $trainValue["next_betriebsstellen_data"][
                      ↪ $nextBetriebsstelleIndex]["used_haltepunkt"];
778               $targetPosition = $cacheInfraLaenge[$targetSection];
779               $startTime = null;
780               $endTime = null;
781               $prevBetriebsstelle = null;
782               for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"])
                      ↪ ; $i++) {
783                 if ($trainValue["next_betriebsstellen_data"][$i]["angekommen"])
                        ↪ {
784                   $prevBetriebsstelle = $i;
785                   break;
786                 }
787               }
788               if ($nextBetriebsstelleIndex == 0) {
789                 $startTime = microtime(true) + $timeDifference;
790                 $endTime = $startTime;
791               } else {
792                 $endTime = $trainValue["next_betriebsstellen_data"][
                        ↪ $nextBetriebsstelleIndex]["zeiten"]["
                        ↪ ankunft_soll_timestamp"];
793                 if (isset($prevBetriebsstelle)) {
794                   if ($trainValue["next_betriebsstellen_data"][
                          ↪ $prevBetriebsstelle]["zeiten"]["verspaetung"] > 0) {
795                     $startTime = $trainValue["next_betriebsstellen_data"][
                            ↪ $prevBetriebsstelle]["zeiten"]["abfahrt_soll_timestamp
                            ↪ "] + $trainValue["next_betriebsstellen_data"][
                            ↪ $nextBetriebsstelleIndex - 1]["zeiten"]["verspaetung"
                            ↪ ];
796                   } else {
```

```
797                 $startTime = $trainValue["next_betriebsstellen_data"][
                        ↪ $prevBetriebsstelle]["zeiten"]["abfahrt_soll_timestamp
                        ↪ "];
798               }
799             } else {
800               $startTime = microtime(true) + $timeDifference;
801             }
802           }
803           $reachedBetriebsstele = true;
804
805           if ($startTime < microtime(true) + $timeDifference) {
806             $startTime = microtime(true) + $timeDifference;
807           }
808
809           if (isset($trainValue["earliest_possible_start_time"])) {
810             if ($startTime < $trainValue["earliest_possible_start_time"]) {
811               $startTime = $trainValue["earliest_possible_start_time"];
812             }
813           }
814
815           $verapetung = updateNextSpeed($trainValue, $startTime, $endTime,
                  ↪ $targetSection, $targetPosition, $reachedBetriebsstele,
                  ↪ $nextBetriebsstelleIndex, $wendet, false, $allreachedInfras
                  ↪ );
816
817           if ($nextBetriebsstelleIndex != 0) {
818             // TODO: Reicht nicht einer der Einträge aus? Wenn ja, welcher?
819             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                    ↪ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] =
                    ↪ $verapetung;
820             $trainValue["next_betriebsstellen_data"][
                    ↪ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] =
                    ↪ $verapetung;
821           } else {
822             $end = $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                    ↪ $nextBetriebsstelleIndex]["zeiten"]["
                    ↪ abfahrt_soll_timestamp"];
823             $start = $startTime;
824             if ($start + $verapetung + $globalFirstHaltMinTime < $end) {
```

```
825                 $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                        ↪ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] = 0;
826                 $trainValue["next_betriebsstellen_data"][
                        ↪ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] = 0;
827               } else {
828                 $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                        ↪ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] =
                        ↪ $start + $verapetung + $globalFirstHaltMinTime - $end;
829                 $trainValue["next_betriebsstellen_data"][
                        ↪ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] =
                        ↪ $start + $verapetung + $globalFirstHaltMinTime - $end;
830               }
831             }
832           }
833         } else {
834           if ($trainValue["current_speed"] > 0) {
835             emergencyBreak($trainValue["id"]);
836           }
837         }
838       } else {
839         $startTime = microtime(true) + $timeDifference;
840         if (isset($trainValue["earliest_possible_start_time"])) {
841           if ($startTime < $trainValue["earliest_possible_start_time"]) {
842             $startTime = $trainValue["earliest_possible_start_time"];
843           }
844         }
845         $endTime = $startTime;
846         $targetSection = null;
847         $targetPosition = null;
848         $reachedBetriebsstele = true;
849         $wendet = false;
850         $signalId = null;
851         for ($i = 0; $i < sizeof($trainValue["last_get_naechste_abschnitte"]);
                ↪  $i++) {
852           if (isset($trainValue["last_get_naechste_abschnitte"][$i]["signal_id
                  ↪ "])) {
853             $signalId = $trainValue["last_get_naechste_abschnitte"][$i]["
                    ↪ signal_id"];
```

```
854             $targetSection = $trainValue["last_get_naechste_abschnitte"][$i]["
                    ↪ infra_id"];
855             $targetPosition = $cacheInfraLaenge[$targetSection];
856           }
857         }
858         if (!isset($signalId)) {
859           // gibt kein nächstes Signal
860           if ($trainValue["current_speed"] != 0) {
861             emergencyBreak($trainValue["id"]);
862           }
863         } else {
864           $signal = getSignalbegriff($signalId)[0]["geschwindigkeit"];
865
866           if ($signal > -25 && $signal < 0) {
867             $wendet = true;
868           }
869
870           updateNextSpeed($trainValue, $startTime, $endTime, $targetSection,
                  ↪ $targetPosition, $reachedBetriebsstele, $signalId, $wendet,
                  ↪ true, array());
871         }
872       }
873     }
874   } else {
875     if ($trainValue["current_speed"] != 0) {
876       emergencyBreak($trainValue["id"]);
877     }
878   }
879 }
880 }
881
882 // Vergleicht für ein Fahrzeug die zuletzt ermittelte Fahrstraße mit der
      ↪ aktuellen
883 // Fahrstraße und berechnet den Fahrtverlauf neu, wenn das nötig ist.
884 function compareTwoNaechsteAbschnitte(int $id) {
885
886   global $allUsedTrains;
887   global $allTimes;
888
```

```
889  if (sizeof($allUsedTrains[$id]["error"]) == 0) {
890    $newSections = calculateNextSections($id, false);
891    $newNextSection = $newSections[0];
892    $newNextLenghts = $newSections[1];
893    $newNextVMax = $newSections[2];
894    $oldNextSections = $allUsedTrains[$id]["next_sections"];
895    $oldLenghts = $allUsedTrains[$id]["next_lenghts"];
896    $oldNextVMax = $allUsedTrains[$id]["next_v_max"];
897    $currentSectionOld = $allUsedTrains[$id]["current_section"];
898    $keyCurrentSection = array_search($currentSectionOld, $oldNextSections);
899    $keyLatestSection = array_key_last($oldNextSections);
900    $dataIsIdentical = true;
901    $numberOfSection = $keyLatestSection - $keyCurrentSection + 1;
902    $compareNextSections = array();
903    $compareNextLenghts = array();
904    $compareNextVMax = array();
905
906    for($i = $keyCurrentSection; $i <= $keyLatestSection; $i++) {
907      array_push($compareNextSections, $oldNextSections[$i]);
908      array_push($compareNextLenghts, $oldLenghts[$i]);
909      array_push($compareNextVMax, $oldNextVMax[$i]);
910    }
911
912    if (sizeof($newNextSection) != ($numberOfSection)) {
913      $dataIsIdentical = false;
914    } else {
915      for ($i = 0; $i < $keyLatestSection - $keyCurrentSection; $i++) {
916        if ($newNextSection[$i] != $compareNextSections[$i] || $newNextLenghts[
            ↪ $i] != $compareNextLenghts[$i] || $newNextVMax[$i] !=
            ↪ $compareNextVMax[$i]) {
917          $dataIsIdentical = false;
918          break;
919        }
920      }
921    }
922
923    if (!$dataIsIdentical) {
924      echo "Die Fahrstraße des Zuges mit der ID: ", $id, " hat sich geändert.\n
          ↪ ";
```

