Entwurf und Umsetzung einer halbautomatischen Modelleisenbahnsteuerung

Manuel Huber

Abbildung 1: Eisenbahn-Sicherungstechnik im Wandel der Zeit

Neufahrn 2018

Inhaltsverzeichnis

1 Bedeutung der Eisenbahn-Sicherungstechnik

Eine der Besonderheiten des Systems Eisenbahn im Vergleich mit anderen Verkehrsmitteln ist der geringe Haftreibungsbeiwert. Einerseits ist dies vorteilhaft, da die geringe Haftreibung zu einem geringeren Energieverbrauch bei großen transportierten Massen und damit einer besonders großen Energieeffizienz führt. Andererseits sorgt die geringe Reibung auch für lange Bremswege, die beispielsweise auf signalisierten Hauptbahnen mit Höchstgeschwindigkeiten bis zu $160\frac{\text{km}}{\text{h}}$ gesetzlich mit maximal 1000m angegeben werden [1, §35]. Da der benötigte Bremsweg bei höheren Geschwindigkeiten nicht mehr vom Triebfahrzeugführer überblickt werden kann, ist ein Fahren auf Sicht in der Regel nicht mehr möglich. Stattdessen muss der Zug durch Sicherungsanlagen geschützt werden, die bei höheren Geschwindigkeiten daher unverzichtbar sind.

Zu Beginn des Eisenbahnwesens in Deutschland wurden verschiedene Signalisierungsverfahren angewandt, die zur Wahrung der Sicherheit unbedingt vom Zugpersonal beachtet werden mussten. Die immer weiter zusammenwachsenden Netze einerseits und die politische Lage andererseits machten eine Vereinheitlichung notwendig, die schließlich ab 1875 zu einer nahezu einheitlichen Signalordnung führte [2].

Ein Überfahren von Halt zeigenden Signalen war aber weiterhin möglich. Andere Sicherungsmittel wie Gleissperren oder Schutzweichen, die gegen das Überfahren durch Züge oder Zugteile schützen, wurden bis auf wenige Ausnahmen nicht in Hauptgleisen angewandt. Da auch einige Eisenbahnstrecken bis vor wenigen Jahren noch ohne jegliche Zugbeeinflussung an Signalen betrieben wurden [3, 4], kam es zu schweren Unfällen mit vielen Toten bei Zugzusammenstößen auf der freien Strecke durch Missachtung von Halt zeigenden Signalen, wie 2011 in Hordorf [5].

Um Unfälle besser verhindern zu können, kam es noch unter der Regie der Deutschen Reichsbahn zur ersten Entwicklung von elektrischen Zugbeeinflussungssystemen, der Induktiven Zugsicherung (Indusi), die heute als PZB/90 immer noch in erweiterter Form im Einsatz ist. Diese verhindert das Überfahren eines Halt zeigenden Signals mittels an der Strecke punkförmig angeordneter Elektromagnete, die induktiv einen Wahrheitswert (etwa den Zustand eines Signals) an das Triebfahrzeug übertragen können. Auf dieser Basis kann eine möglicherweise benötigte Zwangsbremsung oder Überwachung einer Maximalgeschwindigkeit eingeleitet werden [6].

Durch die heute immer höheren Geschwindigkeiten auf europäischen Gleisen ist aber auch diese Technik nicht mehr für alle Zwecke geeignet. Für Schnellfahrstrecken wurde

von Beginn an die Linienförmige Zugbeeinflussung (LZB) als alternatives System verwendet, da die herkömmliche Technik nicht die nötigen Vorankündigungsstrecken für die langen Bremswege bieten konnte. Auch dichtere Takte lassen sich durch eine Form der LZB ermöglichen, beispielsweise auf der S-Bahn-Stammstrecke München, die mit ungefähr 30 Zügen pro Stunde und Richtung die am dichtesten befahrene Eisenbahnstrecke Deutschlands ist. [7, 8]

Doch eine Fehlerquelle bleibt bis heute bestehen: Der Mensch ist immer noch in die Sicherungssysteme involviert und wird bei einigen Bauformen nicht genug von den technischen Systemen unterstützt. Mehrere Beispiele aus der jüngsten Vergangenheit, darunter am bekanntesten wohl die Zugkollision bei Bad Aibling am 09.02.2016, unterstreichen diesen Sachverhalt. Bei diesem Unfall wurde durch den Bediener aktiv eine Sicherheitsmaßnahme des Systems umgangen, ohne sich vorher vom Freisein der Gleisabschnitte zu überzeugen [9].

