Technische Universitaet Berlin

Bachelorarbeit

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

Friedrich Kasper Völkers

betreut von
Dr.-Ing. Christian Blome

5. 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 Vorgehen 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

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Code-Beispiele

Abkürzungsverzeichnis

PZ Personenzug

GZ Güterzug

EBuEf Eisenbahn-Betriebs- und Experimentierfeld

1 Grundlagen

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

1.1 Aufbau des Eisenbahn-Betriebs- und Experimentierfelds

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

1.2 Worauf in der Arbeit geachtet wird (Proritäten)

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

1.3 Bezug zur realen Welt

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

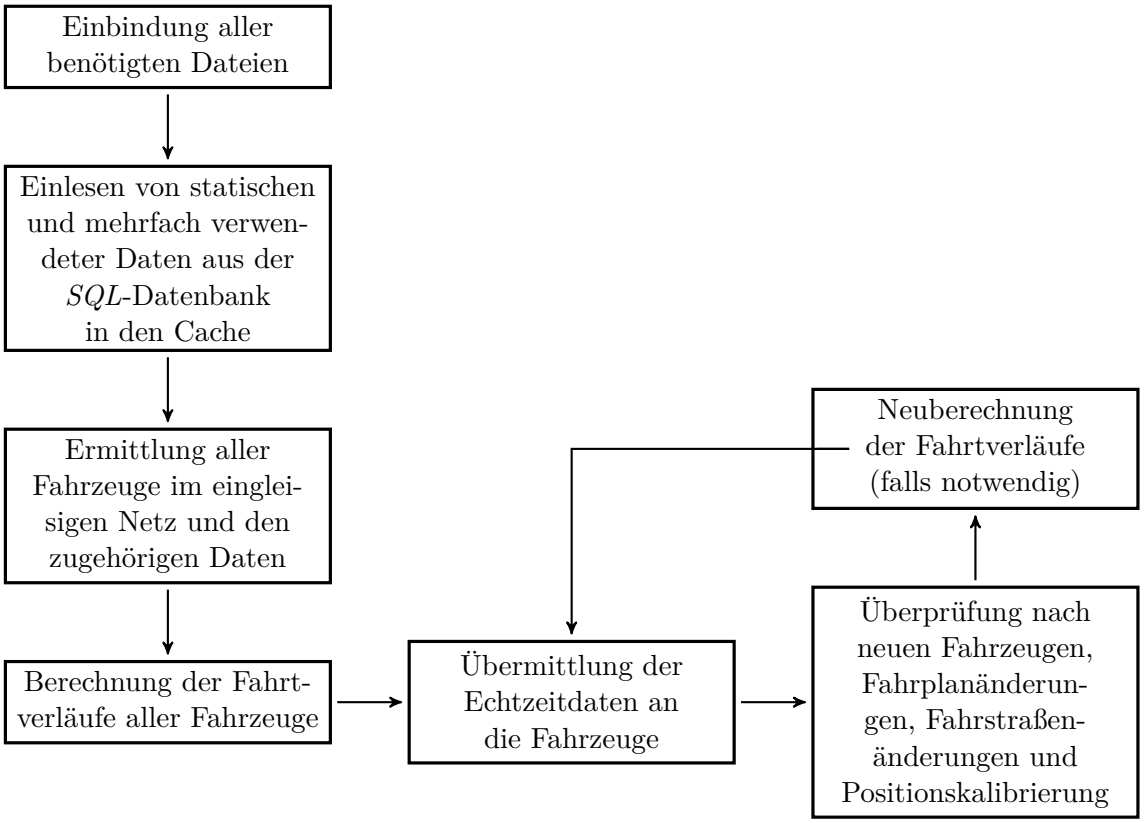

Abbildung 1: Ablauf des Hauptprogramms

2 Ablauf des Hauptprogramms

FAHRTVERLAUF DEFINIEREN/BESCHREIBEN

Um die Fahrzeugsteuerung zu starten, muss die Datei *main.php* ausgeführt werden. Obligatorisch für die Fahrzeugsteuerung ist die Abschnittsüberwachung (*abschnittueberwachung.php*), welche vor dem Start der Fahrzeugsteuerung ausgeführt werden muss. Auf die genaue Verwendung der Abschnittsüberwachung wird in Kapitel 2.6 eingegangen. In der folgenden Abbildung 1 ist der schematische Ablauf des Hauptprogramms abgebildet, welcher auch grob dem Aufbau dieses Kapitel dient.

2.1 Einlesen von statischen und mehrfach verwendeten Daten aus der *SQL*-Datenbank in den Cache

Die Fahrzeugsteuerung benötigt für viele Berechnungen Daten aus der *SQL*-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 Programmstart bzw. bei der ersten Verwendung in den Cache geladen (Code-Beispiel 1). Beispielhaft zu nennen sind hierbei *$cacheInfraLaenge* (Länge

aller Infrastrukturabschnitte in Metern [m]), *$cacheHaltepunkte* (zugehörige Infrastrukturabschnitte für alle Betriebsstellen und Richtung), *$cacheZwischenhaltepunkte* (zugehörige Infrastrukturabschnitte für alle Zwischen-Betriebsstellen, die nur einem Infrastrukturabschnitt zugeordnet sind), *$cacheGbtToInfra* (Zuordnung der Infrastrukturabschnitte zu den GBT-Abschnitten) und *$cacheInfraToGbt* (Zuordnung der GBT-Abschnitte zu den Infrastrukturabschnitten).

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

Code-Beispiel 1: Deklaration und Initialisierung der Cache Variablen (main.php)

2.2 Ermittlung aller Fahrzeuge im eingleisigen Netz und den zugehörigen Daten

ALLTRAINS BESCHREIBEN!

Das eingleisige Netz des Eisenbahn-Betriebs- und Experimentierfelds (EBuEf) kann mittels der Railcon Technik und den Decodern in den Fahrzeugen ermitteln, welches Fahrzeug aktuell welche Infrastrukturabschnitte belegt. Belegt ein Fahrzeug einen Infrastrukturabschnitt, 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*() eingelesen und die zugehörige Adresse wird der Funktion *prepareTrainForRide*() übergeben. Für jedes Fahrzeug, welches dieser Funktion übergeben wird, wird in dem Array *$allUsedTrains* ein neuer Eintrag erstellt, wobei der Index der ID des Zugs entspricht. Dabei wird für jedes Fahrzeug die exakte Position bestimmt und der Fahrplan geladen. 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 Infrastrukturabschnitte, wird mittels der Fahrtrichtung der Züge der Abschnitt ermittelt, in dem sich der Zugkopf befindet. Die aktuelle Position wird daraufhin mit dem Infrastrukturabschnitt und der relativen Position (in Metern) innerhalb des Abschnitts angegeben. Dadurch, dass davon ausgegangen wird, dass das Fahrzeug sich direkt vor dem Signal befindet, entspricht die relative Position der Abschnittslänge. Für die Überprüfung, ob ein Fahrzeug nach Fahrplan fährt oder nicht wird die Funktion *getFahrzeugZugIds*()* aufgerufen. Wenn ein Fahrzeug ohne Fahrplan

unterwegs ist (Rückgabewert der Funktion *getFahrzeugZugIds*()* ist ein leeres Array), wird in dem *$allUsedTrains*-Array dem Fahrzeug unter dem Eintrag *operates_on_timetable* der Wert *false* zugewiesen. In dem Fall, dass für das Fahrzeug ein Fahrplan hinterlegt ist (Rückgabewert der Funktion *getFahrzeugZugIds*()* ist ein Array mit allen Zug-IDs), wird mittels der Funktion *getNextBetriebsstellen*() der Fahrplan für den ersten Eintrag des Zug-ID Arrays aus der Datenbank geladen. Der Fahrplan wird in dem *$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. 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 gestellt ist, werden auch alle Betriebsstellen hinzugefügt, welche eindeutig einem Infrastrukturabschnitt 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 ist, dass die Zwischen-Betriebsstelle auf dem Weg zum nächsten fahrplanmäßigen Halt liegt. In Tabelle 1 ist für eine bessere Übersicht der Aufbau eines Eintrags abgebildet. Für die Ermittlung der Ankunfts- und Abfahrtzeiten

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 2)
haltepunkte (Array)	Alle zugehörigen Infrastrukturabschnitte
fahrplanhalt (Boolescher Wert)	Ist diese Betriebsstelle ein Fahrplanhalt

Tabelle 1: Aufbau eines Arrays in *next_betriebsstellen_data*

wird die Funktion *getFahrplanzeiten*()* 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 den Eintrag *verspaetung* ergänzt. Zudem werden die Ankunfts- und Abfahrtszeiten in das Unixtimestamp-Format mittels der Funktion *getUhrzeit*()* umgewandelt. Der Aufbau des *zeiten*-Arrays ist in der Tabelle 2 dargestellt. Für die Überprüfung, ob eine Betriebsstelle durch die aktuelle Fahrstraße erreichbar ist, müssen den Betriebsstellen die Infrastrukturabschnitte zugeordnet werden. Dafür werden mittels des *$cacheHaltepunkte*-Arrays, jeder Betriebsstelle mögliche Infrastrukturabschnitte zugeordnet. Das *$cacheHaltepunkte*-Array ist so aufgebaut, dass jeder Betriebsstelle für jede Richtung alle Infrastrukturabschnitte zugeordnet sind, welche ein Ausfahrsignal zugeordnet ist.

2.3 Berechnung der Fahrtverläufe aller Fahrzeuge

Nachdem für alle Fahrzeuge die Fahrplandaten (falls vorhanden) hinterlegt sind, wird für jedes Fahrzeug überprüft, wie die Fahrstraße aktuell eingestellt ist. Dafür wird die Funktion *calculateNextSections*() aufgerufen und das Array *$allUsedTrains* für jedes Fahrzeug um die Einträge *next_sections*, *next_lenghts* und *next_v_max* als Array ergänzt. Diese Arrays spei-

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)	Infrastrukturabschnitt der Betriebsstelle, welcher auf der Fahrstraße liegt
wendet (Integer)	Wendeauftrag nach Erreichen der Betriebsstelle (Eintrag aus der Tabelle *fahrplan_sessionfahrplan*)
verspaetung (Integer)	Verspätung, mit der das Fahrzeug diese Betriebsstelle erreicht hat

Tabelle 2: Aufbau des *zeiten*-Arrays in *next_betriebsstellen_data*

chern die IDs, Längen und zulässigen Höchstgeschwindigkeiten der nächsten Infrastrukturabschnitte ab, welche auf der Fahrstraße liegen. Im ersten Schritt wird überprüft, ob das Fahrzeug aktuell in einem Abschnitt steht, welchem ein auf Halt stehendes Signal zugeordnet ist. Wenn das der Fall ist, wird den Arrays *next_sections*, *next_lenghts* und *next_v_max* ein leeres Array zugeordnet. Wenn das Fahrzeug aktuell nicht in einem Abschnitt steht, welchem ein auf Halt stehendes Signal zugeordnet ist, wird über die Funktion *getNaechsteAbschnitte*()* die aktuelle Fahrstraße ermittelt und der Rückgabewert der Funktion *getNaechsteAbschnitte*()* in dem *$allUsedTrains*-Array unter dem Eintrag *last_get_naechste_abschnitte* gespeichert. Diese Speicherung ist notwendig, um zu überprüfen, ob sich die Fahrstraße geändert hat. Nach der Ermittlung der Fahrstraße und der Zuordnung der Infrastrukturabschnitte zu den Betriebsstellen wird im nächsten Schritt überprüft, welche Betriebsstellen des Fahrplans auf der aktuellen Fahrstraße liegen. Dafür wird mittels der Funktion *checkIfFahrstrasseIsCorrrect*() über alle Betriebsstellen der Fahrzeuge in aufsteigender Reihenfolge iteriert und die *haltepunkte* mit den Werten aus dem Array *next_sections* verglichen. Bei jedem Aufruf der Funktion wird dem Fahrzeug anfangs (falls das Fahrzeug nach Fahrplan fährt) in dem Array *$allUsedTrains* der Eintrag *fahrstrasse_is_correct* der Wert *false* zugewiesen und erst dann auf *true* gesetzt, wenn eine Betriebsstelle auf der Fahrstraße liegt. Beim Iterieren über die Betriebsstellen wird jeder Betriebsstelle anfangs der Wert *false* für den Eintrag *is_on_fahrstrasse* zugeordnet. Sobald ein Infrastrukturabschnitt einer Betriebsstelle in dem Array *next_sections* ist, wird dem Eintrag *is_on_fahrstrasse* der Wert *true* zugewiesen und unter dem Eintrag *used_haltepunkt* der Infrastrukturabschnitt gespeichert, welche auf der Fahrstraße liegt. Bei dem Iterieren über alle Betriebsstellen werden nur die Betriebsstellen beachtet, welche das Fahrzeug noch nicht erreicht hat. Dies wird über den Eintrag *angekommen (true/false)* jeder Betriebsstelle ermittelt. 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

calculateFahrverlauf() aufgerufen und innerhalb der Funktion überprüft, ob die Fahrstraße aktuell richtig eingestellt ist (*fahrstrasse_is_correct* == *true*). Wenn die Fahrstraße korrekt eingestellt ist, wird zwischen Fahrzeugen unterschieden, die nach Fahrplan unterwegs sind und Fahrzeugen, die keinen Fahrplan haben. Für Fahrzeuge mit Fahrplan muss im ersten Schritt der nächste Halt ermittelt werden. 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 *$nextBetriebsstelleIndex* abgespeichert. Sollte unter den nächsten Betriebsstellen keine dabei sein, auf die die Kriterien zutreffen, wird in einer zweiten *for*-Schleife nach den selben Kriterien (außer dem des fahrplanmäßigen Halts) nach einer Betriebsstelle gesucht und sobald eine Betriebsstelle gefunden wurde, wird die Schleife abgebrochen und der Index der Betriebsstelle unter der Variablen *$nextBetriebsstelleIndex* abgespeichert. In dem Fall, dass keine nächste Betriebsstelle ermittelt werden konnte und das Fahrzeug aktuell eine Geschwindigkeit hat, für die gilt: $v > 0km/h$, so wird eine Gefahrenbremsung eingeleitet (siehe Kapitel 3.8).