```
925         calculateNextSections($id);
926         $adresse = $allUsedTrains[$id]["adresse"];
927         $allTimes[$adresse] = array();
928         checkIfFahrstrasseIsCorrrect($id);
929         calculateFahrtverlauf($id);
930       }
931     }
932   }
```

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(
        ↪ $indexReachedBetriebsstelle)];
64  }
65
66  // TODO: Zug steht schon am Ziel (ist das nötig?)
67  if ($targetSection == $currentSection && $targetPosition == $currentPosition)
      ↪  {
68    /*
69    if ($indexReachedBetriebsstelle == $globalIndexBetriebsstelleFreieFahrt) {
70      $indexReachedBetriebsstelle = -1;
71    }
```

```
72      */
73      if ($currentSpeed > 0) {
74        emergencyBreak($train["id"]);
75      } else {
76        return 0;
77      }
78      /*
79      $adress = $train["adresse"];
80      $return = array(array("live_position" => $targetPosition, "live_speed" =>
            ↪ $targetSpeed, "live_time" => $endTime, "live_relative_position" =>
            ↪ $targetPosition, "live_section" => $targetSection, "
            ↪ live_is_speed_change" => false, "live_target_reached" =>
            ↪ $reachedBetriebsstelle, "wendet" => $wendet, "id" => $train["id"], "
            ↪ betriebsstelle_name" => $targetBetriebsstelle));
81      $allTimes[$adress] = array_merge($allTimes[$adress], $return);
82      return 0;
83      */
84    }
85
86    // Wenn ein Infra-Abschnitt eine Geschwindigkeit zulässt, die größer als die
87    // zulässige Höchstgeschwindigkeit des Fahrzeugs ist, wird die
          ↪ Geschwindigkeit
88    // des Infra-Abschnitts reduziert.
89    if ($train_v_max != null) {
90      foreach ($next_sections as $sectionKey => $sectionValue) {
91        if ($next_v_max[$sectionKey] > $train_v_max) {
92          $next_v_max[$sectionKey] = $train_v_max;
93        }
94      }
95    }
96
97    // Ermittlung der Indexe des Start- und Zielabschnitts
98    foreach ($next_sections as $sectionKey => $sectionValue) {
99      if ($sectionValue == $currentSection) {
100       $indexCurrentSection = $sectionKey;
101     }
102
103     if ($sectionValue == $targetSection) {
104       $indexTargetSection = $sectionKey;
```

```
105     }
106   }
107
108   // Berechnet die kumulierten Abstände jedes Infra-Abschnitts für den Anfang
109   // und das Ende der Infra-Abschnitt von der aktuellen Fahrzeugposition
110   $returnCumulativeSections = createCumulativeSections($indexCurrentSection,
          ↪ $indexTargetSection, $currentPosition, $targetPosition, $next_lengths)
          ↪ ;
111   $cumulativeSectionLengthStart = $returnCumulativeSections[0];
112   $cumulativeSectionLengthEnd = $returnCumulativeSections[1];
113   $cumLengthEnd = array();
114   $cumLengthStart = array();
115   $sum = 0;
116
117   foreach ($next_lengths as $index => $value) {
118     if ($index >= $indexCurrentSection) {
119       $cumLengthStart[$index] = $sum;
120       $sum += $value;
121       $cumLengthEnd[$index] = $sum;
122     }
123   }
124
125   // Ermittlung der Distanz bis zum Ziel
126   $distanceToNextStop = $cumulativeSectionLengthEnd[$indexTargetSection];
127   if (getBrakeDistance($currentSpeed, $targetSpeed, $verzoegerung)>
          ↪ $distanceToNextStop && $currentSpeed != 0) {
128     if (!isset($distanceToNextStop)) {
129       emergencyBreak($train["id"]);
130
131       return 0;
132     } else {
133       emergencyBreak($train["id"], $distanceToNextStop);
134
135       return 0;
136     }
137   }
138
139   // Ermittlung der Längen und zulässigen Höchstgeschwindigkeiten der
140   // Infra-Abschnitte inkl. Zuglänge
```

```
141   global $next_v_max_mod;
142   global $next_lengths_mod;
143   global $indexCurrentSectionMod;
144   global $indexTargetSectionMod;
145
146   $next_v_max_mod = array();
147   $next_lengths_mod = array();
148   $indexCurrentSectionMod = null;
149   $indexTargetSectionMod = null;
150
151   if ($indexCurrentSection == $indexTargetSection) {
152     $next_lengths_mod = $next_lengths;
153     $next_v_max_mod = $next_v_max;
154     $indexCurrentSectionMod = $indexCurrentSection;
155     $indexTargetSectionMod = $indexTargetSection;
156     $next_lengths_mod[$indexTargetSectionMod] = $targetPosition;
157   } else {
158     $startPosition = 0;
159     $indexStartPosition = null;
160     $indexEndPosition = null;
161
162     do {
163       $reachedTargetSection = false;
164
165       for ($j = $indexCurrentSection; $j <= $indexTargetSection; $j++) {
166         if ($startPosition >= $cumLengthStart[$j] && $startPosition <
              ↪ $cumLengthEnd[$j]) {
167           $indexStartPosition = $j;
168         }
169       }
170
171       $endPosition = $cumLengthEnd[$indexStartPosition] + $train_length;
172       $current_v_max = $next_v_max[$indexStartPosition];
173
174       if ($endPosition >= $cumLengthEnd[$indexTargetSection]) {
175         $indexEndPosition = $indexTargetSection;
176         $endPosition = $cumLengthEnd[$indexTargetSection - 1] + $targetPosition;
177         $reachedTargetSection = true;
178       } else {
```

```
179         for ($j = $indexCurrentSection; $j <= $indexTargetSection; $j++) {
180           if ($endPosition >= $cumLengthStart[$j] && $endPosition <
                  ↪ $cumLengthEnd[$j]) {
181             $indexEndPosition = $j;
182           }
183         }
184       }
185
186       for ($j = $indexStartPosition + 1; $j <= $indexEndPosition; $j++) {
187         if ($next_v_max[$j] < $current_v_max) {
188           $endPosition = $cumLengthStart[$j];
189           $indexEndPosition = $j - 1;
190         }
191       }
192
193       if ($reachedTargetSection) {
194         if (!($endPosition >= $distanceToNextStop)) {
195           $reachedTargetSection = false;
196         }
197       }
198
199       array_push($next_lengths_mod, ($endPosition - $startPosition));
200       array_push($next_v_max_mod, $current_v_max);
201       $startPosition = $endPosition;
202     } while (!$reachedTargetSection);
203
204     $indexCurrentSectionMod = array_key_first($next_lengths_mod);
205     $indexTargetSectionMod = array_key_last($next_lengths_mod);
206   }
207
208   // Berechnet die kumulierten Abstände jedes Infra-Abschnitts für den Anfang
209   // und das Ende der Infra-Abschnitt von der aktuellen Fahrzeugposition
210   // inkl. der Fahrzeuglänge
211   $returnCumulativeSectionsMod = createCumulativeSections(
          ↪ $indexCurrentSectionMod, $indexTargetSectionMod, $currentPosition,
          ↪ $next_lengths_mod[$indexTargetSectionMod], $next_lengths_mod);
212
213   global $cumulativeSectionLengthStartMod;
214   global $cumulativeSectionLengthEndMod;
```

```
215
216 $cumulativeSectionLengthStartMod = $returnCumulativeSectionsMod[0];
217 $cumulativeSectionLengthEndMod = $returnCumulativeSectionsMod[1];
218 $minTimeOnSpeedIsPossible = checkIfItsPossible();
219 $v_maxFirstIteration = getVMaxBetweenTwoPoints($distanceToNextStop,
    ↪ $currentSpeed, $targetSpeed);
220
221 // Anpassung an die maximale Geschwindigkeit auf der Strecke
222 for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
223   if ($next_v_max[$i] > $maxSpeedNextSections) {
224     $maxSpeedNextSections = $next_v_max[$i];
225   }
226 }
227 if ($maxSpeedNextSections < $v_maxFirstIteration) {
228   $v_maxFirstIteration = $maxSpeedNextSections;
229 }
230
231 // Key Points für die erste Iteration erstellen.
232 array_push($keyPoints, createKeyPoint(0, getBrakeDistance($currentSpeed,
    ↪ $v_maxFirstIteration, $verzoegerung), $currentSpeed,
    ↪ $v_maxFirstIteration));
233 array_push($keyPoints, createKeyPoint(($distanceToNextStop - getBrakeDistance
    ↪ ($v_maxFirstIteration, $targetSpeed, $verzoegerung)),
    ↪ $distanceToNextStop, $v_maxFirstIteration, $targetSpeed));
234
235 //$trainChange = convertKeyPointsToTrainChangeArray($keyPoints);
236 $trainChange = createTrainChanges(true);
237 $trainPositionChange = $trainChange[0];
238 $trainSpeedChange = $trainChange[1];
239 $speedOverPositionAllIterations = array();
240
241 // Überprüfung, ob das Fahrzeug in Infra-Abschnitten zu schnell ist
242 while (checkIfTrainIsToFastInCertainSections()["failed"]) {
243   $tempKeyPoints = $keyPoints;
244
245   // Berechnung der Echtzeitdaten
246   $trainChange = createTrainChanges(true);
247   $trainPositionChange = $trainChange[0];
248   $trainSpeedChange = $trainChange[1];
```

```
249
250     // Hinzufügen der Echtzeitdaten des vorherigen Iterationsschritt
251     // für die Visualisierung
252     array_push($speedOverPositionAllIterations, array($trainPositionChange,
            ↪ $trainSpeedChange));
253
254     // Überprüfung, ob durch den Fahrtverlauf zulässige Höchst-
255     // geschwindigkeiten überschritten werden
256     $keyPoints = recalculateKeyPoints($tempKeyPoints);
257     $localKeyPointsTwo = array();
258
259     // Entfernen von doppelten $keyPoints
260     for ($i = 0; $i < sizeof($keyPoints); $i++) {
261       if ($i < sizeof($keyPoints) - 1) {
262         if (!($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"])) {
263           array_push($localKeyPointsTwo, $keyPoints[$i]);
264         } else {
265           $i++;
266         }
267       } else {
268         array_push($localKeyPointsTwo, $keyPoints[$i]);
269       }
270     }
271
272     // Berechnung der Echtzeitdaten nach der Neukalibrierung
273     $keyPoints = $localKeyPointsTwo;
274     $trainChange = createTrainChanges(true);
275     $trainPositionChange = $trainChange[0];
276     $trainSpeedChange = $trainChange[1];
277   }
278
279   // Fügt die aktuelle Zeit zum ersten $keyPoint hinzu
280   $keyPoints[0]["time_0"] = $startTime;
281   $keyPoints = deleteDoubledKeyPoints($keyPoints);
282   $keyPoints = calculateTimeFromKeyPoints();
283
284   if ($useMinTimeOnSpeed && $minTimeOnSpeedIsPossible) {
```

```
285     array_push($speedOverPositionAllIterations, array($trainPositionChange,
            ↪ $trainSpeedChange));
286     toShortOnOneSpeed();
287   }
288
289   // Ermittlung der Echtzeitdaten
290   $trainChange = createTrainChanges(true);
291   $trainPositionChange = $trainChange[0];
292   $trainSpeedChange = $trainChange[1];
293   $timeToNextStop = end($keyPoints)["time_1"] - $keyPoints[0]["time_0"];
294
295   // Überprüfung, ob das Fahrzeug mit einer Verspätung am Ziel ankommt.
296   // Fahrzeuge, die ohne Fahrplan fahren, werden nicht betrachtet.
297   if (!$freieFahrt) {
298     if ($timeToNextStop > $maxTimeToNextStop) {
299       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";
300     } else {
301       echo "Aktuell benötigt der Zug mit der Adresse ", $train["adresse"], " ",
              ↪  number_format($timeToNextStop, 2), " Sekunden, obwohl er ",
              ↪ number_format($maxTimeToNextStop, 2), " Sekunden zur Verfügung hat
              ↪ .\n";
302
303       if ($slowDownIfTooEarly) {
304         echo "Evtl. könnte der Zug zwischendurch die Geschwindigkeit verringern,
                ↪  um Energie zu sparen.";
305
306         array_push($speedOverPositionAllIterations, array($trainPositionChange,
                ↪ $trainSpeedChange));
307         $keyPointsPreviousStep = array();
308         $finish = false;
309         $possibleSpeedRange = null;
310         $returnSpeedDecrease = checkIfTheSpeedCanBeDecreased();
311
312         while ($returnSpeedDecrease["possible"] && !$finish) {
313           $possibleSpeedRange = findMaxSpeed($returnSpeedDecrease);
314
```

```
315             if ($possibleSpeedRange["min_speed"] == $possibleSpeedRange["max_speed
                    ↪ "]) {
316               break;
317             }
318
319             $localKeyPoints = $keyPoints;
320             $newCalculatedTime = null;
321             $newKeyPoints = null;
322
323             for ($i = $possibleSpeedRange["max_speed"]; $i >= $possibleSpeedRange[
                    ↪ "min_speed"]; $i = $i - 10) {
324               $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["
                      ↪ speed_1"] = $i;
325               $localKeyPoints[$possibleSpeedRange["first_key_point_index"] + 1]["
                      ↪ speed_0"] = $i;
326               $localKeyPoints[$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"]);
327               $localKeyPoints[$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));
328               $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
329               $newCalculatedTime = $localKeyPoints[array_key_last($localKeyPoints)
                      ↪ ]["time_1"];
330
331               if ($i == 10) {
332                 if ($newCalculatedTime > $maxTimeToNextStop) {
333                   $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["
                          ↪ speed_1"] = $i + 10;
334                   $localKeyPoints[$possibleSpeedRange["first_key_point_index"] +
                          ↪ 1]["speed_0"] = $i + 10;
335                   $localKeyPoints[$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"
                ↪ ]);
            $localKeyPoints[$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));