Doch egal, welches System auf einer Strecke eingesetzt wird, es finden immer die gleichen Grundprinzipien in unterschiedlich stark technisierten Formen Anwendung. Zu den einfachsten Möglichkeiten, auf Nebenbahnen Zugbetrieb durchzuführen, gehört der Zugleitbetrieb. Der Zugleiter muss alle Zugfahrten erlauben und erhält per fernmündlicher Übertragung Informationen über die aktuellen Positionen der Züge. Unter der Annahme, dass der Zugleiter keine Fehler macht und kein Triebfahrzeugführer ohne Erlaubnis auf die Strecke fährt, ist gewährleistet, dass jeder Zug immer seinen ganzen Gleisabschnitt zur freien Verfügung hat. [10, Modul 436.0002]

Auf Bahnhöfen, auf denen rangiert wird, muss aber auch das Streckengleis gegen ungewünscht rollende Waggons geschützt sein. Dazu dienen Flankenschutzeinrichtungen, die mit Handschlössern verschlossen werden. Der Zugführer hat einen Schlüssel bei sich, der auf der gesamten Strecke für bestimmte handstellbare Elemente (Weichen oder Gleissperren) nutzbar ist („Zugführerschlüssel“). Ein Zugverkehr auf dem Hauptgleis ist erst wieder erlaubt, wenn alle Flankenschutzmittel in Grundstellung verschlossen sind, der Schlüssel wieder beim Zugführer ist und dieser Zustand dem Zugleiter gemeldet wurde. [10, Modul 436.0004]

Abbildung 2: Überblick über die Gleisanlagen in Sternhaus-Ramberg. Das rechte Gleis ist das Hauptgleis. Die Weiche im Hintergrund ist in Grundstellung verschlossen.

Abbildung 3: Gleissperre in Sternhaus-Ramberg mit zwei Handschlössern. Der Zugführerschlüssel passt in das rote Handschloss. Im umgelegten Zustand wird ein Schlüssel freigegeben, der in das Handschloss der Weiche passt.

Diese einfache Form der Schlüsselabhängigkeit und allgemein das einfache Verfahren des Zugleitbetriebes erlaubt natürlich nur die Betriebsführung bei wenig Verkehrsaufkommen. Aber auch bei einem signalisierten Betrieb ist immer noch der Einsatz von Schlüsseln möglich, indem diese mittels einer Schlüsselsperre in Abhängigkeit mit dem restlichen mechanischen oder elektrischen Stellwerk gebracht werden. Ortsbediente Bereiche, in denen rangiert wird, müssen somit nicht aufwendig vom Stellwerk aus gesichert werden, sondern es kann das einfache Verfahren mit Schlüsselsperren angewandt werden [11].

2 Entwurf und Umsetzung einer halbautomatischen Modelleisenbahnsteuerung

2.1 Planung und Konzept

Die vorhandene Modelleisenbahnanlage Bundenthal [12] soll als Versuchsanlage für eine halbautomatische Steuerung des Zugverkehrs dienen. Die Anlage ist im Maßstab 1:160 umgesetzt.

Beim Aufbau des Versuchs sollen möglichst alle dauerhaften Veränderungen an der Anlagensubstanz ausgeschlossen werden, da Teile der Hardware im Gleisbereich anschließend wieder restlos entfernt werden müssen.

Um den Aufwand im Hardwareaufbau klein zu halten, wurde nur ein Ausschnitt des Gleisnetzes für den Versuch ausgewählt. Als sinnvoller Bereich wurden zwei der drei Bahnhofsgleise, das anschließende Ausziehgleis und ein daneben liegendes Abstellgeleis ausgewählt. Auf den anderen Gleisen wäre ein sinnvoller Betrieb nicht möglich, da ein automatisches Rangieren unter den Versuchsbedingungen technisch nicht möglich ist.