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 Infrastrukturabschnitt das Fahrzeug zum Stehen kommen soll (*used_haltepunk*-Eintrag der nächsten Betriebsstelle) und an welcher relativen Position innerhalb des Abschnitts das Fahrzeug angehalten soll (Länge des Infrastrukturabschnitts). Neben den Informationen zur Position, müssen noch die Informationen zur Zeit ermittelt werden. Dafür reicht es nicht aus, die Ankunfts- und Abfahrtszeit aus den Daten der Betriebsstelle zu übernehmen, denn in dem Fall einer Verspätung beispielsweise, würde das Fahrzeug zu kurz an einer Betriebsstelle anhalten. Deswegen, 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 *$endTime*) die aktuelle Simulationszeit genommen. Wenn die nächste Betriebsstelle nicht dem ersten fahrplanmäßigen Halt entspricht, wird als Zielzeit die Ankunftszeit der Betriebsstelle festgelegt und als Startzeit die Abfahrtszeit der vorherigen Betriebsstelle (*$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 überprüft, ob die Startzeit kleiner ist als die frühstmögliche Startzeit des Fahrzeugs (*earliest_possible_start_time*-Eintrag des Fahrzeugs) und dieser Zeit gleichgesetzt, falls das der Fall 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 $30s$ erhöht.

Allen Zügen, die ohne Fahrplan unterwegs sind, wird als Ziel-Infrastrukturabschnitt der letzte Infrastrukturabschnitt aus dem Array *last_get_naechste_abschnitte* verwendet, welchem ein Signal zugeordnet ist. Die Ziel-Position innerhalb des Abschnitts entspricht dabei ebenfalls der Länge des Abschnitts und die Überprüfung, ob ein Wendeauftrag nach dem Erreichen des Ziel-Infrastrukturabschnitt 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 Infrastrukturabschnitten aus dem *last_get_naechste_abschnitte*-Array ein Signal zugeordnet sein und die aktuelle Geschwindigkeit des Fahrzeugs ist größer als $0km/h$, so wird eine Gefahrenbremsung eingeleitet. Andernfalls wird die Funktion an dieser Stelle abgebrochen und es wird erst dann wieder versucht einen Fahrtverlauf zu berechnen, wenn sie die Fahrstraße geändert hat.

Nach der Ermittlung aller notwendigen Daten für die Berechnung des Fahrtverlaufs, wird für jedes Fahrzeug die Funktion *updateNextSpeed*() aufgerufen, welche den Fahrtverlauf berechnet und in Kapitel 3 im Detail beschrieben wird. Wichtig an dieser Stelle ist allerdings der Rückgabewert der Funktion. Dieser gibt für Fahrzeuge mit Fahrplan die Verspätung in Sekunden an, 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 oder nicht kann nur ermittelt werden, wenn die Ankunftszeit definiert ist. Für den ersten fahrplanmäßigen Halt ist in der *SQL*-Tabelle *fahrplan_sessionfahrplan* allerdings keine Ankunftszeit hinterlegt. Für den Fall, dass für ein Fahrzeug eine Fahrplan hinterlegt ist, das Fahrzeug in einem Infrastrukturabschnitt steht, welcher keiner 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. Aus diesem Grund, wurde in der Datei *globalVariables.php* die Variable *$globalFirstHaltMinTime* definiert, die angibt, wie lange ein Fahrzeug an der ersten Betriebsstelle des Fahrplans halten sollte. 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:

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

2.4 Übermittlung der Echtzeitdaten an die Fahrzeuge

Nach dem Aufruf der Funktion *updateNextSpeed*() sind für alle Fahrzeuge, für die ein Fahrtverlauf berechnet wurde, in dem Array *$allTimes* alle Echtzeitdaten enthalten. Das Array beinhaltet für jedes Fahrzeug, für das Echtzeitdaten ermittelt wurden, wiederum ein Array. Dieses Array ist unter der Adresse des Zuges abgespeichert und beinhaltet alle Echtzeitdaten eines Zuges. Der Aufbau eines Array mit Echtzeitdaten ist in Tabelle 3 dargestellt. In einer *while*-Schleife wird über alle Einträge des *$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 wenn dieser kleiner ist als die aktuelle Simulationszeit ist, ausgeführt. Nach jedem Durchlauf der *while*-Schleife wird diese mit der Funktion *sleep*() für $0,03s$ pausiert. An dieser Stelle wurde sich für einen Wert von $0,03s$ entschieden, da so die Position auf einen Meter genau bestimmt werden kann, wenn das Fahrzeug eine Geschwindigkeit von $120km/h$ hat.

Wenn für ein Fahrzeug neue Echtzeitinformationen vorliegen, wird im ersten Schritt überprüft, ob der Eintrag *live_is_speed_change true* ist, wenn das der Fall ist, wird die neue Geschwindigkeit über die Funktion *sendFahrzeugbefehl*()* dem Fahrzeug übergeben und mittels einer Terminal-Ausgabe angezeigt. Im zweiten Schritt wird der aktuelle Infrastrukturabschnitt, die aktuelle Position innerhalb des Abschnitts und die Geschwindigkeit in dem Array *$allUsedTrains* abgespeichert.

Sollte das Fahrzeug nach dem Ausführen der Echtzeitdaten einen Wendeauftrag bekom-

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 Infrastrukturabschnitt
live_section (Integer)	Infrastrukturabschnitt
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 3: Aufbau eines Eintrags aus dem*$allTimes*-Array

men und dementsprechend der Eintrag *wendet true* sein, so wird die Funktion *changeDirection*() aufgerufen. In der Funktion wird neben der Richtungsänderung auch die neue Position ermittelt (die Position eines Fahrzeugs wird immer durch den Zugkopf beschrieben) und überprüft, ob die Richtung geändert werden kann. Für die Überprüfung wird die Funktion *getNaechsteAbschnitte*()* aufgerufen und der aktuelle Infrastrukturabschnitt und die neue Richtung übergeben. Über die zurückgegebenen Abschnitte inklusive des aktuellen Abschnitts wird im nächsten Schritt iteriert und ermittelt, wie die neue Position des Fahrzeugs ist. Sollte die Länge aller nächsten Abschnitte inklusive des aktuellen Abschnitts in der Summe kleiner sein, als die Zuglänge, so 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*()* die Geschwindigkeit $-4km/h$ (entspricht einem Wendeauftrag) übergeben.

Bei einem Fahrtverlauf kann es vorkommen, dass Fahrzeuge mit Fahrplan auf der Fahrt mehrere Betriebsstellen passieren. Damit dem Eintrag *angekommen* dieser Betriebsstellen auch der Wert *true* zugewiesen werden kann, wird überprüft, ob in den Echtzeitdaten dem Eintrag *live_all_targets_reached* ein Wert zugewiesen ist. Dieser Eintrag enthält - falls das Fahrzeug eine Betriebsstelle erreicht hat - den Index der Betriebsstelle und weist der Betriebsstelle unter dem Eintrag *angekommen* den Wert true zu.

Wenn die letzten Echtzeitdaten eines Fahrzeugs übermittelt wurden (*live_target_reached* == *true*) und das Fahrzeug dementsprechend zum Stehen gekommen ist, wird überprüft, wie sich das Fahrzeug als nächstes verhalten soll. Dafür wird zwischen vier Fällen (siehe Tabelle 4) Für die Überprüfung, ob sich der Fahrplan eines Fahrzeugs geändert hat, wird über die Funktion *getFahrzeugZugIds*() die aktuelle Zug-ID abgefragt und mit der vorhe-

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

Tabelle 4: Verhalten eines Fahrzeugs nach dem Erreichen des Ziels

rigen 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*(), *addStopsectionsForTimetable*(), *calculateNextSections*(), *checkIfFahrstrasseIsCorrrect*() und *calculateFahrverlauf*() aufgerufen. Abgesehen von der ersten Funktion, werden diese Funktionen auch beim Start des Programms ausgeführt, welcher in Kapitel 2.2 beschrieben wird. Die Funktion *getFahrplanAndPositionForOneTrain*() ähnelt der in Kapitel 2.2 beschrieben Funktion *prepareTrainForRide*(), 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*() und *calculateFahrverlauf*() aufgerufen.

2.5 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 *$timeCheckFahrstrasseInterval* (*main.php*) festgelegt werden und ist standardgemäß auf 3 Sekunden festgelegt. Bei der Ermittlung und dem Vergleich der Fahrstraße wird für jedes Fahrzeug die Funktion *compareTwoNaechsteAbschnitte*() aufgerufen. Innerhalb dieser Funktion wird die in Kapitel 2.2 erläutere Funktion *calculateNextSections*() aufgerufen, mit dem Unterschied, dass die ermittelten nächsten Infrastrukturabschnitte inkl. der Längen und zulässigen Höchstgeschwindigkeiten nicht dem Fahrzeug hinterlegt werden, sondern lokal in der Funktion gespeichert. Damit die ermittelten Daten für ein Fahrzeug berechnet werden, aber nicht dem Fahrzeug hinterlegt werden, kann der Parameter *$writeResultToTrain* der Funktion *calculateNextSections*() (standardgemäß auf *true* gesetzt) auf *false* gesetzt werden.

2.6 Neukalibrierung der Fahrzeugposition

Für eine genau Fahrzeugsteuerung ist die aktuelle Position der Züge **essenziäl!** und muss während der Fahrt kalibriert werden, damit Ungenauigkeiten ausgeglichen werden können. Dafür werden die Daten aus der *SQL*-Tabelle *fahrzeuge_abschnitte* benötigt, welche durch Abschnittsüberwachung ermittelt werden. Die Abschnittsüberwachung schreibt für jedes Fahrzeug den aktuellen Infrastrukturabschnitt 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 Infrastrukturabschnitt einfährt und seit der Einfahrt in den Abschnitt die Geschwindigkeit nicht verändert hat, wird die aktuelle Positi-

on neu ermittelt. Würde sich das Fahrzeug in einem 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. Aus diesem Grund wird, sobald das Fahrzeug laut den Echtzeitdaten einen neuen Abschnitt befährt und aktuell nicht die Geschwindigkeit anpasst (*live_is_speed_change* == *false*), dem Eintrag *calibrate_section_one* der aktuelle Infrastrukturabschnitt hinzugefügt und dem Eintrag *calibrate_section_two* wird ebenfalls der aktuelle Infrastrukturabschnitt hinzugefügt, wenn *calibrate_section_one* ein Wert zugewiesen ist und dieser nicht dem aktuellen Infrastrukturabschnitt 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 Infrastrukturabschnitt zugewiesen, wenn das Fahrzeug in diesem seit der Einfahrt die Geschwindigkeit nicht verändert hat. Wenn dem Eintrag *$useRecalibration* aus der Datei *globalVariables.php* der Wert *true* zugewiesen ist, wird in regelmäßigen Abständen überprüft, ob eine Neukalibrierung möglich ist. Das Zeitintervall, in dem die Überprüfung stattfindet ist standardmäßig auf auf 3 Sekunden eingestellt, kann aber mittels der Variable *$timeCheckCalibrationInterval* (*main.php*) angepasst werden.