          }

          $finish = true;
          $newKeyPoints = $localKeyPoints;
          break;
        }
        if (($newCalculatedTime - $startTime) > $maxTimeToNextStop) {
          if ($i == $possibleSpeedRange["max_speed"]) {
            $localKeyPoints = $keyPointsPreviousStep;
            $localKeyPoints = deleteDoubledKeyPoints($localKeyPoints);
            $keyPoints = $localKeyPoints;
            $finish = true;
            break;
          }
          $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["
              ↪ speed_1"] = $i + 10;
          $localKeyPoints[$possibleSpeedRange["first_key_point_index"] + 1][
              ↪ "speed_0"] = $i + 10;
          $localKeyPoints[$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"
              ↪ ]);
          $localKeyPoints[$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));
          $newKeyPoints = $localKeyPoints;
```

```
356                 $finish = true;
357                 $keyPoints = $localKeyPoints;
358
359                 break;
360               }
361               if ($i == $possibleSpeedRange["min_speed"]) {
362                 $newKeyPoints = $localKeyPoints;
363                 $newKeyPoints = deleteDoubledKeyPoints($newKeyPoints);
364                 $keyPoints = $newKeyPoints;
365                 break;
366               }
367               $newKeyPoints = $localKeyPoints;
368             }
369             $keyPointsPreviousStep = $localKeyPoints;
370
371             if ($newKeyPoints != null) {
372               $keyPoints = $newKeyPoints;
373             }
374
375             $keyPoints = deleteDoubledKeyPoints($keyPoints);
376             $returnSpeedDecrease = checkIfTheSpeedCanBeDecreased();
377           }
378
379           $keyPoints = calculateTimeFromKeyPoints();
380
381           if ($useSpeedFineTuning && $returnSpeedDecrease["possible"]) {
382             $trainChangeReturn = createTrainChanges(true);
383             $trainPositionChange = $trainChangeReturn[0];
384             $trainSpeedChange = $trainChangeReturn[1];
385             $newCalculatedTime = $keyPoints[array_key_last($keyPoints)]["time_1"];
386             speedFineTuning(($maxTimeToNextStop - ($newCalculatedTime - $startTime
                  ↪ )), $returnSpeedDecrease["range"][array_key_last(
                  ↪ $returnSpeedDecrease["range"])]["KeyPoint_index"]);
387           }
388
389           $keyPoints = calculateTimeFromKeyPoints();
390           $timeToNextStop = end($keyPoints)["time_1"] - $keyPoints[0]["time_0"];
391
```

```
392         echo "\nDurch die Anpassung der Geschwindigkeit benötigt der Zug mit der
                ↪  Adresse ", $train["adresse"], " jetzt ", number_format(
                ↪ $timeToNextStop, 2), " Sekunden bis\n";
393
394         if (abs($timeToNextStop - $maxTimeToNextStop) <
                ↪ $globalFloatingPointNumbersRoundingError) {
395           echo "zum nächsten planmäßigen Halt (", $targetBetriebsstelle, ") und
                  ↪ wird diesen genau pünktlich erreichen.\n";
396         } else if (($timeToNextStop - $maxTimeToNextStop) > 0) {
397           echo "zum nächsten planmäßigen Halt (", $targetBetriebsstelle, ") und
                  ↪ wird diesen mit einer Verspätung von ", number_format(
                  ↪ $timeToNextStop - $maxTimeToNextStop, 2), " Sekunden erreichen
                  ↪ .\n";
398         } else {
399           echo "zum nächsten planmäßigen Halt (", $targetBetriebsstelle, ") und
                  ↪ wird diesen ", number_format($timeToNextStop -
                  ↪ $maxTimeToNextStop, 2), " Sekunden zu früh erreichen.\n";
400         }
401       } else {
402         echo "Dadurch, dass \$slowDownIfTooEarly = true ist, wird das Fahrzeug "
                ↪ , number_format($maxTimeToNextStop - $timeToNextStop, 2), "
                ↪ Sekunden zu früh am Ziel ankommen.";
403       }
404     }
405   } else {
406     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";
407   }
408
409   // Berechnung der Echtzeitdaten
410   $returnTrainChanges = createTrainChanges(false);
411   $trainPositionChange = $returnTrainChanges[0];
412   $trainSpeedChange = $returnTrainChanges[1];
413   $trainTimeChange = $returnTrainChanges[2];
414   $trainRelativePosition = $returnTrainChanges[3];
415   $trainSection = $returnTrainChanges[4];
416   $trainIsSpeedChange = $returnTrainChanges[5];
```

```
417   $trainTargetReached = array();
418   $trainBetriebsstelleName = array();
419   $trainWendet = array();
420   $allReachedTargets = array();
421   $allreachedInfrasIndex = array();
422   $allreachedInfrasID = array();
423   $allreachedInfrasUsed = array();
424
425   foreach ($allreachedInfras as $value) {
426     array_push($allreachedInfrasIndex, $value["index"]);
427     array_push($allreachedInfrasID, $value["infra"]);
428   }
429
430   foreach ($trainPositionChange as $key => $value) {
431     $trainBetriebsstelleName[$key] = $targetBetriebsstelle;
432     if (array_key_last($trainPositionChange) != $key) {
433       $trainTargetReached[$key] = false;
434       $trainWendet[$key] = false;
435     } else {
436       if ($wendet) {
437         $trainWendet[$key] = true;
438       } else {
439         $trainWendet[$key] = false;
440       }
441
442       if ($reachedBetriebsstelle) {
443         $trainTargetReached[$key] = true;
444       } else {
445         $trainTargetReached[$key] = false;
446       }
447     }
448   }
449
450   for($i = sizeof($trainSection) - 1; $i >= 0; $i--) {
451     if (in_array($trainSection[$i], $allreachedInfrasID) && !in_array(
          ↪ $trainSection[$i], $allreachedInfrasUsed)) {
452       array_push($allreachedInfrasUsed, $trainSection[$i]);
453       $Infraindex = array_search($trainSection[$i], $allreachedInfrasID);
454       $allReachedTargets[$i] = $allreachedInfrasIndex[$Infraindex];
```

```
455     } else {
456       $allReachedTargets[$i] = null;
457     }
458   }
459   ksort($allReachedTargets);
460   $returnArray = array();
461   $adress = $train["adresse"];
462   $trainID = array();
463   $id = $train["id"];
464
465   foreach ($trainPositionChange as $key => $value) {
466     $trainID[$key] = $id;
467   }
468
469   foreach ($trainPositionChange as $trainPositionChangeIndex =>
        ↪ $trainPositionChangeValue) {
470     array_push($returnArray, array("live_position" => $trainPositionChangeValue
          ↪ ,
471       "live_speed" => $trainSpeedChange[$trainPositionChangeIndex],
472       "live_time" => $trainTimeChange[$trainPositionChangeIndex],
473       "live_relative_position" => $trainRelativePosition[
            ↪ $trainPositionChangeIndex],
474       "live_section" => $trainSection[$trainPositionChangeIndex],
475       "live_is_speed_change" => $trainIsSpeedChange[$trainPositionChangeIndex],
476       "live_target_reached" => $trainTargetReached[$trainPositionChangeIndex],
477       "id" => $trainID[$trainPositionChangeIndex],
478       "wendet" => $trainWendet[$trainPositionChangeIndex],
479       "betriebsstelle" => $trainBetriebsstelleName[$trainPositionChangeIndex],
480       "live_all_targets_reached" => $allReachedTargets[
            ↪ $trainPositionChangeIndex]));
481   }
482
483   $allTimes[$adress] = $returnArray;
484   safeTrainChangeToJSONFile($indexCurrentSection, $indexTargetSection,
        ↪ $indexCurrentSectionMod, $indexTargetSectionMod,
        ↪ $speedOverPositionAllIterations);
485
486   return (end($trainTimeChange) - $trainTimeChange[0]) - ($endTime - $startTime
        ↪ );
```

```
487 }
488
489 // TODO: bei sizeof($v_max) == 0 muss entweder eine Notbresmung passieren oder
        ↪ funktion wird abgebrochen (leere Echtzeitdaten)
490 // Ermittelt die maximale Geschwindigkeit zwischen zwei Punkten
491 function getVMaxBetweenTwoPoints(float $distance, int $v_0, int $v_1) {
492
493   global $verzoegerung;
494   global $globalFloatingPointNumbersRoundingError;
495
496   $v_max = array();
497
498   for ($i = 0; $i <= 120; $i = $i + 10) {
499     if ((getBrakeDistance($v_0, $i, $verzoegerung) + getBrakeDistance($i, $v_1,
          ↪  $verzoegerung)) < ($distance +
          ↪ $globalFloatingPointNumbersRoundingError)) {
500       array_push($v_max, $i);
501     }
502   }
503
504   if (sizeof($v_max) == 0) {
505     if ($v_0 == 0 && $v_1 == 0 && $distance > 0) {
506       echo "Der zug müsste langsamer als 10 km/h fahren, um das Ziel zu
            ↪ erreichen.";
507     } else {
508       //emergencyBreak($id);
509     }
510   } else {
511     if ($v_0 == $v_1 && max($v_max) < $v_0) {
512       $v_max = array($v_0);
513     }
514   }
515
516   return max($v_max);
517 }
518
519 // Erstellt einen $keyPoint
520 function createKeyPoint (float $position_0, float $position_1, int $speed_0,
        ↪ int $speed_1) {
```

```
521   return array("position_0" => $position_0, "position_1" => $position_1, "
          ↪ speed_0" => $speed_0, "speed_1" => $speed_1);
522 }
523
524 // Ermittelt aus den $keyPoint die Echtzeitdaten (nur Geschwindigkeit und
        ↪ Position)
525 function convertKeyPointsToTrainChangeArray (array $keyPoints) {
526
527   global $verzoegerung;
528
529   $trainSpeedChangeReturn = array();
530   $trainPositionChnageReturn = array();
531   array_push($trainPositionChnageReturn, $keyPoints[0]["position_0"]);
532   array_push($trainSpeedChangeReturn, $keyPoints[0]["speed_0"]);
533
534   for ($i = 0; $i <= (sizeof($keyPoints) - 2); $i++) {
535     if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
536       for ($j = $keyPoints[$i]["speed_0"]; $j < $keyPoints[$i]["speed_1"]; $j =
            ↪  $j + 2) {
537         array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn)
              ↪ + getBrakeDistance($j, ($j + 2), $verzoegerung)));
538         array_push($trainSpeedChangeReturn, ($j + 2));
539       }
540     } elseif ($keyPoints[$i]["speed_0"] > $keyPoints[$i]["speed_1"]) {
541       for ($j = $keyPoints[$i]["speed_0"]; $j > $keyPoints[$i]["speed_1"]; $j =
            ↪  $j - 2) {
542         array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn)
              ↪ + getBrakeDistance($j, ($j - 2), $verzoegerung)));
543         array_push($trainSpeedChangeReturn, ($j - 2));
544       }
545     }
546     array_push($trainPositionChnageReturn, $keyPoints[$i + 1]["position_0"]);
547     array_push($trainSpeedChangeReturn, $keyPoints[$i + 1]["speed_0"]);
548   }
549
550   if (end($keyPoints)["speed_0"] < end($keyPoints)["speed_1"]) {
551     for ($j = end($keyPoints)["speed_0"]; $j < end($keyPoints)["speed_1"]; $j =
          ↪  $j + 2) {
```

```
552       array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
              ↪ getBrakeDistance($j, ($j + 2), $verzoegerung)));
553       array_push($trainSpeedChangeReturn, ($j + 2));
554     }
555   } else if (end($keyPoints)["speed_0"] > end($keyPoints)["speed_1"]) {
556     for ($j = end($keyPoints)["speed_0"]; $j > end($keyPoints)["speed_1"]; $j =
          ↪ $j - 2) {
557       array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
              ↪ getBrakeDistance($j, ($j - 2), $verzoegerung)));
558       array_push($trainSpeedChangeReturn, ($j - 2));
559     }
560   }
561
562   return array($trainPositionChnageReturn, $trainSpeedChangeReturn);
563 }
564
565 // Wandelt die Daten der Infra-Abschnitte und der Iterationsschritte der
566 // Fahrtverlaufsberechnung in JSON-Dateien um, damit die Fahrtverläufe
567 // visuell dargestellt werden können.
568 function safeTrainChangeToJSONFile(int $indexCurrentSection, int
        ↪ $indexTargetSection, int $indexCurrentSectionMod, int
        ↪ $indexTargetSectionMod, array $speedOverPositionAllIterations) {
569
570   global $trainPositionChange;
571   global $trainSpeedChange;
572   global $next_v_max;
573   global $cumulativeSectionLengthEnd;
574   global $next_v_max_mod;
575   global $cumulativeSectionLengthEndMod;
576
577   $speedOverPosition = array_map('toArr', $trainPositionChange,
          ↪ $trainSpeedChange);
578   $speedOverPosition = json_encode($speedOverPosition);
579   $fp = fopen('../json/speedOverPosition.json', 'w');
580   fwrite($fp, $speedOverPosition);
581   fclose($fp);
582
583   $v_maxFromUsedSections = array();
584
```

```
585   for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
586     array_push($v_maxFromUsedSections, $next_v_max[$i]);
587   }
588
589   $VMaxOverCumulativeSections = array_map('toArr', $cumulativeSectionLengthEnd,
          ↪  $v_maxFromUsedSections);
590   $VMaxOverPositionsJSon = json_encode($VMaxOverCumulativeSections);
591   $fp = fopen('../json/VMaxOverCumulativeSections.json', 'w');
592   fwrite($fp, $VMaxOverPositionsJSon);
593   fclose($fp);
594
595   $v_maxFromUsedSections = array();
596
597   for ($i = $indexCurrentSectionMod; $i <= $indexTargetSectionMod; $i++) {
598     array_push($v_maxFromUsedSections, $next_v_max_mod[$i]);
599   }
600
601   $VMaxOverCumulativeSectionsMod = array_map('toArr',
          ↪ $cumulativeSectionLengthEndMod, $v_maxFromUsedSections);