Für die Neukalibrierung wird die Funktion *getCalibratedPosition*() (Code-Beispiel 2) aufgerufen, welche als Rückgabewert die aktuelle relative Position und den aktuellen Infrastrukturabschnitt zurückgibt.

```
1  function getCalibratedPosition ($id, $speed) {
2    global $cacheFahrzeugeAbschnitte;
3    $DB = new DB_MySQL();
4    $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];
5    unset($DB);
6    if (in_array($id, array_keys($cacheFahrzeugeAbschnitte))) {
7      if ($positionReturn->unixtimestamp ==
           ↪ $cacheFahrzeugeAbschnitte[$id]["unixtimestamp"]) {
8        return array("possible" => false);
9      }
10   }
11   $timeDiff = time() - $positionReturn->unixtimestamp;
12   $position = ($speed / 3.6) * $timeDiff;
13   return array("possible" => true, "section" => $positionReturn->infra_id, "position"
         ↪ => $position);
14 }
```

Code-Beispiel 2: *getCalibratedPosition*() (main.php)

Sollte die ermittelte Position innerhalb des Abschnitts größer als die Länge des Abschnitts, welche in dem Array *$cacheInfraLaenge* abgespeichert ist, sein, wird die Neukalibrierung nicht durchgeführt. Der aktuelle Infrastrukturabschnitt wird aus der Tabelle *fahrzeuge_abschnitte* der Datenbank geladen und durch die aktuelle Geschwindigkeit des Fahrzeugs und die Differenz der Zeit zwischen dem Einfahren in den Abschnitt und der aktuellen Zeit wird die relative Position innerhalb des Abschnitts berechnet.

relative Position = Geschwindigkeit · Zeitdifferenz (aktuelle Zeit − Zeit des Einfahrens)

2.7 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 *$allTrainsOnTheTrack* die zugehörige Adresse gespeichert (*findTrainsOnTheTracks*()). Für die Überprüfung, ob Fahrzeuge entfernt wurden oder neu hinzugekommen sind, wird die Funktion *updateAllTrainsOnTheTrack*() verwendet. Diese Funktion wird - wie die Neukalibrierung in Kapitel 2.6 - alle 3 Sekunden ausgeführt. Bei dem Aufruf der Funktion werden alle Fahrzeuge geladen, denen in der *fma*-Tabelle aus der Datenbank ein Infrastrukturabschnitt zugeordnet ist und mit dem Array *$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 Infrastrukturabshcnitt 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*(), *addStopsectionsForTimetable*(), *calculateNextSections*(), *checkIfTrainReachedHaltepunkt*(), *checkIfFahrstrasseI*
und *calculateFahrverlauf*() aufgerufen (siehe Kapitel 2.2 und 2.3). Alle entfernten Fahrzeuge werden aus dem Array *$allUsedTrains* entfernt und somit nicht mehr von der Fahrzeugsteuerung beachtet.

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 Abschnitte
$next_lenghts (Array)	Längen aller Abschnitte
$next_v_max (Array)	Höchstgeschwindigkeit aller Abschnitte
$indexCurrentSection (Integer)	Index des aktuellen Abschnitts
$indexTargetSection (Integer)	Index des Ziel-Abschnitts
$cumulativeSectionLengthStart (Array)	Absolute Startposition aller Abschnitte
$cumulativeSectionLengthEnd (Array)	Absolute Endposition aller Abschnitte
$trainPositionChange (Array)	Alle absoluten Positionen des Fahrtverlaufs
$trainSpeedChange (Array)	Alle Geschwindigkeiten des Fahrtverlaufs

Tabelle 5: Beschreibung der verwendeten Variablen für die Fahrtverlaufsberechnung

3 Berechnung des Fahrtverlaufs

Der Fahrtverlauf eines Fahrzeuges wird bei der Berechnung in zwei verschiedenen Arten gespeichert. Einmal in so genannten *$keyPoints*, welche in einem Array die Start- und Zielgeschwindigkeit (*time_0* und *time_0*), die Start- und Endposition (*position_0* und *position_1*) und die Start- und Endzeit (*time_0* und *time_1*) einer Beschleunigung bzw. Verzögerung abspeichern. Für die Überprüfung, ob ein Fahrzeug die zulässige Höchstgeschwindigkeit in einem Infrastrukturabschnitt überschreitet, für die spätere Übermittlung der Echtzeitdaten an das Fahrzeug und die exakte Positionsbestimmung, werden mittels der *$keyPoints* für jede Geschwindigkeitsänderungen (und bei konstanter Geschwindigkeit in 1 Meter Abständen) die aktuelle relative Position innerhalb eines Infrastrukturabschnitts, der Infrastrukturabschnitt, die aktuelle Zeit und die aktuelle Geschwindigkeit in einem Array gespeichert. Der Fahrtverlauf wird mit der Funktion *updateNextSpeed*() berechnet, welche als Parameter unter anderem die Zugdaten aus dem *$allUsedTrains*-Array, Start- und Endzeit der Fahrt (*$startTime* und *$endTime*), den Ziel-Infrastrukturabschnitt (*$targetSection*) und die relative Position in dem Ziel-Infrastrukturabschnitt (*$targetPosition*) übergeben bekommt. Die für die Berechnung benötigten Daten werden in den in Tabelle 5 beschriebenen Variablen gespeichert. In dem folgenden Abschnitt werden die einzelnen Schritte beschrieben, die durchlaufen werden um den optimalen Fahrtverlauf zu berechnen. In der Darstellung 2 wird der Ablauf grob schematisch dargestellt.

3.1 Ermittlung der Start- und Endposition der einzelnen Infrastrukturabschnitte unter Berücksichtigung der Zuglänge

Für die Berechnung eines exemplarischen Fahrtverlaufs wurden die in Tabelle 6 definierten Infrastrukturabschnitte benutzt. Diese Abschnitte wurden so gewählt, sodass alle Funktionen

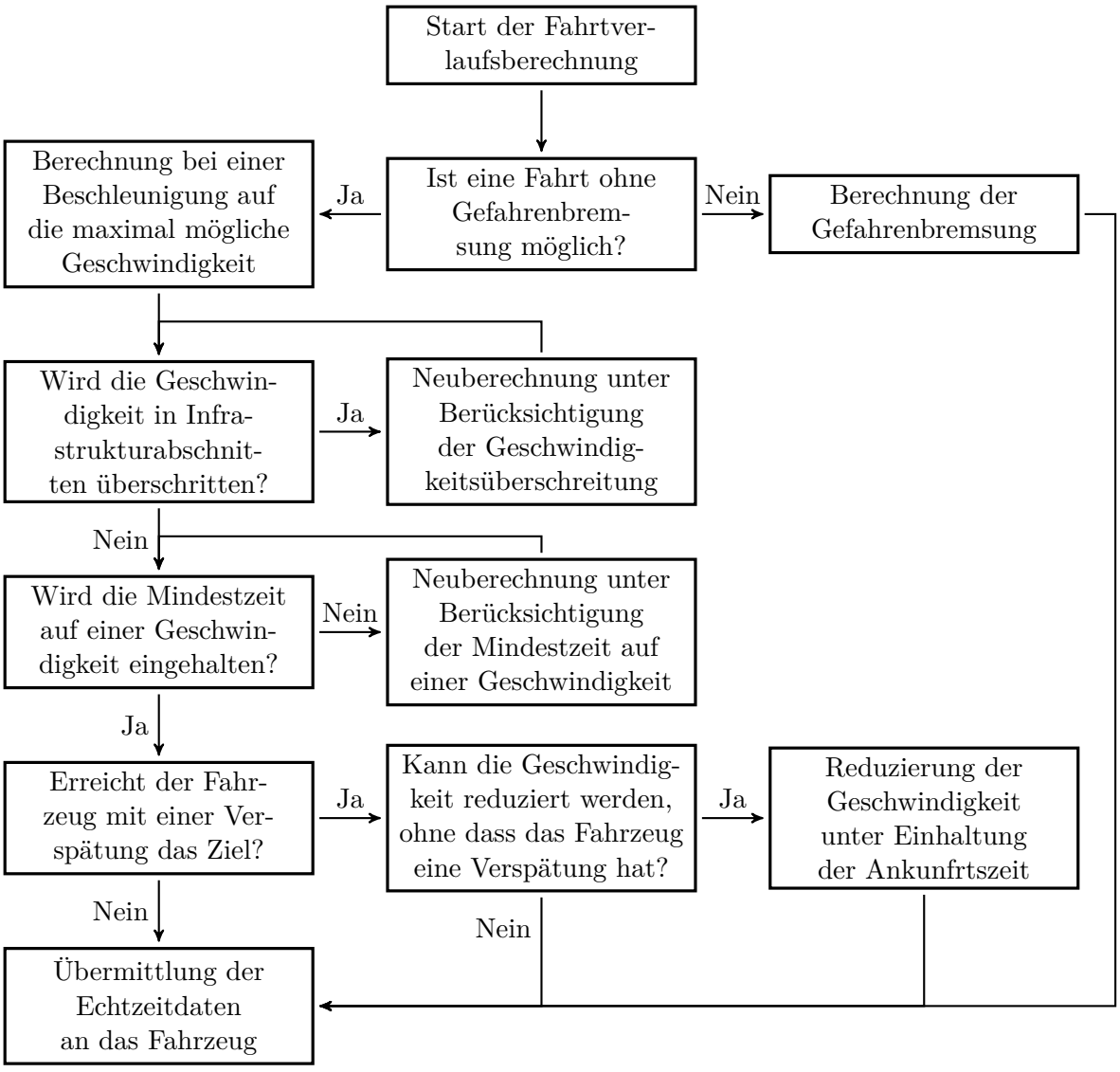

Abbildung 2: Ablaufplan der Fahrtverlaufsberechnung

Infrastrukturabschnitts-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 6: Exemplarische Infrastrukturabschnitte

und die Allgemeingültigkeit des Algorithmus gezeigt werden können und treten so im EBuEf nicht auf. Als exemplarisch gewählte Zugdaten wurden die in Tabelle 7 definierten Daten verwendet. Die zuvor ermittelten nächsten Infrastrukturabschnitte 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 Infrastrukturabschnitt eingefahren ist. In Darstellung 4 sind die Infrastrukturabschnitte dargestellt, so wie sie von dem Fahrzeug ermittelt wurden. Dabei werden alle Abschnitte, die das Fahrzeug schon durchfahren hat oder hinter dem Zielabschnitt liegen nicht dargestellt. Zudem wird in dem aktuellen Abshcnitt die relative Position von der Länge abgezogen und in der Zielabschnitt wird nur bis zur relativen Zielposition abgebildet. Dementsprechend ist der erste Abschnitt in der Darstellung 4 der Abschnitt mit der ID 1001. Dieser hat aufgrund der aktuellen relativen Position des Fahrzeugs eine Länge von 290 m. Und der letzte Abschnitt ist der Abschnitt mit der ID 1010 und einer Länge von ebenfalls 290 m. Bei

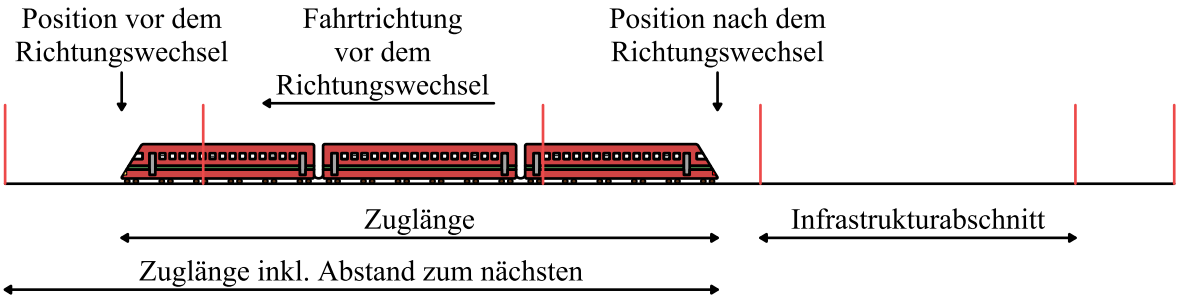

Abbildung 3: Eigene Darstellung der Positionsbestimmung bei einem Richtungswechsel

der Berücksichtigung der Fahrzeuglänge wird durch alle Infrastrukturabschnitt iteriert und die Zuglänge auf die Länge das Abschnitts addiert. Von dieser neu ermittelten Endposi-

relative Startposition	10 m
relative Zielposition	290 m
aktueller Infrastrukturabschnitt	1001
Ziel-Infrastrukturabschnitt	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 7: Exemplarische Zugdaten

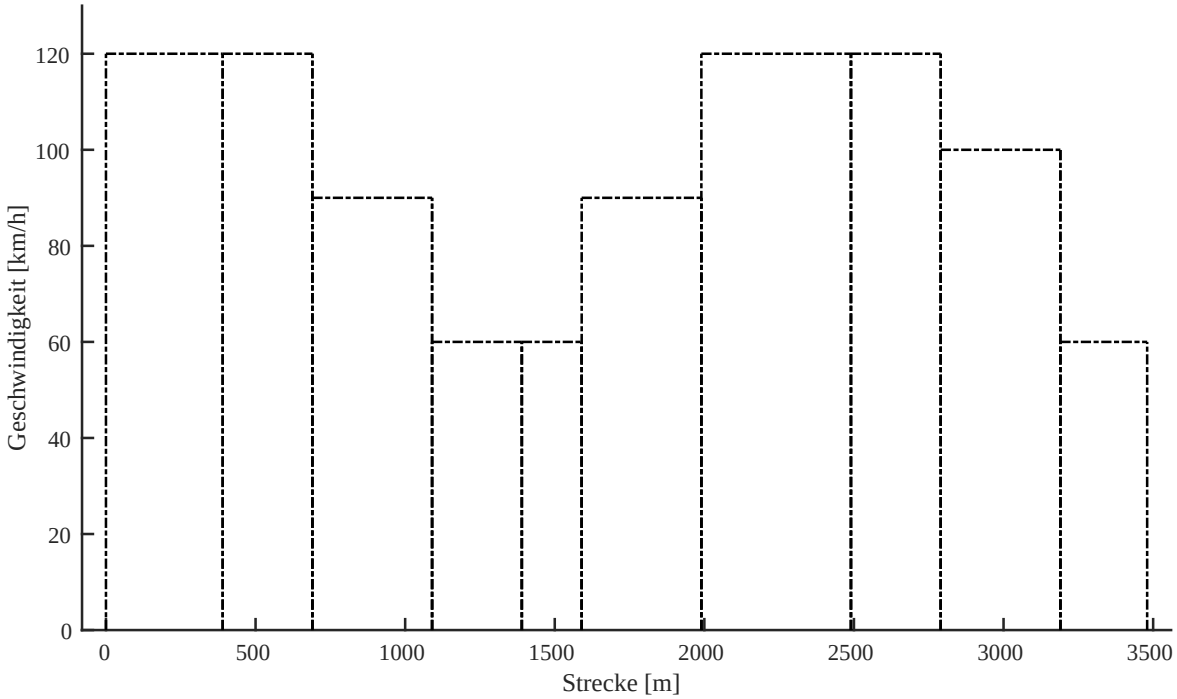

Abbildung 4: Darstellung der Infrastrukturabschnitte und die zugehörige Höchstgeschwindigkeit