602   $VMaxOverPositionsJSon = json_encode($VMaxOverCumulativeSectionsMod);
603   $fp = fopen('../json/VMaxOverCumulativeSectionsMod.json', 'w');
604   fwrite($fp, $VMaxOverPositionsJSon);
605   fclose($fp);
606
607   $jsonReturn = array();
608
609   for ($i = 0; $i < sizeof($speedOverPositionAllIterations); $i++) {
610     $iteration = array_map('toArr', $speedOverPositionAllIterations[$i][0],
          ↪ $speedOverPositionAllIterations[$i][1]);
611     array_push($jsonReturn, $iteration);
612   }
613
614   $speedOverPosition = json_encode($jsonReturn);
615   $fp = fopen('../json/speedOverPosition_prevIterations.json', 'w');
616   fwrite($fp, $speedOverPosition);
617   fclose($fp);
618 }
619
620 // Überprüft, ob das Fahrzeug in Infra-Abschnitten die zulässige
```

```php
// 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 {
```

```
655     return array("failed" => true, "failed_sections" => array_unique(
            ↪ $faildSections));
656   }
657 }
658
659 // Löscht $keyPoint, bei denen Start- und Zielgeschwindigkeit identisch ist.
660 // Der erste $keyPoint wird dabei nicht betrachtet.
661 function deleteDoubledKeyPoints($temporaryKeyPoints) {
662   do {
663     $foundDoubledKeyPoints = false;
664     $doubledIndex = array();
665
666     for ($i = 1; $i < (sizeof($temporaryKeyPoints) - 1); $i++) {
667       if ($temporaryKeyPoints[$i]["speed_0"] == $temporaryKeyPoints[$i]["
              ↪ speed_1"]) {
668         $foundDoubledKeyPoints = true;
669         array_push($doubledIndex, $i);
670       }
671     }
672
673     foreach ($doubledIndex as $index) {
674       unset($temporaryKeyPoints[$index]);
675     }
676
677     $temporaryKeyPoints = array_values($temporaryKeyPoints);
678   } while ($foundDoubledKeyPoints);
679
680   return $temporaryKeyPoints;
681 }
682
683 /*
684 function calculateTrainTimeChange() {
685
686   global $keyPoints;
687   global $verzoegerung;
688
689   $returnAllTimes = array();
690   $returnAllTimes[0] = $keyPoints[0]["time_0"];
691
```

```
692 for ($i = 0; $i < (sizeof($keyPoints) - 1); $i++) {
693   if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
694     for ($j = $keyPoints[$i]["speed_0"] + 2; $j <= $keyPoints[$i]["speed_1"];
          ↪  $j = $j + 2) {
695       array_push($returnAllTimes, (end($returnAllTimes) + (getBrakeTime($j -
            ↪ 2, $j, $verzoegerung))));
696
697     }
698     array_push($returnAllTimes, (end($returnAllTimes) +
          ↪ distanceWithSpeedToTime($keyPoints[$i]["speed_1"], ($keyPoints[$i +
          ↪  1]["position_0"] - $keyPoints[$i]["position_1"]))));
699
700   } else if ($keyPoints[$i]["speed_0"] > $keyPoints[$i]["speed_1"]) {
701     for ($j = $keyPoints[$i]["speed_0"] - 2; $j >= $keyPoints[$i]["speed_1"];
          ↪  $j = $j - 2) {
702       array_push($returnAllTimes, (end($returnAllTimes) + (getBrakeTime($j +
            ↪ 2, $j, $verzoegerung))));
703
704     }
705     array_push($returnAllTimes, (end($returnAllTimes) +
          ↪ distanceWithSpeedToTime($keyPoints[$i]["speed_1"], ($keyPoints[$i +
          ↪  1]["position_0"] - $keyPoints[$i]["position_1"]))));
706   }
707 }
708
709 if ($keyPoints[array_key_last($keyPoints)]["speed_0"] < $keyPoints[
      ↪ array_key_last($keyPoints)]["speed_1"]) {
710   for ($i = ($keyPoints[array_key_last($keyPoints)]["speed_0"] + 2); $i <=
        ↪ $keyPoints[array_key_last($keyPoints)]["speed_1"]; $i = $i + 2) {
711     array_push($returnAllTimes, (end($returnAllTimes) + (getBrakeTime($i - 2,
          ↪  $i, $verzoegerung))));
712   }
713 } else if ($keyPoints[array_key_last($keyPoints)]["speed_0"] > $keyPoints[
      ↪ array_key_last($keyPoints)]["speed_1"]) {
714   for ($i = ($keyPoints[array_key_last($keyPoints)]["speed_0"] - 2); $i >=
        ↪ $keyPoints[array_key_last($keyPoints)]["speed_1"]; $i = $i - 2) {
715     array_push($returnAllTimes, (end($returnAllTimes) + (getBrakeTime($i + 2,
          ↪  $i, $verzoegerung))));
716   }
```

```
717   }
718   return $returnAllTimes;
719 }
720 */
721
722 // Ermittelt die Zeiten der $keyPoint ausgehend vom ersten $keyPoint. Mit dem
723 // Parameter $inputKeyPoints können $keyPoints übergeben werden, bei den die
724 // Zeit ermittelt wird. Wenn keine $keyPoints übergeben werden, werden die
725 // globalen $keyPoints verwendet. Mit dem Parameter $skippingKeys können
726 // $keyPoints übersprungen werden. Das auslassen von $keyPoints ist für die
727 // Funktion postponeSubsection() relevant.
728 function calculateTimeFromKeyPoints($inputKeyPoints = null, $skippingKeys =
      ↪ null) {
729
730   global $keyPoints;
731   global $verzoegerung;
732
733   if ($inputKeyPoints == null) {
734     $localKeyPoints = $keyPoints;
735   } else {
736     $localKeyPoints = $inputKeyPoints;
737   }
738
739   $keys = array_keys($localKeyPoints);
740
741   if ($skippingKeys != null) {
742     foreach ($skippingKeys as $skip) {
743       unset($keys[array_search($skip, $keys)]);
744     }
745   }
746
747   $keys = array_values($keys);
748
749   for ($i = 0; $i < (sizeof($keys) - 1); $i++) {
750     $localKeyPoints[$keys[$i]]["time_1"] = getBrakeTime($localKeyPoints[$keys[
        ↪ $i]]["speed_0"], $localKeyPoints[$keys[$i]]["speed_1"],
        ↪ $verzoegerung) + $localKeyPoints[$keys[$i]]["time_0"];
751     $localKeyPoints[$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"];
752   }
753
754   $localKeyPoints[end($keys)]["time_1"] = getBrakeTime($localKeyPoints[end(
        ↪ $keys)]["speed_0"], $localKeyPoints[end($keys)]["speed_1"],
        ↪ $verzoegerung) + $localKeyPoints[end($keys)]["time_0"];
755
756   return $localKeyPoints;
757 }
758
759 // Ermittelt die Echtzeitdaten eines Fahrtverlaufs auf Grundlage der $keyPoints
      ↪ .
760 // Mit dem Parameter $onlyPositionAndSpeed kann festgelegt werden, ob nur die
761 // Position und Geschwindigkeit berechnet werden soll oder alle benötigten
      ↪ Daten.
762 function createTrainChanges(bool $onlyPositionAndSpeed) {
763
764   global $keyPoints;
765   global $verzoegerung;
766   global $cumulativeSectionLengthStart;
767   global $cumulativeSectionLengthEnd;
768   global $next_sections;
769   global $indexCurrentSection;
770   global $indexTargetSection;
771   global $currentPosition;
772   global $globalFloatingPointNumbersRoundingError;
773   global $globalDistanceUpdateInterval;
774
775   $returnTrainSpeedChange = array();
776   $returnTrainTimeChange = array();
777   $returnTrainPositionChange = array();
778   $returnTrainRelativePosition = array();
779   $returnTrainSection = array();
780   $returnIsSpeedChange = array();
781
782   // Ermittelt für alle bis auf den letzten $keyPoint die Echtzeitdaten der
783   // Zeit, Geschwindigkeit und Position
784   for ($i = 0; $i < (sizeof($keyPoints) - 1); $i++) {
```

```
785     array_push($returnTrainTimeChange, $keyPoints[$i]["time_0"]);
786     array_push($returnTrainSpeedChange, $keyPoints[$i]["speed_0"]);
787     array_push($returnTrainPositionChange, $keyPoints[$i]["position_0"]);
788     array_push($returnIsSpeedChange, true);
789     if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
790       for ($j = ($keyPoints[$i]["speed_0"] + 2); $j <= $keyPoints[$i]["speed_1"
          ↪ ]; $j = $j + 2) {
791         array_push($returnTrainPositionChange, (end($returnTrainPositionChange)
              ↪ + getBrakeDistance(($j - 2), $j, $verzoegerung)));
792         array_push($returnTrainSpeedChange, $j);
793         array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
              ↪ getBrakeTime(($j - 2), $j, $verzoegerung))));
794         array_push($returnIsSpeedChange, true);
795       }
796     } else {
797       for ($j = ($keyPoints[$i]["speed_0"] - 2); $j >= $keyPoints[$i]["speed_1"
          ↪ ]; $j = $j - 2) {
798         array_push($returnTrainPositionChange, (end($returnTrainPositionChange)
              ↪ + getBrakeDistance(($j + 2), $j, $verzoegerung)));
799         array_push($returnTrainSpeedChange, $j);
800         array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
              ↪ getBrakeTime(($j + 2), $j, $verzoegerung))));
801         array_push($returnIsSpeedChange, true);
802       }
803     }
804
805     // Ermittelt für die Strecke zwischen zwei $keyPoints die Echtzeitdaten
806     // der Zeit, Geschwindigkeit und Position
807     $startPosition = $keyPoints[$i]["position_1"];
808     $endPosition = $keyPoints[$i + 1]["position_0"];
809     $speedToNextKeyPoint = $keyPoints[$i]["speed_1"];
810     $distanceForOneTimeInterval = $speedToNextKeyPoint / 3.6;
811
812     for ($position = $startPosition + $distanceForOneTimeInterval; $position <
          ↪ $endPosition; $position = $position + $distanceForOneTimeInterval) {
813       array_push($returnTrainPositionChange, $position);
814       array_push($returnTrainSpeedChange, $speedToNextKeyPoint);
815       array_push($returnTrainTimeChange, end($returnTrainTimeChange) +
              ↪ $globalDistanceUpdateInterval);
```

```
816       array_push($returnIsSpeedChange, false);
817     }
818   }
819   array_push($returnTrainPositionChange, $keyPoints[array_key_last($keyPoints)
        ↪ ]["position_1"] - getBrakeDistance($keyPoints[array_key_last(
        ↪ $keyPoints)]["speed_0"],$keyPoints[array_key_last($keyPoints)]["
        ↪ speed_1"],$verzoegerung));
820   array_push($returnTrainSpeedChange, $keyPoints[array_key_last($keyPoints)]["
        ↪ speed_0"]);
821   array_push($returnTrainTimeChange, $keyPoints[array_key_last($keyPoints)]["
        ↪ time_0"]);
822   array_push($returnIsSpeedChange, true);
823
824   // Ermittelt für den letzten $keyPoint die Echtzeitdaten der Zeit,
825   // Geschwindigkeit und Position
826   if ($keyPoints[array_key_last($keyPoints)]["speed_0"] < $keyPoints[
        ↪ array_key_last($keyPoints)]["speed_1"]) {
827     for ($j = ($keyPoints[array_key_last($keyPoints)]["speed_0"] + 2); $j <=
          ↪ $keyPoints[array_key_last($keyPoints)]["speed_1"]; $j = $j + 2) {
828       array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
            ↪  getBrakeDistance(($j - 2), $j, $verzoegerung)));
829       array_push($returnTrainSpeedChange, $j);
830       array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
            ↪ getBrakeTime(($j - 2), $j, $verzoegerung))));
831       array_push($returnIsSpeedChange, true);
832     }
833   } else {
834     for ($j = ($keyPoints[array_key_last($keyPoints)]["speed_0"] - 2); $j >=
          ↪ $keyPoints[array_key_last($keyPoints)]["speed_1"]; $j = $j - 2) {
835       array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
            ↪  getBrakeDistance(($j + 2), $j, $verzoegerung)));
836       array_push($returnTrainSpeedChange, $j);
837       array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
            ↪ getBrakeTime(($j + 2), $j, $verzoegerung))));
838       array_push($returnIsSpeedChange, true);
839     }
840   }
841
842   if ($onlyPositionAndSpeed) {
```

```
843     return array($returnTrainPositionChange, $returnTrainSpeedChange);
844   } else {
845     // Ermittelt die relativen Positionen innerhalb der Infra-Abschnitte
846     // zu den absoluten Positionen
847     foreach ($returnTrainPositionChange as $absolutPositionKey =>
          ↪ $absolutPositionValue) {
848       foreach ($cumulativeSectionLengthStart as $sectionStartKey =>
            ↪ $sectionStartValue) {
849         if ($absolutPositionValue >= $sectionStartValue && $absolutPositionValue
              ↪  < $cumulativeSectionLengthEnd[$sectionStartKey]) {
850           if ($sectionStartKey == $indexCurrentSection && $sectionStartKey ==
                ↪ $indexTargetSection) {
851             $returnTrainRelativePosition[$absolutPositionKey] =
                  ↪ $absolutPositionValue + $currentPosition;
852           } else if ($sectionStartKey == $indexCurrentSection) {
853             $returnTrainRelativePosition[$absolutPositionKey] =
                  ↪ $absolutPositionValue + $currentPosition;
854           } else if ($sectionStartKey == $indexCurrentSection) {
855             $returnTrainRelativePosition[$absolutPositionKey] =
                  ↪ $absolutPositionValue - $sectionStartValue;
856           } else {
857             $returnTrainRelativePosition[$absolutPositionKey] =
                  ↪ $absolutPositionValue - $sectionStartValue;
858           }
859           break;
860         } else if ($absolutPositionKey == array_key_last(
              ↪ $returnTrainPositionChange) && abs($absolutPositionValue -
              ↪ floatval($cumulativeSectionLengthEnd[$sectionStartKey])) <
              ↪ $globalFloatingPointNumbersRoundingError) {
861           $returnTrainRelativePosition[$absolutPositionKey] =
                  ↪ $cumulativeSectionLengthEnd[$sectionStartKey] -
                  ↪ $sectionStartValue;
862           break;
863         } else {
864           debugMessage("Eine absolute Position konnte keine relativen Position
                ↪ in einem Abschnitt zugeordnet werden!");
865         }
866       }
867     }
```

```
868
869     // Ermittelt die Infra-Abschnitte der Echtzeitdaten zu den absoluten
            ↪ Positionen
870     foreach ($returnTrainPositionChange as $absolutPositionKey =>
            ↪ $absolutPositionValue) {
871       foreach ($cumulativeSectionLengthStart as $sectionStartKey =>
              ↪ $sectionStartValue) {
872         if ($absolutPositionValue >= $sectionStartValue && $absolutPositionValue
                ↪  < $cumulativeSectionLengthEnd[$sectionStartKey]) {
873           if ($sectionStartKey == $indexCurrentSection && $sectionStartKey ==
                  ↪ $indexTargetSection) {
874             $returnTrainSection[$absolutPositionKey] = $next_sections[
                    ↪ $sectionStartKey];
875           } else if ($sectionStartKey == $indexCurrentSection) {
876             $returnTrainSection[$absolutPositionKey] = $next_sections[
                    ↪ $sectionStartKey];
877           } else if ($sectionStartKey == $indexTargetSection) {
878             $returnTrainSection[$absolutPositionKey] = $next_sections[
                    ↪ $sectionStartKey];
879           } else {
880             $returnTrainSection[$absolutPositionKey] = $next_sections[
                    ↪ $sectionStartKey];
881           }
882           break;
883         } else if ($absolutPositionKey == array_key_last(
                ↪ $returnTrainPositionChange) && abs($absolutPositionValue -
                ↪ floatval($cumulativeSectionLengthEnd[$sectionStartKey])) <
                ↪ $globalFloatingPointNumbersRoundingError) {
884           $returnTrainSection[$absolutPositionKey] = $next_sections[
                  ↪ $sectionStartKey];
885           break;
886         } else {
887           debugMessage("Eine absolute Position konnte keine relativen Position
                  ↪ in einem Abschnitt zugeordnet werden!");
888         }
889       }
890     }
891     return array($returnTrainPositionChange, $returnTrainSpeedChange,
            ↪ $returnTrainTimeChange, $returnTrainRelativePosition,
```

```
        ↪ $returnTrainSection, $returnIsSpeedChange);
  }
}

// Überprüft, ob es zwischen zwei benachbarten $keyPoints zu einer
    ↪ Geschwindigkeits-
// überschreitung kommt.
function recalculateKeyPoints(array $tempKeyPoints) {

  $returnKeyPoints = array();
  $numberOfPairs = sizeof($tempKeyPoints) / 2;

  for($j = 0; $j < $numberOfPairs; $j++) {
    $i = $j * 2;
    $return = checkBetweenTwoKeyPoints($tempKeyPoints, $i);

    foreach ($return as $keyPoint) {
      array_push($returnKeyPoints, $keyPoint);
    }
  }

  return $returnKeyPoints;
}

// Überprüft, ob zwischen zwei $keyPoints die zulässige Höchstgeschwindigkeit
// überschritten wird
function checkBetweenTwoKeyPoints(array $temKeyPoints, int $keyPointIndex) {

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

  $failedSections = array();
  $groupedFailedSections = array();
  $returnKeyPoints = array();
```

```
929   $failedPositions = array();
930   $failedSpeeds = array();
931
932   foreach ($trainPositionChange as $trainPositionChangeKey =>
          ↪ $trainPositionChangeValue) {
933     if ($trainPositionChangeValue >= $temKeyPoints[$keyPointIndex]["position_0"
          ↪ ] && $trainPositionChangeValue <= $temKeyPoints[$keyPointIndex + 1][
          ↪ "position_1"]) {
934       foreach ($cumulativeSectionLengthStartMod as
            ↪ $cumulativeSectionLengthStartKey =>
            ↪ $cumulativeSectionLengthStartValue) {
935         if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
936           if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
                ↪ $cumulativeSectionLengthStartKey - 1]) {
937             array_push($failedSections, ($cumulativeSectionLengthStartKey - 1));
938             array_push($failedSpeeds, $trainSpeedChange[$trainPositionChangeKey
                  ↪ ]);
939             $failedPositions[$trainPositionChangeKey] = $trainPositionChange[
                  ↪ $trainPositionChangeKey];
940           }
941           break;
942         } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
943           if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
944             if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
                  ↪ $cumulativeSectionLengthStartKey]) {
945               array_push($failedSections, $cumulativeSectionLengthStartKey);
946               array_push($failedSpeeds, $trainSpeedChange[
                    ↪ $trainPositionChangeKey]);
947               $failedPositions[$trainPositionChangeKey] = $trainPositionChange[
                    ↪ $trainPositionChangeKey];
948             }
949             break;
950           }
951         }
952       }
953     }
954   }
955
956   // Alle Infra_abschnitte zwischen denn beiden KeyPoints, bei denen die
```

```
957   // zulässige Höchstgeschwindigkeit überschritten wird
958   $failedSections = array_unique($failedSections);
959
960   // Wenn es kein Fehler gibt, werden die beiden KeyPoints zurückgegeben und
961   // wen es einen Fehler gibt, wird der erste der beiden KeyPoints im
962   // $returnKeyPoints gespeichert
963   if (sizeof($failedSections) == 0) {
964     return array($temKeyPoints[$keyPointIndex], $temKeyPoints[$keyPointIndex +
          ↪ 1]);
965   } else {
966     $returnKeyPoints[0]["speed_0"] = $temKeyPoints[$keyPointIndex]["speed_0"];
967     $returnKeyPoints[0]["position_0"] = $temKeyPoints[$keyPointIndex]["
          ↪ position_0"];
968   }
969
970   // Einteilung der benachbarten failedSections in zusammenhängende Gruppen
971   $previous = NULL;
972   $index = 0;
973   foreach($failedSections as $key => $value) {
974     if($value > $previous + 1) {
975       $index++;
976     }
977     $groupedFailedSections[$index][] = $value;
978     $previous = $value;
979   }
980
981   // Iteration über die zusammenhängenden $failedSections
982   foreach ($groupedFailedSections as $groupSectionsIndex => $groupSectionsValue
          ↪ ) {
983     $firstFailedPositionIndex = null;
984     $lastFailedPositionIndex = null;
985     $firstFailedPosition = null;
986     $lastFailedPosition = null;
987     $lastElement = array_key_last($returnKeyPoints);
988     $failedSection = null;
989
990     // Ermittlung der Section mit der kleinsten v_max von allen $failedSections
991     // in der Gruppe
992     if (sizeof($groupSectionsValue) == 1) {
```

```
993       $failedSection = $groupSectionsValue[0];
994     } else {
995       $slowestSpeed = 200;
996       for ($i = 0; $i <= (sizeof($groupSectionsValue) - 1); $i++) {
997         if ($next_v_max_mod[$groupSectionsValue[$i]] < $slowestSpeed) {
998           $slowestSpeed = $next_v_max_mod[$groupSectionsValue[$i]];
999           $failedSection = $groupSectionsValue[$i];
1000        }
1001      }
1002    }
1003
1004    // Start- und Endposition der $failedSection
1005    $failedSectionStart = $cumulativeSectionLengthStartMod[$failedSection];
1006    $failedSectionEnd = $cumulativeSectionLengthEndMod[$failedSection];
1007
1008    // Bestimmung der ersten und letzten Position, in der es in der
           ↪ $failedSection
1009    // zu einer Geschwindigkeitsüberschreitung kommt
1010    foreach ($failedPositions as $failPositionIndex => $failPositionValue) {
1011      if ($failPositionValue > $failedSectionStart && $failPositionValue <
           ↪ $failedSectionEnd) {
1012        if ($firstFailedPositionIndex == null) {
1013          $firstFailedPositionIndex = $failPositionIndex;
1014        }
1015        $lastFailedPositionIndex = $failPositionIndex;
1016      }
1017    }
1018
1019    // Bestimmung des letzten Punktes, bei dem die Geschwindigkeit noch
1020    // nicht zu schnell war
1021    //
1022    // Wenn der Punkt davor außerhalb der failedSection liegt
1023    // => Startpunkt = Anfang der Section
1024    // Wenn der Punkt davor innerhalb der failed Section liegt
1025    // => Startpunkt = der Punkt davor
1026    if ($firstFailedPositionIndex != 0) {
1027      if ($trainPositionChange[$firstFailedPositionIndex - 1] <
           ↪ $failedSectionStart) {
1028        $firstFailedPosition = $failedSectionStart;
```

```
1029      } else {
1030        $firstFailedPosition = $trainPositionChange[$firstFailedPositionIndex -
              ↪ 1];
1031      }
1032    } else {
1033      $firstFailedPosition = $failedSectionStart;
1034    }
1035
1036    // Bestimmung der ersten Position, bei dem die Geschwindigkeit nicht
1037    // mehr zu hoch war.
1038    if ($lastFailedPositionIndex != array_key_last($trainPositionChange)) {
1039      if ($trainPositionChange[$lastFailedPositionIndex + 1] >
            ↪ $failedSectionEnd) {
1040        $lastFailedPosition = $failedSectionEnd;
1041      } else {
1042        $lastFailedPosition = $trainPositionChange[$lastFailedPositionIndex +
              ↪ 1];
1043      }
1044    } else {
1045      $lastFailedPosition = $failedSectionEnd;
1046    }
1047
1048    $returnKeyPoints[$lastElement + 1]["position_1"] = $firstFailedPosition;
1049    $returnKeyPoints[$lastElement + 1]["speed_1"] = $next_v_max_mod[
          ↪ $failedSection];
1050    $returnKeyPoints[$lastElement + 2]["position_0"] = $lastFailedPosition;
1051    $returnKeyPoints[$lastElement + 2]["speed_0"] = $next_v_max_mod[
          ↪ $failedSection];
1052  }
1053
1054  // Zielwerte des letzten $keyPoint vom zweiten $keyPoint übernehmen
1055  $returnKeyPoints[array_key_last($returnKeyPoints) + 1]["position_1"] =
        ↪ $temKeyPoints[$keyPointIndex + 1]["position_1"];
1056  $returnKeyPoints[array_key_last($returnKeyPoints)]["speed_1"] = $temKeyPoints
        ↪ [$keyPointIndex + 1]["speed_1"];
1057  $numberOfPairs = sizeof($returnKeyPoints) / 2;
1058
1059  for($j = 0; $j < $numberOfPairs; $j++) {
1060    $i = $j * 2;
```

```
1061      $distance = $returnKeyPoints[$i + 1]["position_1"] - $returnKeyPoints[$i]["
              ↪ position_0"];
1062      $vMax = getVMaxBetweenTwoPoints($distance, $returnKeyPoints[$i]["speed_0"],
              ↪  $returnKeyPoints[$i + 1]["speed_1"]);
1063      // TODO: Der Teil kann weg, getVMaxBetweenTwoPoints() kann nicht -10 zurü
              ↪ ckgeben
1064      /*
1065      if ($vMax == -10) {
1066        $returnKeyPoints[$i]["position_0"] = $returnKeyPoints[$i + 1]["position_1
              ↪ "] - (getBrakeDistance($returnKeyPoints[$i]["speed_0"],
              ↪ $returnKeyPoints[$i + 1]["speed_1"], $verzoegerung));
1067        $distance = $returnKeyPoints[$i + 1]["position_1"] - $returnKeyPoints[$i
              ↪ ]["position_0"];
1068        $vMax = getVMaxBetweenTwoPoints($distance, $returnKeyPoints[$i]["speed_0
              ↪ "], $returnKeyPoints[$i + 1]["speed_1"]);
1069      }
1070      */
1071      $returnKeyPoints[$i]["speed_1"] = $vMax; //TODO
1072      $returnKeyPoints[$i]["position_1"] = $returnKeyPoints[$i]["position_0"] +
              ↪ getBrakeDistance($returnKeyPoints[$i]["speed_0"], $vMax,
              ↪ $verzoegerung);
1073      $returnKeyPoints[$i + 1]["speed_0"] = $vMax;
1074      $returnKeyPoints[$i + 1]["position_0"] = $returnKeyPoints[$i + 1]["
              ↪ position_1"] - getBrakeDistance($vMax, $returnKeyPoints[$i + 1]["
              ↪ speed_1"], $verzoegerung);
1075    }
1076
1077    return $returnKeyPoints;
1078  }
1079
1080  // Wenn ein Key Point beschleunigt und der nächste Key Point abbremst, wird
1081  // die Geschwindigkeit zwischen den beiden KeyPoints als $v_maxBetweenKeyPoints
1082  // gespeichert und als $v_minBetweenKeyPoints der größere Wert von
1083  // $keyPoints[$i]["speed_0"] und $keyPoints[$i + 1]["speed_1"]
1084  function checkIfTheSpeedCanBeDecreased() {
1085
1086    global $keyPoints;
1087    global $returnPossibleSpeed;
1088
```

```php
  $returnPossibleSpeed = array();

  for ($i = 0; $i < (sizeof($keyPoints) - 1); $i++) {
    $v_maxBetweenKeyPoints = $keyPoints[$i]["speed_1"];
    $v_minBetweenKeyPoints = null;

    if ($keyPoints[$i]["speed_0"] < $v_maxBetweenKeyPoints && $keyPoints[$i +
        ↪ 1]["speed_1"] < $v_maxBetweenKeyPoints) {
      $v_minBetweenKeyPoints = $keyPoints[$i]["speed_0"];
      if ($keyPoints[$i + 1]["speed_1"] > $v_minBetweenKeyPoints) {
        $v_minBetweenKeyPoints = $keyPoints[$i + 1]["speed_1"];
      }
    }

    if (isset($v_minBetweenKeyPoints)) {
      if ($v_minBetweenKeyPoints == 0 && $v_maxBetweenKeyPoints >= 10) {
        $v_minBetweenKeyPoints = 10;
      } else if ($v_minBetweenKeyPoints == 0 && $v_maxBetweenKeyPoints == 10) {
        $v_minBetweenKeyPoints = null;
      }
    }

    if ($v_minBetweenKeyPoints != null) {
      if ($v_minBetweenKeyPoints % 10 != 0) {
        $rest = $v_minBetweenKeyPoints % 10;
        $v_minBetweenKeyPoints = $v_minBetweenKeyPoints - $rest + 10;
      }

      array_push($returnPossibleSpeed, array("KeyPoint_index" => $i, "values"
          ↪ => range($v_minBetweenKeyPoints, $v_maxBetweenKeyPoints, 10)));
    }
  }
  if (sizeof($returnPossibleSpeed) > 0) {
    return array("possible" => true, "range" => $returnPossibleSpeed);
  } else {
    return array("possible" => false, "range" => array());
  }
}