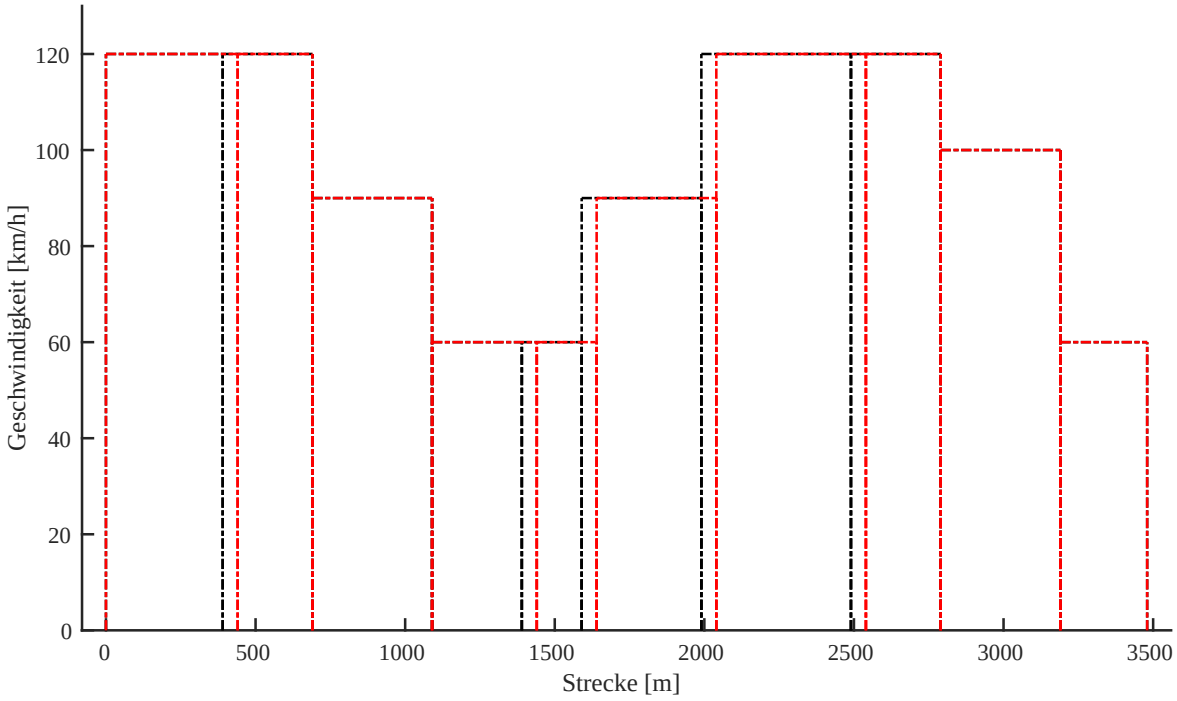

Abbildung 5: Darstellung Infrastrukturabschnitte und die zugehörige Höchstgeschwindigkeit unter Berücksichtigung der Fahrzeuglänge

tion des Abschnitts wird überprüft, ob zwischen der vorherigen Endposition und der neu ermittelten Endposition ein Infrastrukturabschnitt liegt, dessen zulässige Höchstgeschwindigkeit geringer ist, als die des ursprünglichen Abschnitts. Wenn dieser Fall eintritt, wird der Abschnitt nur so weit verlängert, sodass keine Höchstgeschwindigkeit der folgenden Abschnitte überschritten wird. Von der neu ermittelten Endposition wird überprüft, in welchem Abschnitt diese liegt und mit dem Abschnitt wird dann weiter gerechnet. Sobald der Ziel-Abschnitt erreicht wurde, wird die Schleife abgebrochen. Die neu ermittelten Abschnitte werden in den Arrays *$next_lengths_mod* und *$next_v_max_mod* abgespeichert. Durch diese Algorithmus kann es dazu kommen, dass sich die Anzahl der Abschnitte verändert hat. Dementsprechend können die Abschnitte nicht mehr eindeutig mit der Infrastruktur-ID bezeichnet werden. Mittels *$next_lengths_mod* und *$next_v_max_mod* werden mit der Funktion *createCumulativeSections*() für jeden Abschnitt die absolute Start- und Endposition in den Arrays *$cumulativeSectionLengthStartMod* und *$cumulativeSectionLengthEndMod* gespeichert. Diese Umwandlung ist essentiell für die Überprüfung, in welchem Abschnitt ein Fahrzeug sich aktuell befindet. Die neu berechneten Abschnitte werden sind in der Darstellung 5 in rot abgebildet und beschreiben die maximale Geschwindigkeit, die ein Fahrzeug fahren darf an der jeweiligen Position.

3.2 Berechnung bei einer Beschleunigung auf die maximal mögliche Geschwindigkeit

Im ersten Schritt für die Distanz zwischen der aktuellen Position und der Ziel-Position mittels *$cumulativeSectionLengthStart*, *$cumulativeSectionLengthEnd*, *$indexCurrentSection* und *$indexTargetSection* berechnet. Für diese Distanz und die Startgeschwindigkeit wird

```php
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 && $distance > 0) {
      echo "Der zug müsste langsamer als 10 km/h fahren, um das Ziel zu erreichen.";
    } else {
      // TODO: Notbremsung
    }
  } else {
    if ($v_0 == $v_1 && max($v_max) < $v_0) {
      $v_max = array($v_0);
    }
  }
  return max($v_max);
}
```

Code-Beispiel 3: *getVMaxBetweenTwoPoints*()

mit Hilfe der Funktion *getVMaxBetweenTwoPoints*() (Code-Beispiel 3) die maximale Geschwindigkeit ermittelt, die das Fahrzeug aufnehmen kann, um noch 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*() (Code-Beispiel 4) aufgerufen, welche die benötigte Distanz für eine Beschleunigung bzw. Verzögerung berechnet. Durch die gegebene Startgeschwindigkeit und die höchstmögliche Geschwindigkeit wird ein erster Fahrtverlauf berechnet. Dabei werden zwei *$keyPoints* erzeugt. Mithilfe der Funktion *createTrainChanges*() wird aus diesen beiden *$keyPoints* für jede Geschwindigkeitsveränderung die aktuelle absolute Position und Geschwindigkeit ermittelt. An den Positionen, an den das Fahrzeug eine konstante Geschwindigkeit hat, wird in 1 Meter Abständen die absolute Position und die Geschwindigkeit gespeichert. Die ermittelten Daten werden in den

```php
function getBrakeDistance (float $v_0, float $v_1, float $verzoegerung) {
  if ($v_0 > $v_1) {
    return $bremsweg = 0.5 * ((pow($v_0/3.6,2)-pow($v_1/3.6, 2))/($verzoegerung));
  } if ($v_0 < $v_1) {
    return $bremsweg = -0.5 * ((pow($v_0/3.6,2)-pow($v_1/3.6, 2))/($verzoegerung));
  } else {
    return 0;
  }
}
```

Code-Beispiel 4: *getBrakeDistance*()

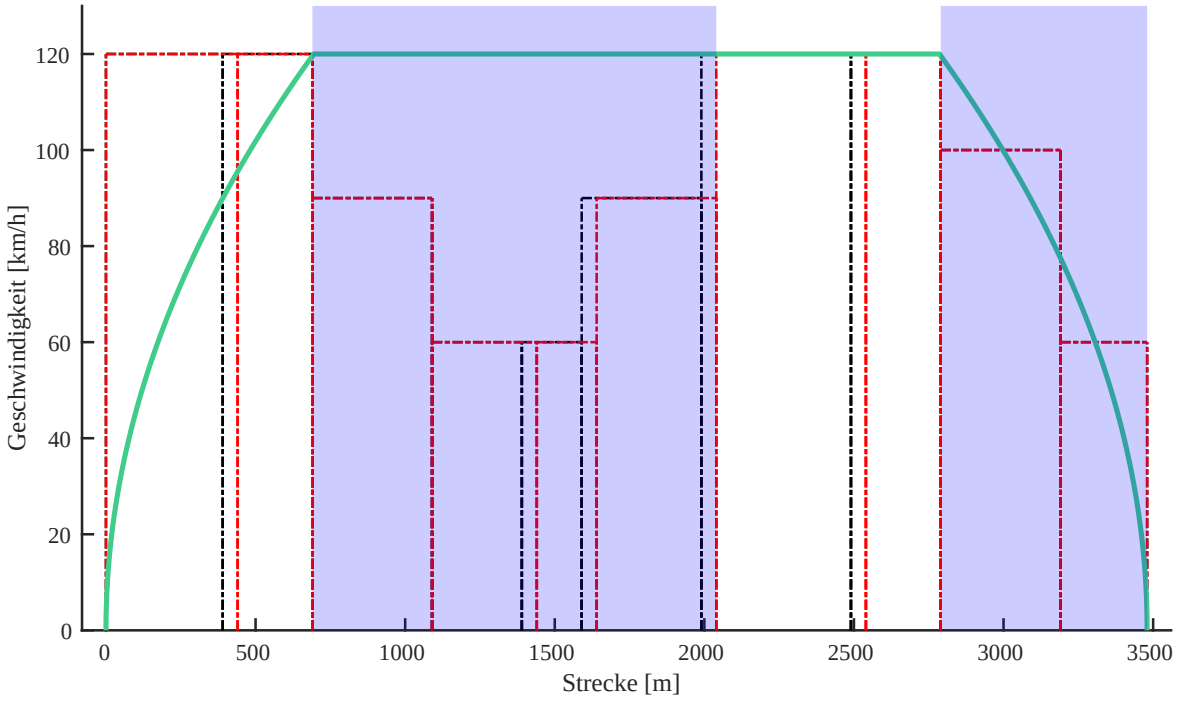

Abbildung 6: Fahrtverlaufberechnung (1. Iteration)

Arrays *$trainPositionChange* und *$trainSpeedChange* gespeichert. In der Darstellung 6 ist das Ergebnis der 1. Iteration abgebildet.

3.3 Überprüfung des Fahrtverlaufs nach Geschwindigkeitsüberschreitungen

Für die Überprüfung, ob bei einem Fahrtverlauf in manchen Infrastrukturabschnitten die zulässige Höchstgeschwindigkeit überschritten wird, wird nach jeder Berechnung die Funktion *checkIfTrainIsToFastInCertainSections*() (Code-Beispiel 5) aufgerufen. In dieser Funktion wird über alle absoluten Positionen (*$trainPositionChange*) iteriert, überprüft in welchem Abschnitt sich diese Position befindet und überprüft, ob die zugehörige Geschwindigkeit aus dem *$trainSpeedChange*-Array die zulässige Höchstgeschwindigkeit überschreitet. Sobald in einem Abschnitt eine Geschwindigkeitsüberschreitung vorliegt, wird der zugehörige Index des Abschnitts in dem *$faildSections*-Array gespeichert. Diese Abschnitte sind in der Darstellung 6 Lila hinterlegt. Als Rückgabewert der Funktion wird wird ein Array wiedergegeben, welches abspeichert, ob es zu einer Geschwindigkeitsüberschreitung gekommen ist (*"failed"*) und wenn das der Fall ist auch die Indexe der Abschnitte (*"failed_sections"*).

3.4 Neuberechnung unter Berücksichtigung der Geschwindigkeitsüberschreitung

In dem Fall, dass es zu einer Geschwindigkeitsüberschreitung gekommen ist, wird der Fahrtverlauf neu berechnet. Als Grundlage dafür diesen die *"failed_sections"* aus der *checkIfTrain IsToFastInCertainSections*() Funktion (Code-Beispiel 5). Die Funktion *recalculateKeyPoints*() vergleicht immer zwei benachbarte *$keyPoints* und berechnet in dem Fall einer Geschwin-

```php
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 5: *checkIfTrainIsToFastInCertainSections*()

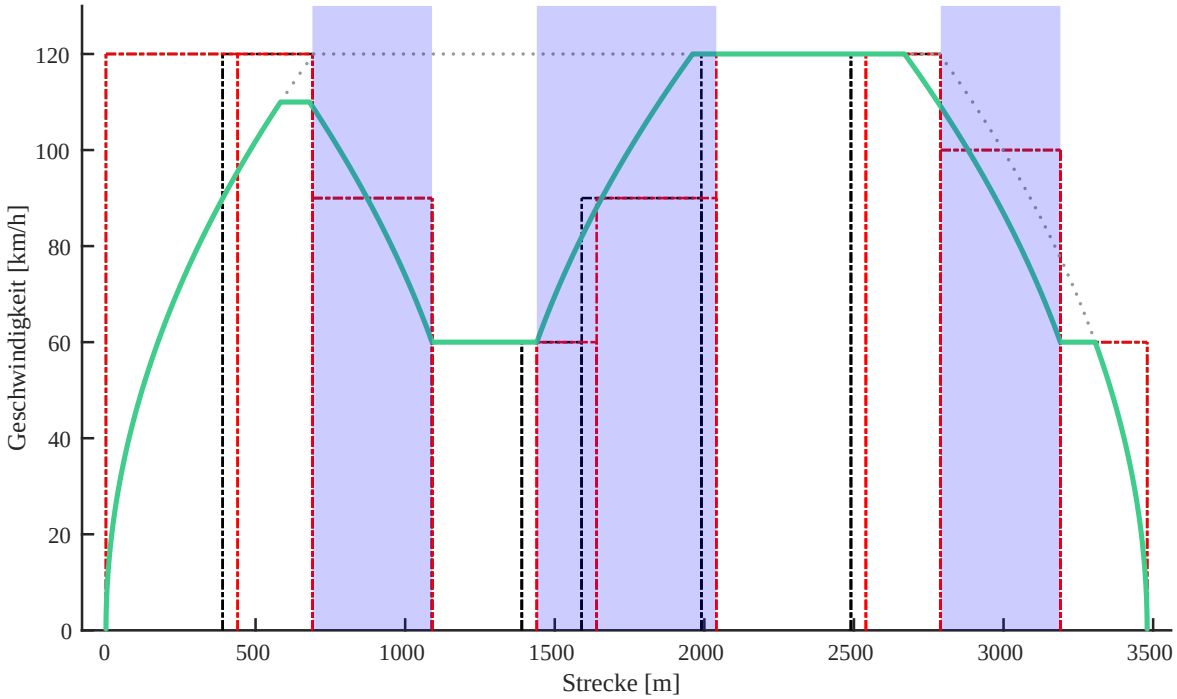

Abbildung 7: Fahrtverlaufberechnung (2. Iteration)

digkeitsüberschreitung mit der Funktion *checkBetweenTwoKeyPoints*() diese neu. In dem Fall, dass zwischen zwei benachbarten *$keyPoints* die zulässige Höchstgeschwindigkeit überschritten wird, wird die absolute Start- und End-Position dieser Geschwindigkeitsüberschreitung gespeichert. Im folgenden Schritt wird wie in dem Abschnitt 3.2 zwischen den Start-Werten des ersten *$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 es immer eine gerade anzahl an *$keyPoints* gibt und somit in jedem Iterationsschritt zwei benachbarte *$keyPoints* verglichen werden können. Nachdem alle *$keyPoint*-Paare überprüft werden, werden mit Hilfe der *createTrainChanges*() Funktion die Arrays *$trainPositionChange* und *$trainSpeedChange* erzeugt. Dieser neu berechnete Fahrtverlauf wird dann wieder der Funktion *checkIfTrain IsToFastInCertainSections*() Funktion (Code-Beispiel 5) übergeben. Dieser Prozess wird solange durchlaufen, bis es zu keiner Geschwindigkeitsüberschreitung mehr kommt. In den folgenden Abbildungen (Darstellung 7, 8 und 9) werden die Ergebnisse der einzelnen Iterationsschritte visuell abgebildet, wobei die grau gepunkteten Linien die Ergebnisse der vorherigen Iterationsschritte darstellen.

3.5 Einhaltung der Mindestzeit auf einer Geschwindigkeit

1. Ideal: möglichst späte v reduzieren

Für eine möglichst realitätsnahe Simulation kann über die Variable *$globalTimeOnOneSpeed* in der Datei *globalVariables.php* eine Mindestzeit festgelegt werden, die ein Fahrzeug auf einer Geschwindigkeit mindestens einhalten muss. Ebenfalls kann über die Variablen *$useMinTi-*

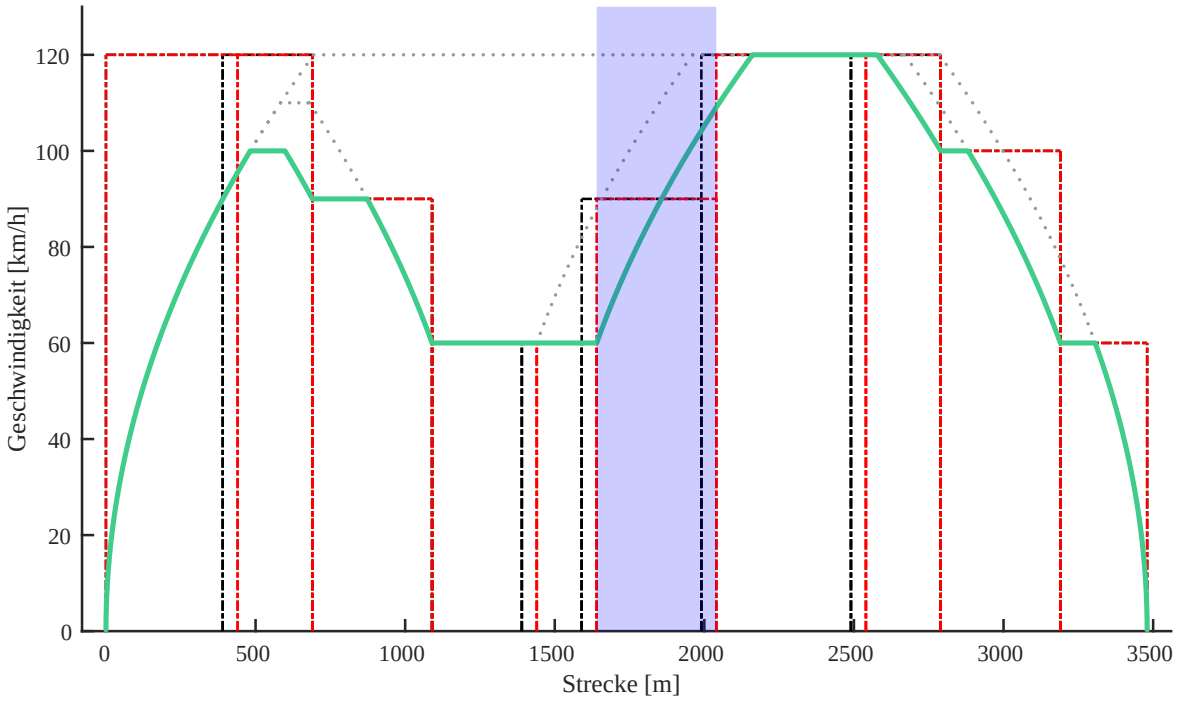

Abbildung 8: Fahrtverlaufberechnung (3. Iteration)

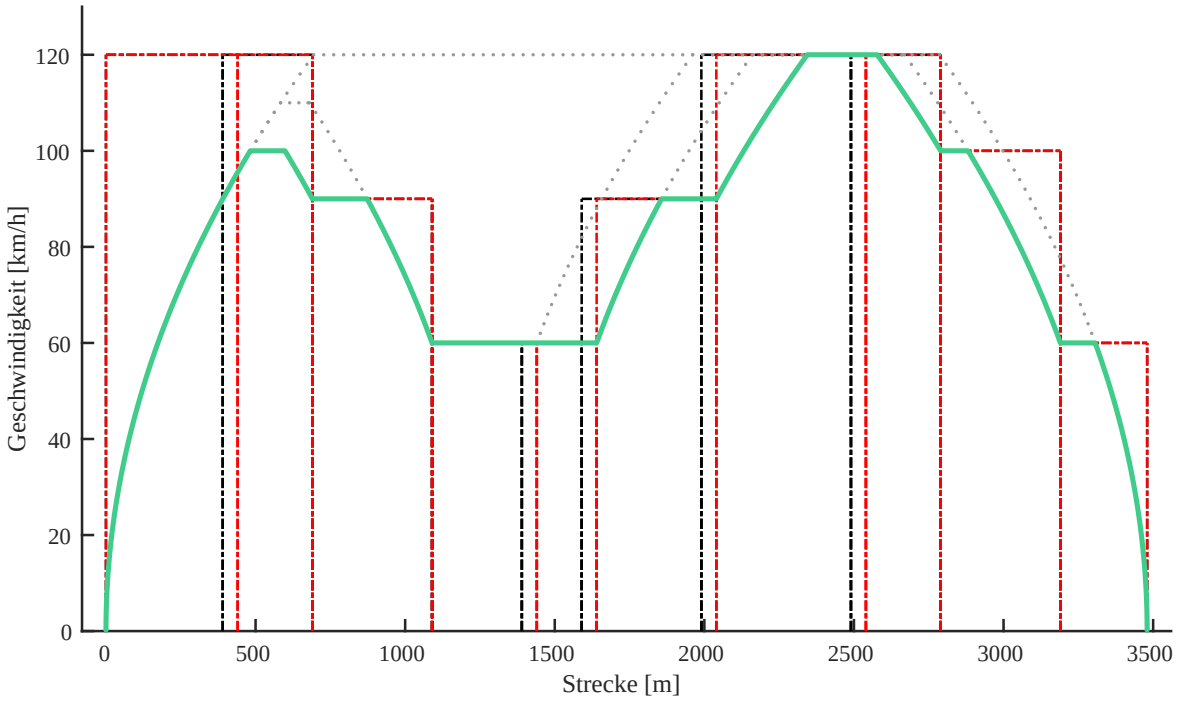

Abbildung 9: Fahrtverlaufberechnung (4. Iteration)

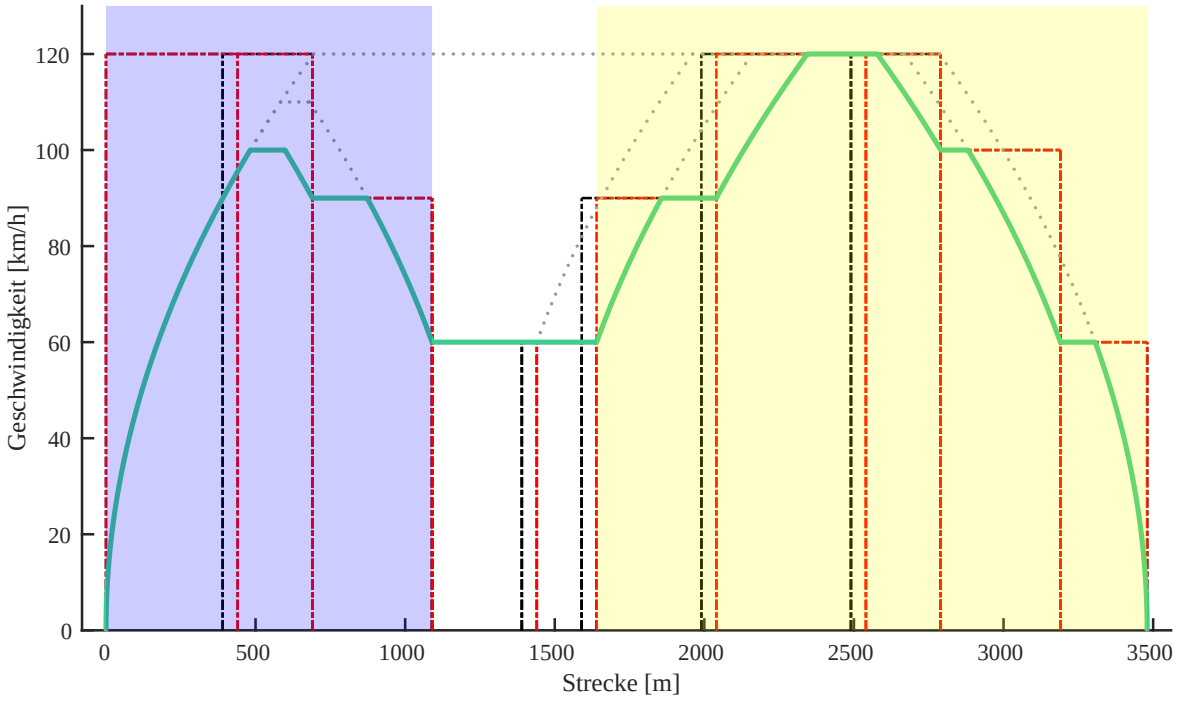

Abbildung 10: Einteilung des Fahrtverlaufs in *$subsections*

meOnSpeed 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 $v > 0$ 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. In dem folgenden Algorithmus werden die ...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 10 wurde der exemplarische Fahrtverlauf somit in zwei *$subsection* unterteilt, welche Lila bzw. Gelb hinterlegt sind. Diese Einteilung wird vorgenommen, da sich die Verschiebung einer Beschleunigung bzw. Verzögerung auf die folgenden bzw. vorherigen Abschnitte auswirkt. Durch diese Einteilung kann verhindert werden, dass es dadurch zu Konflikten kommt. Falls die Beschleunigungen bzw. Verzögerungen soweit nach hinten bzw. nach vorne verschoben werden müssen, kann die maximale Geschwindigkeit auf dieser *$subsection* reduziert werden und die zur Verfügung stehende Strecke vergrößert werden. Wie in Darstellung 10 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 dem 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*() erstellt wurden und mit der Funktion *array_reverse*() in umgekerte Reihenfolge in dem Array *$sub-*

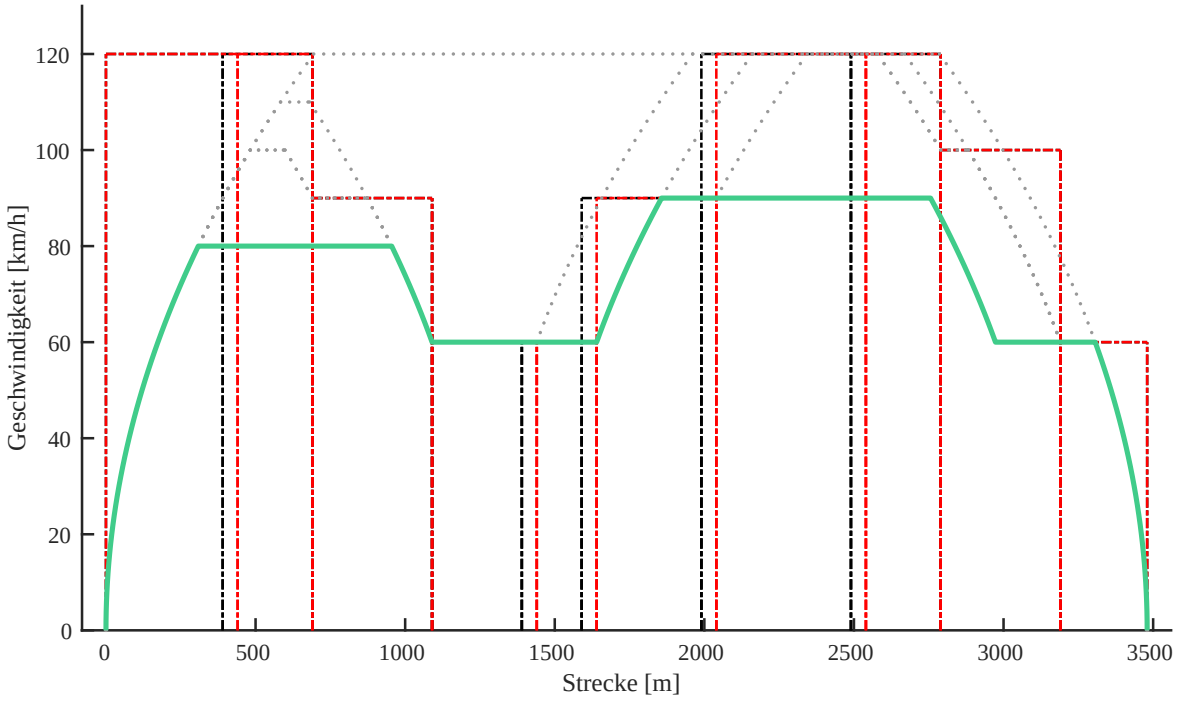

Abbildung 11: Fahrtverlauf unter Einhaltung der Mindestzeit

section_list gesammelt wurden, wird für jede *$subsection* überprüft, ob die Beschleunigungen bzw. Verzögerungen verschoben werden können. Dabei wird über alle konstanten Geschwindigkeiten iteriert, überprüft, ob die Mindestzeit eingehalten wird und wenn das nicht der Fall ist, wird überprüft, ob eine Verschiebung möglich ist. Sollte bei einer Verschiebung die *position_1* des *$keyPoints* hinter *position_0* des zweiten *$keyPoints* liegen (bei einer Beschleunigung), wird der zweite *$keyPoint* gelöscht. 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 $10km/h$ 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 11 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.