```

```
1126 // Wenn in 'global_variables.php' der Variablen $useSpeedFineTuning der Wert
1127 // 'true' zugewiesen ist und das Fahrzeug zu früh an der nächsten
         ↪ Betriebsstelle
1128 // ankommt, wird überprüft, ob durch eine vorzeitige Einleitung einer Verzö
         ↪ gerung
1129 // die exakte Ankunftszeit eingehalten werden kann.
1130 function speedFineTuning(float $timeDiff, int $index) {
1131
1132   global $keyPoints;
1133   global $verzoegerung;
1134   global $globalTimeOnOneSpeed;
1135   global $useMinTimeOnSpeed;
1136
1137   $speed_0 = $keyPoints[$index]["speed_1"];
1138   $speed_1 = null;
1139   $availableDistance = $keyPoints[$index + 1]["position_0"] - $keyPoints[$index
         ↪ ]["position_1"];
1140   $timeBetweenKeyPoints = $keyPoints[$index + 1]["time_0"] - $keyPoints[$index
         ↪ ]["time_1"];
1141   $availableTime = $timeBetweenKeyPoints + $timeDiff;
1142
1143   if ($keyPoints[$index + 1]["speed_1"] != 0) {
1144     $speed_1 = $keyPoints[$index + 1]["speed_1"];
1145     $lengthDifference = calculateDistanceforSpeedFineTuning($keyPoints[$index +
         ↪  1]["speed_0"], $keyPoints[$index + 1]["speed_1"],
         ↪ $availableDistance, $availableTime);
1146
1147     if ($useMinTimeOnSpeed) {
1148       if (distanceWithSpeedToTime($speed_0, $availableDistance -
           ↪ $lengthDifference) > $globalTimeOnOneSpeed &&
           ↪ distanceWithSpeedToTime($speed_1, $lengthDifference) >
           ↪ $globalTimeOnOneSpeed) {
1149         $keyPoints[$index + 1]["position_0"] = $keyPoints[$index + 1]["
             ↪ position_0"] - $lengthDifference;
1150         $keyPoints[$index + 1]["position_1"] = $keyPoints[$index + 1]["
             ↪ position_1"] - $lengthDifference;
1151       }
1152     } else {
```

```
1153      $keyPoints[$index + 1]["position_0"] = $keyPoints[$index + 1]["position_0
              ↪ "] - $lengthDifference;
1154      $keyPoints[$index + 1]["position_1"] = $keyPoints[$index + 1]["position_1
              ↪ "] - $lengthDifference;
1155    }
1156  } else if ($keyPoints[$index + 1]["speed_0"] > 10) {
1157    $speed_1 = 10;
1158    $lengthDifference = calculateDistanceforSpeedFineTuning($keyPoints[$index +
          ↪  1]["speed_0"],10, $availableDistance, $availableTime);
1159
1160    if ($useMinTimeOnSpeed) {
1161      if (distanceWithSpeedToTime($speed_0, $availableDistance -
              ↪ $lengthDifference) > $globalTimeOnOneSpeed &&
              ↪ distanceWithSpeedToTime($speed_1, $lengthDifference) >
              ↪ $globalTimeOnOneSpeed) {
1162        $firstKeyPoint = createKeyPoint(($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);
1163        $secondKeyPoint = createKeyPoint(($keyPoints[$index + 1]["position_1"] -
                ↪  getBrakeDistance(10, 0, $verzoegerung)),$keyPoints[$index + 1]["
                ↪ position_1"],10,$keyPoints[$index + 1]["speed_1"]);
1164        $keyPoints[$index + 1] = $secondKeyPoint;
1165        array_splice( $keyPoints, ($index + 1), 0, array($firstKeyPoint));
1166      }
1167    } else {
1168      $firstKeyPoint = createKeyPoint(($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)
              ↪ ;
1169      $secondKeyPoint = createKeyPoint(($keyPoints[$index + 1]["position_1"] -
              ↪ getBrakeDistance(10, 0, $verzoegerung)),$keyPoints[$index + 1]["
              ↪ position_1"],10,$keyPoints[$index + 1]["speed_1"]);
1170      $keyPoints[$index + 1] = $secondKeyPoint;
1171      array_splice( $keyPoints, ($index + 1), 0, array($firstKeyPoint));
1172    }
1173  }
```

```
1174 }
1175
1176 // Sucht den KeyPoint der zu maximalen Geschwindigkeit beschleunigt
1177 // Wenn die maximale Geschwindigkeit mehrfach erreciht wird, wird
1178 // der letzte dieser KeyPoints genommen
1179 //
1180 // Zu dem Index wird auch noch die Speed Range abgespeichert wie bei
1181 // checkIfTheSpeedCanBeDecreased()
1182 // TODO: Kann man diese beiden Funktionen kombinieren?
1183 function findMaxSpeed(array $speedDecrease) {
1184   $maxSpeed = 0;
1185   $minSpeed = 0;
1186   $keyPointIndex = null;
1187
1188   for ($i = 0; $i < sizeof($speedDecrease["range"]); $i++) {
1189     if (max($speedDecrease["range"][$i]["values"]) >= $maxSpeed) {
1190       $maxSpeed = max($speedDecrease["range"][$i]["values"]);
1191       $minSpeed = min($speedDecrease["range"][$i]["values"]);
1192       $keyPointIndex = $speedDecrease["range"][$i]["KeyPoint_index"];
1193     }
1194   }
1195   return array("min_speed" => $minSpeed, "max_speed" => $maxSpeed, "
       ↪ first_key_point_index" => $keyPointIndex);
1196 }
1197
1198 // Überprüft beim Start der Fahrtverlaufsberechnung, ob es möglich ist einen
    ↪ Fahrtverlauf zu ermitteln
1199 function checkIfItsPossible() {
1200
1201   global $currentSpeed;
1202   global $distanceToNextStop;
1203   global $verzoegerung;
1204   global $globalTimeOnOneSpeed;
1205   global $errorMinTimeOnSpeed;
1206
1207   $minTimeIsPossible = true;
1208
1209   if ($currentSpeed == 0) {
1210     $distance_0 = getBrakeDistance(0, 10, $verzoegerung);
```

```
1211      $distance_1 = getBrakeDistance(10, 0, $verzoegerung);
1212      $time = distanceWithSpeedToTime(10, $distanceToNextStop - $distance_0 -
              ↪ $distance_1);
1213
1214      if ($time < $globalTimeOnOneSpeed) {
1215        $minTimeIsPossible = false;
1216
1217        if ($errorMinTimeOnSpeed) {
1218          // TODO: Notbremsung einleiten/Zug bekommt Fehler
1219        }
1220
1221        echo "Der Zug schafft es ohne Notbremsung am Ziel anzukommen, kann aber
              ↪ nicht die mind. Zeit einhalten.\n";
1222      }
1223    } else {
1224      if (getBrakeDistance($currentSpeed, 0, $verzoegerung) !=
              ↪ $distanceToNextStop) {
1225        $distance_0 = getBrakeDistance($currentSpeed, 10, $verzoegerung);
1226        $distance_1 = getBrakeDistance(10, 0, $verzoegerung);
1227        $time = distanceWithSpeedToTime(10, $distanceToNextStop - $distance_0 -
              ↪ $distance_1);
1228
1229        if ($time < $globalTimeOnOneSpeed) {
1230          $minTimeIsPossible = false;
1231
1232          if ($errorMinTimeOnSpeed) {
1233            // TODO: Notbremsung einleiten/Zug bekommt Fehler
1234          }
1235
1236          echo "Der Zug schafft es, ohne Notbremsung am Ziel anzukommen.\n";
1237        }
1238      }
1239    }
1240    return $minTimeIsPossible;
1241  }
1242
1243  // Überprüft, ob der vorgeschriebene Wert aus der Variablen
       ↪ $globalTimeOnOneSpeed
1244  // eingehalten wird, falls die Variable $useMinTimeOnSpeed den Wert 'true' hat
```

```
1245 function toShortOnOneSpeed () {
1246
1247   global $keyPoints;
1248   global $verzoegerung;
1249
1250   $localKeyPoints = $keyPoints;
1251   $subsections = createSubsections($localKeyPoints);
1252
1253   while (toShortInSubsection($subsections)) {
1254     $breakesOnly = true;
1255     foreach ($subsections as $sectionKey => $sectionValue) {
1256       if ($sectionValue["failed"]) {
1257         if (!$sectionValue["brakes_only"]) {
1258           $breakesOnly = false;
1259         }
1260
1261         $return = postponeSubsection($localKeyPoints, $sectionValue);
1262
1263         if (!$return["fail"]) {
1264           $localKeyPoints = $return["keyPoints"];
1265         } else {
1266           if (!$sectionValue["brakes_only"]) {
1267             $localKeyPoints[$sectionValue["max_index"]]["speed_1"] -= 10;
1268             $localKeyPoints[$sectionValue["max_index"] + 1]["speed_0"] -= 10;
1269             $localKeyPoints[$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);
1270             $localKeyPoints[$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);
1271             $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1272             $localKeyPoints = deleteDoubledKeyPoints($localKeyPoints);
1273             break;
1274           }
1275         }
```

```
1276        }
1277      }
1278      $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1279      $localKeyPoints = array_values($localKeyPoints);
1280      $subsections = createSubsections($localKeyPoints);
1281
1282      if ($breakesOnly) {
1283        break;
1284      }
1285    }
1286    $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1287    $keyPoints = $localKeyPoints;
1288  }
1289
1290  // Überprüft, ob innerhalb einer $subsection Beschleuniguns- und Bremsvorgänge
       ↪ später bzw. früher eingeleitet werden können.
1291  function postponeSubsection (array $localKeyPoints, array $subsection) {
1292
1293    global $globalTimeOnOneSpeed;
1294    global $verzoegerung;
1295
1296    $deletedKeyPoints = array();
1297    $numberOfKeyPoints = sizeof($subsection["indexes"]);
1298    $indexMaxSection = array_search($subsection["max_index"], $subsection["
       ↪ indexes"]);
1299    $indexLastKeyPoint = array_key_last($subsection["indexes"]);
1300
1301    if ($subsection["is_prev_section"]) {
1302      $timeDiff = $localKeyPoints[$subsection["indexes"][0]]["time_0"] -
         ↪ $localKeyPoints[$subsection["indexes"][0] - 1]["time_1"] -
         ↪ $globalTimeOnOneSpeed;
1303      if ($timeDiff < 0) {
1304        $positionDiff = abs($timeDiff) * $localKeyPoints[$subsection["indexes"
           ↪ ][0]]["speed_0"] / 3.6;
1305        if (!($localKeyPoints[$subsection["indexes"][0]]["position_1"] +
           ↪ $positionDiff > $localKeyPoints[$subsection["indexes"][
           ↪ $indexMaxSection + 1]]["position_0"])) {
1306          $localKeyPoints[$subsection["indexes"][0]]["position_0"] +=
             ↪ $positionDiff;
```

```
1307        $localKeyPoints[$subsection["indexes"][0]]["position_1"] +=
                ↪ $positionDiff;
1308        if ($localKeyPoints[$subsection["indexes"][0]]["position_1"] >
                ↪ $localKeyPoints[$subsection["indexes"][0] + 1]["position_0"]) {
1309          array_push($deletedKeyPoints, $subsection["indexes"][0] + 1);
1310          $numberOfKeyPoints -= 1;
1311          $v_0 = $localKeyPoints[$subsection["indexes"][0]]["speed_0"];
1312          $v_1 = $localKeyPoints[$subsection["indexes"][0] + 1]["speed_1"];
1313          $localKeyPoints[$subsection["indexes"][0]]["position_1"] =
                  ↪ $localKeyPoints[$subsection["indexes"][0]]["position_0"] +
                  ↪ getBrakeDistance($v_0, $v_1, $verzoegerung);
1314          $localKeyPoints[$subsection["indexes"][0]]["speed_1"] = $v_1;
1315        }
1316        $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
                ↪ $deletedKeyPoints);
1317      }
1318    }
1319  }
1320
1321  for ($i = 1; $i <= $indexMaxSection; $i++) {
1322    if (!in_array($subsection["indexes"][$i], $deletedKeyPoints)) {
1323      $timeDiff = $localKeyPoints[$subsection["indexes"][$i]]["time_0"] -
              ↪ $localKeyPoints[$subsection["indexes"][$i] - 1]["time_1"] -
              ↪ $globalTimeOnOneSpeed;
1324      if ($timeDiff < 0) {
1325        $positionDiff = abs($timeDiff) * $localKeyPoints[$subsection["indexes"][
              ↪ $i]]["speed_0"] / 3.6;
1326        if (!($localKeyPoints[$subsection["indexes"][$i]]["position_1"] +
              ↪ $positionDiff > $localKeyPoints[$subsection["indexes"][
              ↪ $indexMaxSection + 1]]["position_0"])) {
1327          $localKeyPoints[$subsection["indexes"][$i]]["position_0"] +=
                  ↪ $positionDiff;
1328          $localKeyPoints[$subsection["indexes"][$i]]["position_1"] +=
                  ↪ $positionDiff;
1329          if ($i < $indexMaxSection && $localKeyPoints[$subsection["indexes"][$i
                  ↪ ]]["position_1"] > $localKeyPoints[$subsection["indexes"][$i] +
                  ↪  1]["position_0"]) {
1330            array_push($deletedKeyPoints, ($subsection["indexes"][$i] + 1));
1331            $numberOfKeyPoints -= 1;
```

```
1332            $v_0 = $localKeyPoints[$subsection["indexes"][$i]]["speed_0"];
1333            $v_1 = $localKeyPoints[$subsection["indexes"][$i] + 1]["speed_1"];
1334            $localKeyPoints[$subsection["indexes"][$i]]["position_1"] =
                    ↪ $localKeyPoints[$subsection["indexes"][$i]]["position_0"] +
                    ↪ getBrakeDistance($v_0, $v_1, $verzoegerung);
1335            $localKeyPoints[$subsection["indexes"][$i]]["speed_1"] = $v_1;
1336          }
1337          $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
                  ↪ $deletedKeyPoints);
1338        }
1339      }
1340    }
1341  }
1342
1343  if ($subsection["is_next_section"]) {
1344    $timeDiff = $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint] +
        ↪ 1]["time_0"] - $localKeyPoints[$subsection["indexes"][
        ↪ $indexLastKeyPoint]]["time_1"] - $globalTimeOnOneSpeed;
1345    if ($timeDiff < 0) {
1346      $positionDiff = abs($timeDiff) * $localKeyPoints[$indexLastKeyPoint]["
          ↪ speed_1"] / 3.6;
1347      if (!($localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["
          ↪ position_0"] - $positionDiff < $localKeyPoints[$subsection["indexes
          ↪ "][$indexMaxSection]]["position_0"])) {
1348        $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["position_0"
            ↪ ] -= $positionDiff;
1349        $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["position_1"
            ↪ ] -= $positionDiff;
1350        if ($localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["
            ↪ position_0"] < $localKeyPoints[$subsection["indexes"][
            ↪ $indexLastKeyPoint] - 1]["position_1"]) {
1351          array_push($deletedKeyPoints, ($subsection["indexes"][
              ↪ $indexLastKeyPoint] - 1));
1352          $numberOfKeyPoints -= 1;
1353          $v_0 = $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint] -
              ↪ 1]["speed_0"];
1354          $v_1 = $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["
              ↪ speed_1"];
```

```
1355          $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["
                  ↪ position_0"] = $localKeyPoints[$subsection["indexes"][
                  ↪ $indexLastKeyPoint]]["position_1"] - getBrakeDistance($v_0,
                  ↪ $v_1, $verzoegerung);
1356          $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["speed_0"]
                  ↪  = $v_0;
1357        }
1358        $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
                ↪ $deletedKeyPoints);
1359      }
1360    }
1361  }
1362
1363  for ($i = $indexLastKeyPoint - 1; $i > $indexMaxSection; $i--) {
1364    if (!in_array($i, $deletedKeyPoints)) {
1365      $timeDiff = $localKeyPoints[$subsection["indexes"][$i + 1]]["time_0"] -
              ↪ $localKeyPoints[$subsection["indexes"][$i]]["time_1"] -
              ↪ $globalTimeOnOneSpeed;
1366      if ($timeDiff < 0) {
1367        $positionDiff = abs($timeDiff) * $localKeyPoints[$indexLastKeyPoint]["
                ↪ speed_1"] / 3.6;
1368        if (!($localKeyPoints[$subsection["indexes"][$i]]["position_0"] -
                ↪ $positionDiff < $localKeyPoints[$subsection["indexes"][
                ↪ $indexMaxSection]]["position_0"])) {
1369          $localKeyPoints[$subsection["indexes"][$i]]["position_0"] -=
                  ↪ $positionDiff;
1370          $localKeyPoints[$subsection["indexes"][$i]]["position_1"] -=
                  ↪ $positionDiff;
1371          if ($i > ($indexMaxSection + 1) && $localKeyPoints[$subsection["
                  ↪ indexes"][$i]]["position_0"] < $localKeyPoints[$subsection["
                  ↪ indexes"][$i] - 1]["position_1"]) {
1372            array_push($deletedKeyPoints, ($subsection["indexes"][$i] - 1));
1373            $numberOfKeyPoints -= 1;
1374            $v_0 = $localKeyPoints[$subsection["indexes"][$i] - 1]["speed_0"];
1375            $v_1 = $localKeyPoints[$subsection["indexes"][$i]]["speed_1"];
1376            $localKeyPoints[$subsection["indexes"][$i]]["position_0"] =
                    ↪ $localKeyPoints[$subsection["indexes"][$i]]["position_1"] -
                    ↪ getBrakeDistance($v_0, $v_1, $verzoegerung);
1377            $localKeyPoints[$subsection["indexes"][$i]]["speed_0"] = $v_0;
```

```
        }
        $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
            ↪ $deletedKeyPoints);
      }
    }
  }
}

$keys = $subsection["indexes"];

foreach ($deletedKeyPoints as $index) {
  unset($keys[array_search($index, $keys)]);
}

// Ordnet die Abschnitte zwischen zwei $subsection den $subsections zu
$keys = array_values($keys);
$failed = false;

if ($subsection["is_prev_section"]) {
  if ($localKeyPoints[$keys[0]]["time_0"] - $localKeyPoints[$keys[0] - 1]["
      ↪ time_1"] < $globalTimeOnOneSpeed) {
    $failed = true;
  }
}

if ($subsection["is_next_section"]) {
  if ($localKeyPoints[end($keys) + 1]["time_0"] - $localKeyPoints[end($keys)
      ↪ ]["time_1"] < $globalTimeOnOneSpeed) {
    $failed = true;
  }
}

for ($i = 1; $i < sizeof($keys); $i++) {
  if ($localKeyPoints[$keys[$i]]["time_0"] - $localKeyPoints[$keys[$i - 1]]["
      ↪ time_1"] < $globalTimeOnOneSpeed) {
    $failed = true;
    break;
  }
}
```

```
1413
1414   if ($failed) {
1415     return array("fail" => true, "keyPoints" => array());
1416   } else {
1417     foreach ($deletedKeyPoints as $index) {
1418       unset($localKeyPoints[$index]);
1419     }
1420
1421     return array("fail" => false, "keyPoints" => $localKeyPoints);
1422   }
1423 }
1424
1425 // Erstellt mittels der $keyPoints die $subsections
1426 function createSubsections (array $localKeyPoints) {
1427
1428   global $globalTimeOnOneSpeed;
1429
1430   $keyPoints = $localKeyPoints;
1431   $subsections = array();
1432   $subsection = array("max_index" => null, "indexes" => array(), "
          ↪ is_prev_section" => false, "is_next_section" => false);
1433   $maxIndex = null;
1434
1435   // Wenn die erste Geschwindigkeit die maximale Geschwindigkeit der ersten
          ↪ Subsection ist.
1436   // TODO: Bei v_0 != 0 hat der erste KeyPoint v_0 == v_1... Kommt es da zu
          ↪ Konflikten?
1437   for($i = 0; $i < sizeof($keyPoints); $i++) {
1438     // subsection zu ende
1439     if ($i > 0) {
1440       if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"] && $keyPoints[
              ↪ $i - 1]["speed_0"] > $keyPoints[$i - 1]["speed_1"] || $i == sizeof(
              ↪ $keyPoints) - 1) {
1441         if ($i == sizeof($keyPoints) - 1) {
1442           array_push($subsection["indexes"], $i);
1443         }
1444
1445         array_push($subsections, $subsection);
1446         $subsection["indexes"] = array();
```

```
      }
    }

    if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
      $subsection["max_index"] = $i;
    }
    array_push($subsection["indexes"], $i);
  }
  // Check if middle section failed
  for ($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;
```

```
1483        }
1484
1485        if (!$isLastSubsection) {
1486          if (!$subsections[$i + 1]["is_prev_section"]) {
1487            $subsections[$i]["is_next_section"] = true;
1488          }
1489        }
1490      } else {
1491        $subsections[$i]["failed"] = false;
1492      }
1493    }
1494
1495    for ($i = 0; $i < sizeof($subsections); $i++) {
1496      if (!isset($subsections[$i]["max_index"])) {
1497        $subsections[$i]["brakes_only"] = true;
1498        $subsections[$i]["max_index"] = $subsections[$i]["indexes"][0];
1499      } else {
1500        $subsections[$i]["brakes_only"] = false;
1501      }
1502    }
1503
1504    $subsections = array_values($subsections);
1505
1506    return array_reverse($subsections);
1507  }
1508
1509  // Überprüfung, ob es in einer $subsection zu einer Unterschreitung
1510  // der Mindestzeit kommt
1511  function failOnSubsection(array $keyPoints, array $subsection) {
1512
1513    global $globalTimeOnOneSpeed;
1514
1515    $failed = false;
1516
1517    for ($i = 1; $i < sizeof($subsection["indexes"]); $i++) {
1518      if ($keyPoints[$subsection["indexes"][$i]]["time_0"] - $keyPoints[
         ↪ $subsection["indexes"][$i] - 1]["time_1"] < $globalTimeOnOneSpeed) {
1519        $failed = true;
1520        break;
```

```
1521      }
1522    }
1523
1524    return $failed;
1525  }
1526
1527  /*
1528  function checkForPostponement(array $localKeyPoints, array $subsection) {
1529    global $globalTimeOnOneSpeed;
1530
1531    $keyPoints = $localKeyPoints;
1532    $timeBeforeMax = 0;
1533    $timeAfterMax = 0;
1534    $foundShortSectionBeforeMax = false;
1535    $foundShortSectionAfterMax = false;
1536    $indexMaxSection = array_search($subsection["max_index"], $subsection["
          ↪ indexes"]);
1537    $indexLastKeyPoint = array_key_last($subsection["indexes"]);
1538    $timeOnMax = $keyPoints[$subsection["max_index"] + 1]["time_0"] - $keyPoints[
          ↪ $subsection["max_index"]]["time_1"] - $globalTimeOnOneSpeed;
1539    if ($timeOnMax < 0) {
1540      return false;
1541    }
1542    if ($subsection["is_prev_section"]) {
1543      $timeDiff = $keyPoints[$subsection["indexes"][0]]["time_0"] - $keyPoints[
            ↪ $subsection["indexes"][0] - 1]["time_1"] - $globalTimeOnOneSpeed;
1544      if ($timeDiff < 0) {
1545        $timeBeforeMax += $timeDiff;
1546        $foundShortSectionBeforeMax = true;
1547      }
1548    }
1549
1550    if ($subsection["is_next_section"]) {
1551      $timeDiff = $keyPoints[$subsection["indexes"][array_key_last($subsection["
            ↪ indexes"])] + 1]["time_0"] - $keyPoints[$subsection["indexes"][
            ↪ array_key_last($subsection["indexes"])]]["time_1"] -
            ↪ $globalTimeOnOneSpeed;
1552      if ($timeDiff < 0) {
1553        $timeAfterMax += $timeDiff;
```

```
1554      $foundShortSectionAfterMax = true;
1555    }
1556  }
1557
1558  for ($i = 1; $i <= $indexMaxSection; $i++) {
1559    if ($keyPoints[$subsection["indexes"][$i]]["time_0"] - $keyPoints[
          ↪ $subsection["indexes"][$i] - 1]["time_1"] < $globalTimeOnOneSpeed ||
          ↪  $foundShortSectionBeforeMax) {
1560      $foundShortSectionBeforeMax = true;
1561      $timeBeforeMax += $keyPoints[$subsection["indexes"][$i]]["time_0"] -
          ↪ $keyPoints[$subsection["indexes"][$i] - 1]["time_1"] -
          ↪ $globalTimeOnOneSpeed;
1562    }
1563  }
1564
1565  for ($i = $indexLastKeyPoint; $i > $indexMaxSection + 1; $i--) {
1566    if ($keyPoints[$subsection["indexes"][$i]]["time_0"] - $keyPoints[
          ↪ $subsection["indexes"][$i] - 1]["time_1"] < $globalTimeOnOneSpeed ||
          ↪  $foundShortSectionAfterMax) {
1567      $foundShortSectionAfterMax = true;
1568      $timeAfterMax += $keyPoints[$subsection["indexes"][$i]]["time_0"] -
          ↪ $keyPoints[$subsection["indexes"][$i] - 1]["time_1"] -
          ↪ $globalTimeOnOneSpeed;
1569    }
1570  }
1571
1572  if ($timeBeforeMax > 0) {
1573    $timeBeforeMax = 0;
1574  }
1575
1576  if ($timeAfterMax > 0) {
1577    $timeAfterMax = 0;
1578  }
1579
1580  // true = kann verschoben werden...
1581  if ($timeOnMax + $timeBeforeMax + $timeAfterMax >= 0) {
1582    return true;
1583  } else {
1584    return false;
```

```
1585    }
1586  }
1587  */
1588
1589  function toShortInSubsection (array $subsections) {
1590
1591    $foundError = false;
1592
1593    foreach ($subsections as $subsection) {
1594      if ($subsection["failed"]) {
1595        $foundError = true;
1596        break;
1597      }
1598    }
1599
1600    return $foundError;
1601  }
1602
1603  function createCumulativeSections ($indexCurrentSection, $indexTargetSection,
       ↪ $currentPosition, $targetPosition, $next_lengths) {
1604
1605    $cumLength = array();
1606    $sum = 0;
1607
1608    foreach ($next_lengths as $index => $value) {
1609      if ($index >= $indexCurrentSection) {
1610        $sum += $value;
1611        $cumLength[$index] = $sum;
1612      }
1613    }
1614    // Berechnung der kummulierten Start- und Endlängen der Abschnitte
1615    // TODO: Geht das auch, wenn Start- und Zielabschnitt der selbe sind?
1616    for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
1617      if ($indexCurrentSection == $indexTargetSection) {
1618        $cumulativeSectionLengthStart[$i] = 0;
1619        $cumulativeSectionLengthEnd[$i] = $targetPosition - $currentPosition;
1620      } else {
1621        if ($i == $indexCurrentSection) {
1622          $cumulativeSectionLengthStart[$i] = 0;
```

```
1623          $cumulativeSectionLengthEnd[$i] = $cumLength[$i] - $currentPosition;
1624        } else if ($i == $indexTargetSection) {
1625          $cumulativeSectionLengthStart[$i] = $cumLength[$i - 1] -
                  ↪ $currentPosition;
1626          $cumulativeSectionLengthEnd[$i] = $cumLength[$i - 1] + $targetPosition -
                  ↪  $currentPosition;
1627        } else {
1628          $cumulativeSectionLengthStart[$i] = $cumLength[$i - 1] -
                  ↪ $currentPosition;
1629          $cumulativeSectionLengthEnd[$i] = $cumLength[$i] - $currentPosition;
1630        }
1631      }
1632    }
1633
1634    return array($cumulativeSectionLengthStart, $cumulativeSectionLengthEnd);
1635  }
1636
1637  function toArr(){
1638    return func_get_args();
1639  }
1640
1641  // Ermittelt die Echtzeitdaten für eine Gefahrenbremsung
1642  function emergencyBreak ($id, $distanceToNextStop = 0) {
1643
1644    global $allUsedTrains;
1645    global $timeDifference;
1646    global $allTimes;
1647
1648    $targetSpeed = 0;
1649    $returnArray = array();
1650    $time = microtime(true) + $timeDifference;
1651    $currentSpeed = $allUsedTrains[$id]["current_speed"];
1652    $notverzoegerung = $allUsedTrains[$id]["notverzoegerung"];
1653    $currentSection = $allUsedTrains[$id]["current_section"];
1654
1655    echo "Der Zug mit der Adresse: ", $allUsedTrains[$id]["adresse"], " leitet
          ↪ jetzt eine Notbremsung ein.\n";
1656
```

```
1657  if (getBrakeDistance($currentSpeed, $targetSpeed, $notverzoegerung) <=
          ↪ $distanceToNextStop) {
1658    for ($i = $currentSpeed; $i >= 0; $i = $i - 2) {
1659      array_push($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));
1660      $time = $time + getBrakeTime($i, $i - 1, $notverzoegerung);
1661    }
1662  } else {
1663    $targetSpeedNotbremsung = getTargetBrakeSpeedWithDistanceAndStartSpeed(
          ↪ $distanceToNextStop, $notverzoegerung, $currentSpeed);
1664    $speedBeforeStop = intval($targetSpeedNotbremsung / 2) * 2;
1665
1666    if ($speedBeforeStop >= 10) {
1667      for ($i = $currentSpeed; $i >= 10; $i = $i - 2) {
1668        array_push($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));
1669        $time = $time + getBrakeTime($i, $i - 1, $notverzoegerung);
1670      }
1671      array_push($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" => $id, "wendet" => false, "
          ↪ betriebsstelle" => 'Notbremsung', "live_all_targets_reached" =>
          ↪ null));
1672    } else {
1673      array_push($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));
```

```
1674        $time = $time + getBrakeTime($currentSpeed, $currentSpeed - 1,
                ↪ $notverzoegerung);
1675        array_push($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));
1676      }
1677    }
1678
1679    $allTimes[$allUsedTrains[$id]["adresse"]] = $returnArray;
1680    array_push($allUsedTrains[$id]["error"], 3);
1681
1682    return 0;
1683  }
```