3.6 Berücksichtigung der Ankunftszeit bei der Berechnung des Fahrtverlaufs

Der berechnete Fahrtverlauf in den Kapiteln 3.2, 3.3, 3.4 und 3.5 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 der Zug allerdings mit dem Fahrtverlauf zu früh am Ziel ankommen würde, wird überprüft, ob es möglich ist

Index	Funktion
max_index	ndex des *$keyPoints* mit der Beschleunigung auf die maximale Geschwindigkeit in der *$subsection*
indexes	Indexe aller beinhalteten *$keyPoints*
is_prev_section	Berücksichtigung des Abschnitts vor der *$subsection*
is_next_section	Berücksichtigung des Abschnitts nach der *$subsection*
failed	Unterschreitung der Mindestzeit auf der *$subsection*
brakes_only	.....

Tabelle 8: Aufbau des *$subsection*-Arrays

die Geschwindigkeit zu reduzieren, sodass der Zug energieeffizienter fahren kann und ohne Verspätung am Ziel ankommt. Ergebnis ist in 11 abgebildet. Im ersten Schritt wird mittels der Funktion *checkIfTheSpeedCanBeDecreased*() 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* wiedergegeben, welches die Einträge *possible* und *range* enthält und als *$returnSpeedDecrease* abgespeichert. Der Eintrag *possible* gibt an, ob das Fahrzeug auf dem gesamten Fahrtverlauf die Geschwindigkeit reduzieren könnte und wird als Boolescher Wert (*true*/*false*) abgespeichert und und in dem *Array range* werden alle Indexe der möglichen *$keyPoints* inklusive der ermittelten Geschwindigkeiten abgespeichert. In dem in Abbildung 11 dargestellten Fahrtverlauf wären so 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 *$keyPoint* mit dem Index 2 die Geschwindigkeiten 60, 70, 80 und 90 km/h. Wenn eine Reduzierung der Geschwindigkeit möglich ist, wird in einer *while*-Schleife versucht die Geschwindigkeit zu reduzieren, bis das Fahrzeug bei der nächsten Reduzierung mit einer Verspätung am Ziel ankommen würde oder eine weitere Reduzierung nicht möglich ist, da die Maximalgeschwindigkeit auf dem Fahrtverlauf 10 km/h beträgt. Innerhalb der *while*-Schleife ermittelt die Funktion *findMaxSpeed*() aus dem *$returnSpeedDecrease*-Array den *$keyPoint* mit der höchsten Geschwindigkeit. Für den Fall, dass mehrere *$keyPoints* die selbe Höchstgeschwindigkeit haben, wird der letzte dieser *$keyPoints* ermittelt. Im Anschluss wird mit einer *for*-Schleife in 10er-Schritten in absteigender Reihenfolge über die möglichen Geschwindigkeiten iteriert und überprüft, ob durch die Anpassung die Ankunftszeit eingehalten werden kann. Sobald die Ankunftszeit nicht eingehalten werden kann, werden die *$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*() erneut aufgerufen. Das Ergebnis dieser Berechnung ist in der Abbildung 12 zu sehen.

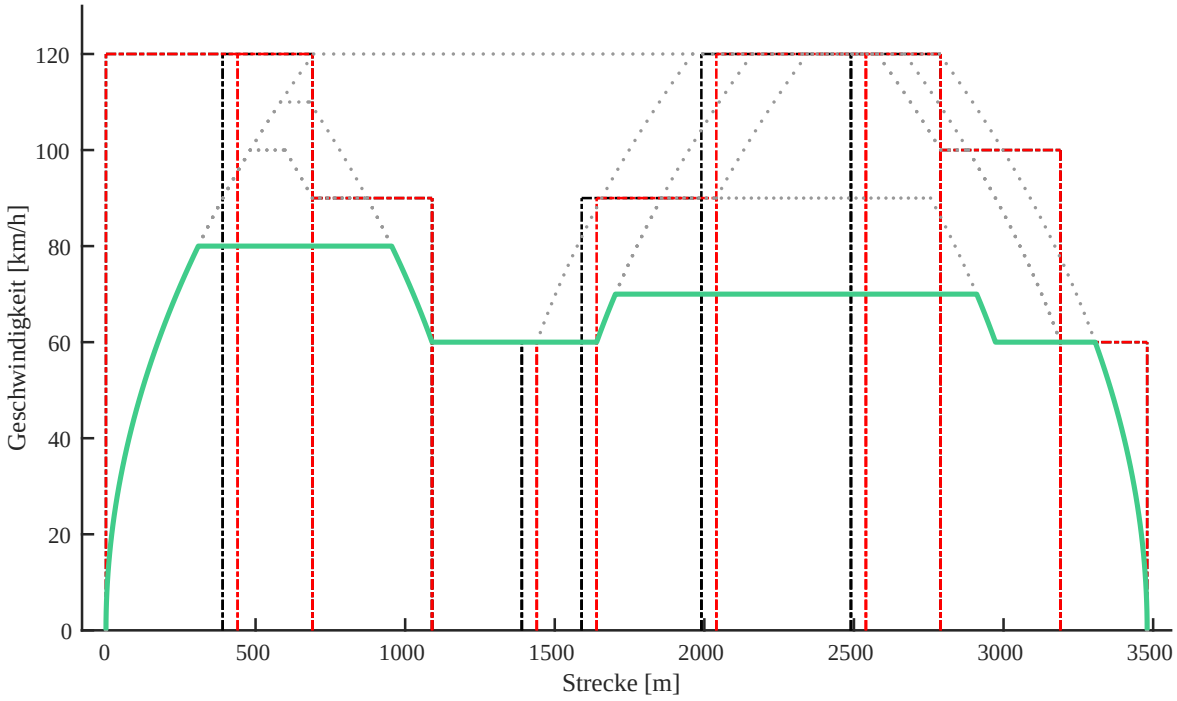

Abbildung 12: Fahrtverlauf mit reduzierter Geschwindigkeit unter Einhaltung der Ankunftszeit

3.7 Berücksichtigung der exakten Ankunftszeit bei der Berechnung des Fahrtverlaufs

Die in Kapitel 3.5 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 noch 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 *$keyPoints* vorzeitiger eingeleitet werden kann. Sollte die Zielgeschwindigkeit der Verzögerung 0 km/h sein, wird die Verzögerung unterteilt in eine Verzögerung auf 10 km/h und eine von 10 km/h auf 0 km/h. Die Position der vorzeitig eingeleiteten Verzögerung wird mittels der Funktion *speedFineTuning*() berechnet, welche als Parameter den Betrag der Differenz zwischen aktueller Soll- und Ist-Ankunftszeit und den Index des vorherigen *$keyPoints* übergeben bekommt. In Abbildung 13 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 9 sind die exakten Werte des exemplarischen Fahrtverlaufs aufgelistet, damit die verwendete Gleichung (Gleichung 16 aus Kapitel 4) an diesem Beispiel angewandt werden kann. In diesem konkreten Beispiel würde das Fahrzeug 3,31 s (t_Δ) zu früh an der Haltestelle ankommen, wodurch das Fahrzeug für die Zurücklegung der Strecken s_1 und s_2 insgesamt 85,42s ($t_{ges} = t_1 + t_2 + t_\Delta$) zur Verfügung hat. Durch das Einsetzen dieser Werte in die Gleichung 16 aus dem Kapitel 4 ergibt sich für t_3 (t_3, t_4, s_3 und s_4 bezeichnen die

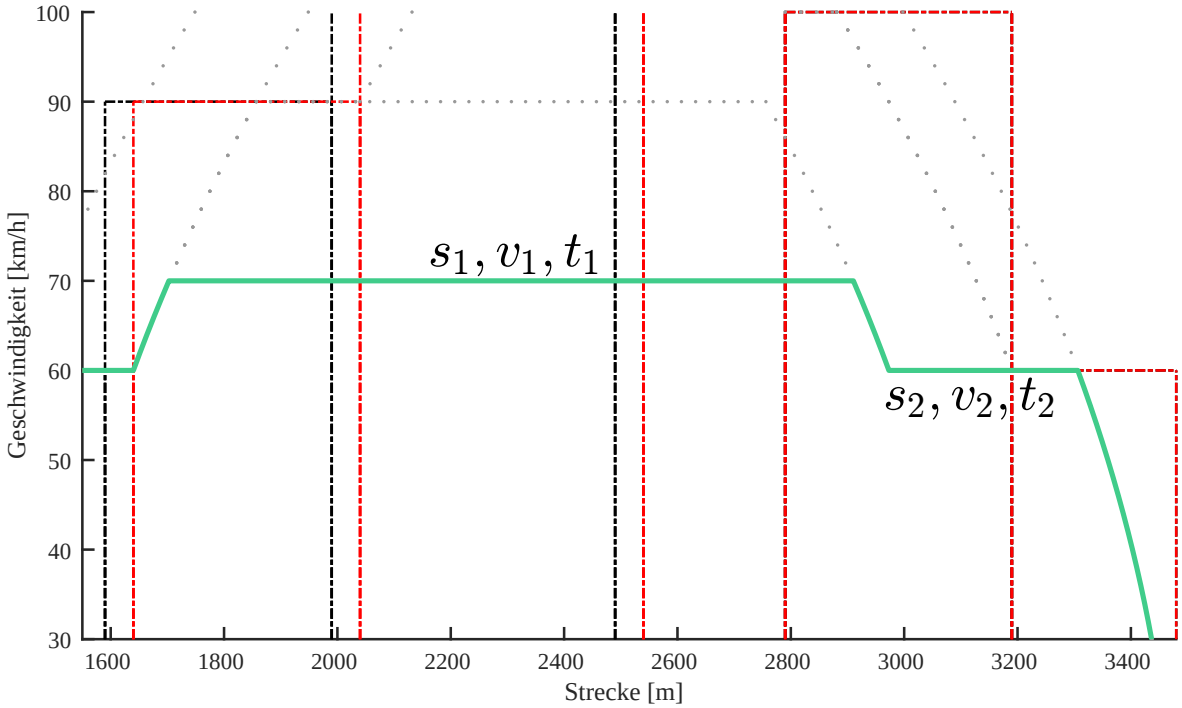