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) : float {
  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

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;
}

function createCacheHaltepunkte() : array{

  $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 .= "%";
    $asig = "ASig";
    $bksig = "BkSig";
```

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

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

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

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

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

A.6 functions_db.php

```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 einen Abschnitt und
      ↪ eine
123 // Richtung.
124 function getSignalForSectionAndDirection(int $section, int $dir) {
125
126   $DB = new DB_MySQL();
127   $signal = $DB->select("SELECT `".DB_TABLE_SIGNALE_STANDORTE."`.`id` FROM `".
          ↪ DB_TABLE_SIGNALE_STANDORTE."` WHERE `".DB_TABLE_SIGNALE_STANDORTE."`.`
          ↪ freimelde_id` = $section AND `".DB_TABLE_SIGNALE_STANDORTE."`.`
          ↪ wirkrichtung` = $dir");
```

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

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

A.7 global_variables.php

```php
<?php

$globalNotverzoegerung = 2; // Bremsverzögerung bei einer Notbremsung
$globalTrainVMax = 10; // Maximale Geschwindigkeit, wenn keine vorgegeben ist
$globalSpeedInCurrentSection = 60; // Maximale Geschwindigkeit im aktuellen
    ↪ Abschnitt
$globalFirstHaltMinTime = 20; // Mindesthaltezeit am ersten fahrplanmäßigen
    ↪ Halt
$globalIndexBetriebsstelleFreieFahrt = 9999999999999;
$globalFloatingPointNumbersRoundingError = 0.0000000001;
$globalTimeOnOneSpeed = 20;
$globalDistanceUpdateInterval = 1;

$useSpeedFineTuning = true;
$useMinTimeOnSpeed = true;
$errorMinTimeOnSpeed = false;
$slowDownIfTooEarly = true;
$useRecalibration = true;
```

A.8 speed_over_position.m

```
%% Load cumulative sections

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);

%% Load modified cumulative sections

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);

%% Load speed over position

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);

%% Load speed over position (all iterationsteps)

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)

% Plot infrastructuresections

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

% Plot modified iterationsteps (incl. trainlength)
line([0 0], [0 vmaxOverPosition_v_max_mod(1)],'Linestyle','-.','LineWidth',2,'color','red', 'HandleVisibility','off');
line([0 vmaxOverPosition_Position_mod(1)], [vmaxOverPosition_v_max_mod(1) vmaxOverPosition_v_max_mod(1)],'Linestyle','-.','LineWidth',2,'color','
```

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

```
%% Format plot

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
```

Literatur

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)

Eidesstattliche Erklärung

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

Ort, Datum Unterschrift