Abbildung 13: speedFineTuning_1

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

Strecken und Zeiten nach der Anpassung) ein Wert von 42,2s. Dementsprechend muss die Verzögerung 19,85s (t_1 - t_3) früher eingeleitet werden Die vorzeitige Einleitung der Verzögerung sorgt dafür, dass das Fahrzeug seinen nächsten Haltepunkt genau pünktlich erreicht und ist in Abbildung 14 dargestellt, wobei durch die gepunktete Linie der Fahrtverlauf vor der Anpassung zu sehen ist. Die neu berechneten Werte sind in Tabelle 10 aufgelistet. Der finale Fahrtverlauf ist in Abbildung 15 dargestellt und kann so dem Fahrzeug übergeben werden.

3.8 Einleitung einer Gefahrenbremsung

Eine Gefahrenbremsung wird eingeleitet, sobald ein Fahrzeug bei einer sofortigen Verzögerung ein auf Halt stehendes Signal überfahren würde, in einem Infrastruktur-Abschnitt die zulässige Höchstgeschwindigkeit überschreiten würde oder an dem nächsten planmäßigen Halt nicht rechtzeitig zum stehen kommen würde. Bei einer Gefahrenbremsung wird mit einer Notbremsverzögerung von $2m/s^2$ abgebremst. Dieser Wert kann in der Datei *globalVariables.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: $v \geq 10km/h$, nach der Geschwindigkeit von $10km/h$ direkt die Geschwindigkeit von $0km/h$ übermittelt bekommen. Dadurch wird bei der Berechnung einer Gefahrenbremsung zwischen drei Fällen unterschieden:

v_1	70 km/h (19,44m/s)
v_2	60 km/h (16,67m/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 9: Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung

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

Tabelle 10: Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung nach der Anpassung

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

Für die Überprüfung, ob das Fahrzeug mit der Notbremsverzögerung vor der Gefahrenstelle zum Stehen kommt, wird mittels der Funktion *getBrakeDistance*() 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 $2km/h$-Schritten auf $0km/h$ abbremsen. In dem Fall, dass der Bremsweg länger als die Strecke bis zur Gefahrenstelle ist, wird überprüft, welche Geschwindigkeit das Fahrzeug an der Gefahrenstelle hat. Für diese Berechnung wird die Gleichung 11 aus dem Kapitel 4 verwendet. Sollte das Fahrzeug an der Gefahrenstelle eine Geschwindigkeit von $v \geq 10km/h$ haben, bremst das Fahrzeug in $2km/h$-Schritten auf $10km/h$ ab und bekommt nach der Übermittlung der $10km/h$ direkt $0km/h$ übergeben. In dem Fall, dass das Fahrzeug an der Gefahrenstelle langsamer als $10km/h$ ist, bremst das Fahrzeug wie im 1. Fall in $2km/h$-Schritten auf $0km/h$ ab. Bei einer Gefahrenbremsung bekommt das jeweilige Fahrzeug eine Fehlermeldung übermittelt und wird nicht weiterfahren. Das liegt daran, dass 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

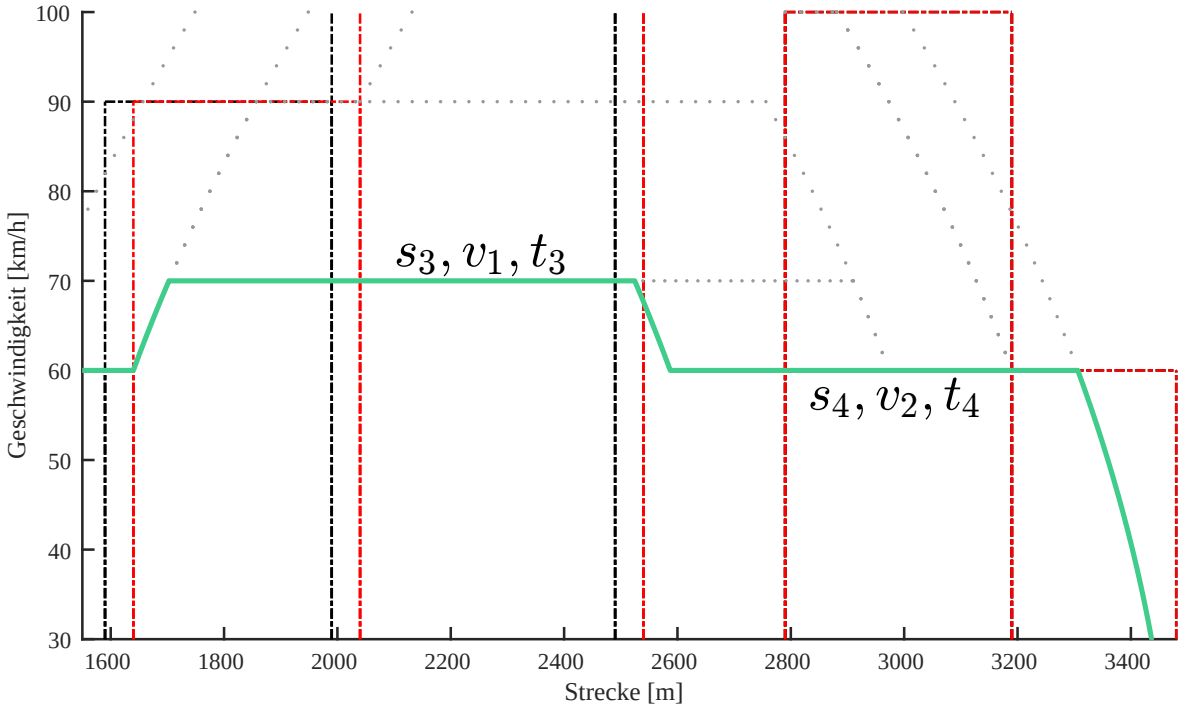

Abbildung 14: speedFineTuning_2

registriert hat und wieder neu positioniert werden.

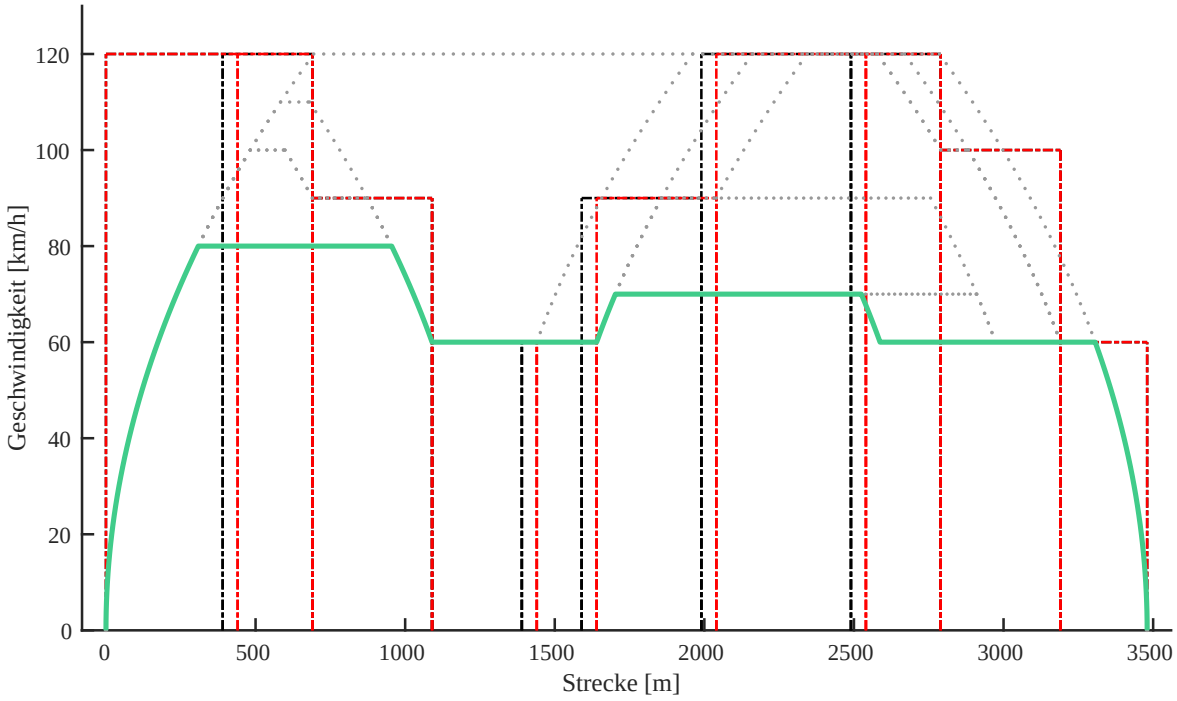

Abbildung 15: Finaler Fahrtverlauf

4 Formeln

Für die im folgenden Abschnitt verwendeten Gleichungen gilt:

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

Bei einer konstanten Beschleunigung gilt:

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

Für die Bestimmung der Geschwindigkeit in Abhängigkeit der Zeit, muss die Beschleunigung $a(t)$ nach der Zeit t integriert werden.[1]

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

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.

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

[1] Richard & Sander (2011, S. 20)

Für die Bestimmung der benötigten Zeit muss die Geschwindigkeit erneut integriert werden.[2] Die dabei entstehende Integrationskonstante s_0 gibt dabei die bereits zurückgelegte Strecke an.

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

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

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

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

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

In dem Programm übernimmt diese Berechnung die Funktion *getBrakeDistance()*. Damit keine negativen Rückgabewerte entstehen, wird im Falle einer Bremsung ($v_1 < v_0$) das Ergebnis mit -1 multipliziert. Beispiel eines Querverweis (9).

```
1 function getBrakeDistance (float $v_0, float $v_1, float $verzoegerung) {
2   if ($v_0 > $v_1) {
3     return $bremsweg = 0.5 * ((pow($v_0/3.6,2)-pow($v_1/3.6, 2))/($verzoegerung));
4   } else if ($v_0 < $v_1) {
5     return $bremsweg = -0.5 * ((pow($v_0/3.6,2)-pow($v_1/3.6, 2))/($verzoegerung));
6   } else {
7     return 0;
8   }
9 }
```

[2] ebd. (S. 20)

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 v_0 auf v_1 zu beschleunigen bzw. abzubremsen und aus der Differenz die benötigte Zeit berechnet.

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

In dem Programm übernimmt diese Berechnung die Funktion *getBrakeTime()*. Damit keine negativen Rückgabewerte entstehen, wird im Falle einer Bremsung ($v_1 < v_0$) das Ergebnis mit -1 multipliziert.

```
1 function getBrakeTime (float $v_0, float $v_1, float $verzoegerung) {
2   if ($v_0 < $v_1) {
3     return (($v_1/3.6)/$verzoegerung) - (($v_0/3.6)/$verzoegerung);
4   } else if ($v_0 > $v_1) {
5     return (($v_0/3.6)/$verzoegerung) - (($v_1/3.6)/$verzoegerung);
6   } else {
7     return 0;
8   }
9 }
```

Für die Berechnung einer Gefahrenbremsung ist es notwendig zu wissen, welche Geschwindigkeit das Fahrzeug an der Stelle der Gefahrenstelle hat. Dafür wird die Gleichung (9) nach v_2 umgestellt.

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

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:

$$t_{ges} = t_1 + t_2 \tag{12}$$

$$s_{ges} = s_1 + s_2 \tag{13}$$

$$s = v \cdot t \tag{14}$$

Durch das Einsetzen der Gleichung (14) in die Gleichung (13) erhält man folgende Gleichung:

$$s_{ges} = v_1 \cdot t_1 + v_2 \cdot t_2 \tag{15}$$

Durch das Umstellen der Gleichung (12) nach t_2 und dem Einsetzen in Gleichung (15) gilt für t_1:

$$t_1 = \frac{s_{ges} - v_2 \cdot t_{ges}}{v_1 - v_2} \tag{16}$$

5 Visualisierung der Fahrtverläufe

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in

vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis.

6 Fazit

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

6.1 Was funktioniert gut?

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

6.2 Was funktioniert nicht?

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

6.3 Wie könnte man die Fehler in der Zukunft ausbessern?

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

6.4 Was für Erweiterungsmöglichenkeiten gibt es?

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam

voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

A Anhang

A.1 main.php

```php
<?php
// Load all required external files
require 'config/multicast.php';
require 'vorbelegung.php';
require 'functions/sort_functions.php';
require 'functions/cache_functions_own.php';
require 'functions/ebuef_functions.php';
require 'functions/fahrtverlauf_functions.php';
require 'globalVariables.php';

// Set timezone
date_default_timezone_set("Europe/Berlin");

// Reports only errors
error_reporting(1);

// Global Variables
global $useRecalibration;

// Define own train errors
$trainErrors = array();
$trainErrors[0] = "Zug stand falsch herum und war zu lang um die Richtung zu ändern.";
$trainErrors[1] = "In der Datenbank ist für den Zug keine Zuglänge angegeben.";
$trainErrors[2] = "In der Datenbank ist für den Zug keine v_max angegeben.";
$trainErrors[3] = "Zug musste eine Notbremsung durchführen.";

// Load static data from the databse into the cache
$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()

// Global variables
$allTrainsOnTheTrack = array();  // All adresses found on the tracks
$allTrains = array();    // All trains with the status 1 or 2
```

```php
$allUsedTrains = array();   // All trains with the status 1 or 2 that are standing on the tracks
$allTimes = array();
$lastMaxSpeedForInfraAndDir = array();

// Get simulation and real time
$simulationStartTimeToday = getUhrzeit(getUhrzeit($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;

// Start Message
startMessage();

// Load all trains
$allTrains = getAllTrains();

// Loads all trains that are in the rail network and prepares everything for the start
findTrainsOnTheTracks();

// Checks if the trains are in the right direction and turns them if it is necessary and possible.
consoleCheckIfStartDirectionIsCorrect();
consoleAllTrainsPositionAndFahrplan();
showErrors();

// Adds all the stops of the trains.
addStopsectionsForTimetable();

// Adds an index (address) to the $allTimes array for each train.
initalFirstLiveData();

// Determination of the current routes of all trains.
calculateNextSections();

// Checks whether the trains are already at the first scheduled stop or not.
checkIfTrainReachedHaltepunkt();

// Checks whether the routes are set correctly.
checkIfFahrstrasseIsCorrrect();

// Calculate driving curve
calculateFahrverlauf();

//$unusedTrains = array_keys($allTimes);
$timeCheckFahrstrasseInterval = 3;
```

```
95  $timeCheckFahrstrasse = $timeCheckFahrstrasseInterval + microtime(true);
96  $timeCheckAllTrainErrorsInterval = 30;
97  $timeCheckAllTrainErrors = $timeCheckAllTrainErrorsInterval + microtime(true);
98  $timeCheckCalibrationInterval = 3;
99  $timeCheckCalibration = $timeCheckCalibrationInterval + microtime(true);
100 $sleeptime = 0.03;
101
102 while (true) {
103   foreach ($allTimes as $timeIndex => $timeValue) {
104     if (sizeof($timeValue) > 0) {
105       $id = $timeValue[0]["id"];
106       if ((microtime(true) + $timeDifference) > $timeValue[0]["live_time"]) {
107         if ($timeValue[0]["live_is_speed_change"]) {
108           $allUsedTrains[$id]["calibrate_section_one"] = null;
109           $allUsedTrains[$id]["calibrate_section_two"] = null;
110           if ($timeValue[0]["betriebsstelle"] == 'Notbremsung') {
111             sendFahrzeugbefehl($timeValue[0]["id"], intval($timeValue[0]["live_speed"]));
112             echo "Der Zug mit der Adresse ", $timeIndex, " leitet gerade eine
                  ↪ Gefahrenbremsung ein und hat seine Geschwindigkeit auf ",
                  ↪ $timeValue[0]["live_speed"], " km/h angepasst.\n";
113           } else {
114             sendFahrzeugbefehl($timeValue[0]["id"], intval($timeValue[0]["live_speed"]));
115             echo "Der Zug mit der Adresse ", $timeIndex, " hat auf der Fahrt nach ",
                  ↪ $timeValue[0]["betriebsstelle"],
116             " seine Geschwindigkeit auf ", $timeValue[0]["live_speed"], " km/h
                  ↪ angepasst.\n";
117           }
118         } else {
119           if (isset($allUsedTrains[$id]["calibrate_section_one"])) {
120             if ($allUsedTrains[$id]["calibrate_section_one"] !=
                    ↪ $timeValue[0]["live_section"]) {
121               $allUsedTrains[$id]["calibrate_section_two"] =
                      ↪ $timeValue[0]["live_section"];
122             }
123           }
124           $allUsedTrains[$id]["calibrate_section_one"] = $timeValue[0]["live_section"];
125         }
126
127         $allUsedTrains[$id]["current_position"] =
                ↪ $timeValue[0]["live_relative_position"];
128         $allUsedTrains[$id]["current_speed"] = $timeValue[0]["live_speed"];
129         $allUsedTrains[$id]["current_section"] = $timeValue[0]["live_section"];
130
131         if ($timeValue[0]["wendet"]) {
132           //$allTimes[$timeIndex] = array();
133           changeDirection($timeValue[0]["id"]);
134         }
135
136         if (isset($timeValue[0]["live_all_targets_reached"])) {
137           $allUsedTrains[$id]["next_betriebsstellen_data"][$timeValue[0]["live_all_targets_reached"]]["ang
                  ↪ = true;
138           echo "Der Zug mit der Adresse ", $timeIndex, " hat den Halt ",
                  ↪ $allUsedTrains[$id]["next_betriebsstellen_data"][$timeValue[0]["live_all_targets_reached"
```

```
            ↪ "_erreicht.\n";
        }

        if ($timeValue[0]["live_target_reached"]) {

          $currentZugId = $allUsedTrains[$id]["zug_id"];
          $newZugId = getFahrzeugZugIds(array($id));

          if (sizeof($newZugId) == 0) {
            $newZugId = null;
          } else {
            $newZugId = getFahrzeugZugIds(array($timeValue[0]["id"]));
            $newZugId = $newZugId[array_key_first($newZugId)]["zug_id"];
          }

          // Fährt nach Fahrplan und hat keine neue Zug ID bekommen
          if (!($currentZugId == $newZugId && $currentZugId != null)) {
            if ($currentZugId != null && $newZugId != null) {
              // neuer fahrplan
              $allUsedTrains[$id]["zug_id"] = $newZugId;
              $allUsedTrains[$id]["operates_on_timetable"] = true;
              getFahrplanAndPositionForOneTrain($id, $newZugId);
              addStopsectionsForTimetable($id);
              calculateNextSections($id);
              checkIfFahrstrasseIsCorrrect($id);
              calculateFahrverlauf($id);

            } else if ($currentZugId == null && $newZugId != null) {
              // fährt jetzt nach fahrplan
              $allUsedTrains[$id]["zug_id"] = $newZugId;
              $allUsedTrains[$id]["operates_on_timetable"] = true;
              getFahrplanAndPositionForOneTrain($id);
              addStopsectionsForTimetable($id);
              calculateNextSections($id);
              checkIfFahrstrasseIsCorrrect($id);
              calculateFahrverlauf($id);

            } else if ($currentZugId != null && $newZugId == null) {
              // fährt jetzt auf freier strecke
              $allUsedTrains[$id]["operates_on_timetable"] = false;
              calculateNextSections($id);
              calculateFahrverlauf($id);
            }
          }
        }
        array_shift($allTimes[$timeIndex]);
      }
    }
  }
  // Neukalibrierung
  if ($useRecalibration) {
    if (microtime(true) > $timeCheckCalibration) {
      foreach ($allUsedTrains as $trainKey => $trainValue) {
```

```
191         if (isset($allUsedTrains[$trainKey]["calibrate_section_two"])) {
192           $newPosition = getCalibratedPosition($trainKey,
                  ↪ $allUsedTrains[$trainKey]["current_speed"]);
193           if ($newPosition["possible"]) {
194             echo "Die Position des Fahrzeugs mit der ID: ", $trainKey, " wird neu
                    ↪ ermittelt.\n";
195             $position = $newPosition["position"];
196             $section = $newPosition["section"];
197             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";
198             if ($position > $cacheInfraLaenge[$section]) {
199               echo "Die Position konnte nicht neu kalibriert werden, da die aktuelle
                      ↪ Position im Abschnitt größer ist, als die Länge des Abschnitts.\n";
200             } else {
201               $allUsedTrains[$trainKey]["current_section"] = $section;
202               $allUsedTrains[$trainKey]["current_position"] = $position;
203               calculateNextSections($trainKey);
204               checkIfFahrstrasseIsCorrrect($trainKey);
205               calculateFahrverlauf($trainKey, true);
206               echo "Die Position des Fahrzeugs mit der ID: ", $trainKey, " wurde neu
                      ↪ ermittelt.\n";
207             }
208           } else {
209             //echo "Die Position des Fahrzeugs mit der ID: ", $trainKey, " konnte nicht
                    ↪ neu ermittelt werden.\n";
210           }
211         }
212       }
213       $timeCheckCalibration = $timeCheckCalibration + $timeCheckCalibrationInterval;
214     }
215   }
216   // Überprüfung, ob die Fahrstraße sich geändert hat und ob neue Fahrzeuge auf das
        ↪ Netz gesetzt wurden
217   if (microtime(true) > $timeCheckFahrstrasse) {
218     foreach ($allUsedTrains as $trainID => $trainValue) {
219       compareTwoNaechsteAbschnitte($trainID);
220     }
221     $returnUpdate = updateAllTrainsOnTheTrack();
222     $newTrains = $returnUpdate["new"];
223     $removeTrains = $returnUpdate["removed"];
224
225     if (sizeof($newTrains) > 0) {
226       echo "Neu hinzugefügte Züge: \n";
227       foreach ($newTrains as $newTrain) {
228         $id = $cacheDecoderToID[$newTrain];
229         echo "\tID:\t", $id, "\tAdresse:\t", $newTrain;
230
231       }
232       echo "\n";
233     }
```

```
234     foreach ($newTrains as $newTrain) {
235       $allTrains = getAllTrains();
236       $id = $cacheDecoderToID[$newTrain];
237       prepareTrainForRide($newTrain);
238       consoleCheckIfStartDirectionIsCorrect($id);
239       consoleAllTrainsPositionAndFahrplan($id);
240       addStopsectionsForTimetable($id);
241       //initalFirstLiveData($id);
242       calculateNextSections($id);
243       checkIfTrainReachedHaltepunkt($id);
244       checkIfFahrstrasseIsCorrrect($id);
245       calculateFahrverlauf($id);
246     }
247     if (sizeof($removeTrains) > 0) {
248       echo "Entfernte  Züge: \n";
249       foreach ($removeTrains as $removeTrain) {
250         $id = $cacheDecoderToID[$removeTrain];
251         unset($allUsedTrains[$id]);
252         // DELETE FROM ALLTIMES
253         echo "\tID:\t", $id, "\tAdresse:\t", $removeTrain;
254       }
255       echo "\n";
256     }
257     $timeCheckFahrstrasse = $timeCheckFahrstrasse + $timeCheckFahrstrasseInterval;
258   }
259   // Ausgabe aller aktuellen Daten der Fahrzeuge
260   if (microtime(true) > $timeCheckAllTrainErrors) {
261     consoleAllTrainsPositionAndFahrplan();
262     showFahrplan();
263     $timeCheckAllTrainErrors = $timeCheckAllTrainErrors +
          ↪ $timeCheckAllTrainErrorsInterval;
264   }
265   sleep($sleeptime);
266 }
```

A.2 globalVariables.php

```
1  <?php
2
3  $globalNotverzoegerung = 2; // Bremsverzögerung bei einer Notbremsung
4  $globalMinSpeed = 10; // Maximale Geschwindigkeit, wenn keine vorgegeben ist
5  $globalSpeedInCurrentSection = 60; // Maximale Geschwindigkeit im aktuellen Abschnitt
6  $globalFirstHaltMinTime = 20; // calculateFahrverlauf -> Zeit fürs Wenden...
7  $globalIndexBetriebsstelleFreieFahrt = 9999999999999;
8  $globalFloatingPointNumbersRoundingError = 0.0000000001;
9  $globalTimeOnOneSpeed = 20;
10
11 $useSpeedFineTuning = true;
12 $useMinTimeOnSpeed = true;
13 $errorMinTimeOnSpeed = false;
14 $slowDownIfTooEarly = true;
15 $useRecalibration = true;
```

Literatur

Maschek, U. (2013). Zugbeeinflussung. In *Sicherung des schienenverkehrs* (S. 184–212). Springer.

Richard, H. & Sander, M. (2011). *Technische mechanik. dynamik: Grundlagen - effektiv und anwendungsnah.* Vieweg+Teubner Verlag.

Wende, D. (2013). *Fahrdynamik des schienenverkehrs.* Springer-Verlag.