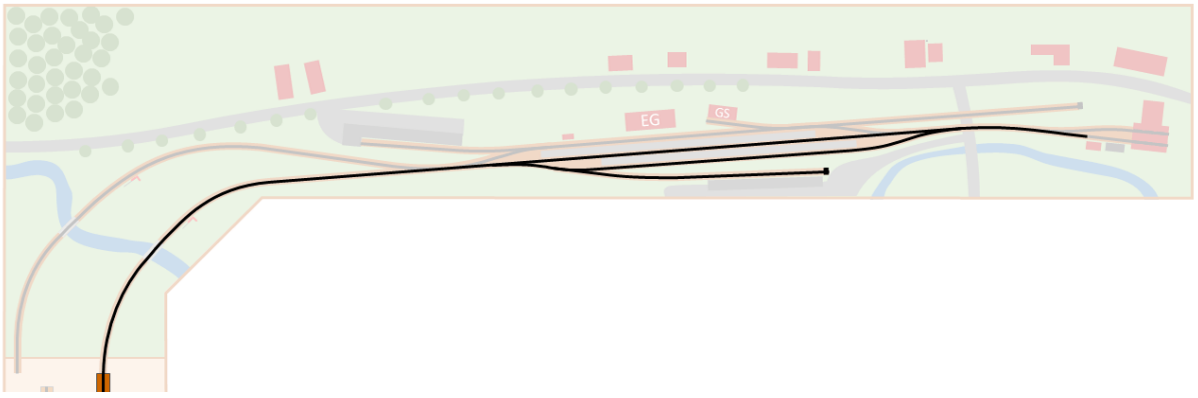

Abbildung 4: Versuchs-Gleisplan

Der Zugverkehr soll, wie bei den meisten heute in Europa verwendeten Systemen, im Raumabstand erfolgen [13, S. 39]. Vor jeder Zugbewegung wird ein Teil des in feste Abschnitte eingeteilten Gleisnetzes für die alleinige Verwendung des Zuges reserviert. Dies ist nur möglich, wenn sich derzeit kein Fahrzeug im Abschnitt befindet. Eine Einfahrt eines anderen Zuges in denselben Bereich muss durch die Stellwerkslogik ausgeschlossen werden, um den sicheren Betrieb zu gewährleisten.

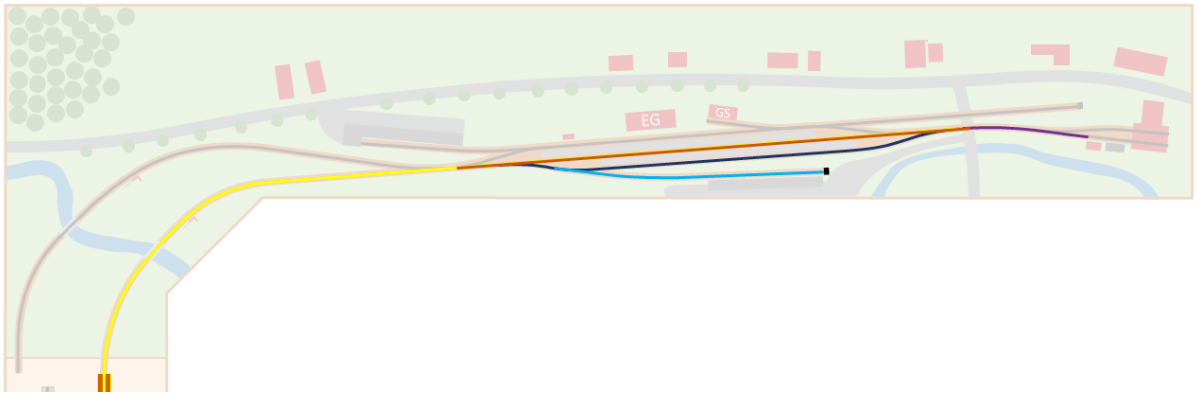

Abbildung 5: Block-Aufteilung im Versuch

Bei einigen alten Sicherungssystemen wurde zwischen Fahrstraßen in Bahnhöfen und Blockabschnitten auf der freien Strecke unterschieden. Bei diesem Versuch soll keine solche Unterscheidung stattfinden und alle Zugbewegungen als Fahrstraßen durchgeführt werden.

Pro Abschnitt werden am Anfang und Ende Lichtschranken installiert, die die Zugbewegungen überwachen. Direkt aneinander liegende Blöcke teilen sich ggf. auch eine gemeinsame Lichtschranke. Die Lichtschranken dienen nicht der unmittelbaren Ortung von Fahrzeugen im Gleisabschnitt, sondern ergeben erst eine mittelbare Ortung in Verbindung mit dem der Steuerungssoftware bekannten Belegungszustand (vgl. 2.2.2).

Das Steuerungsprogramm soll unter Angabe von Start- und Zielblock die Zugfahrt automatisch durchführen. Dazu zählt das Ermitteln und Einstellen eines sicheren Fahrwegs ohne Kollisionen mit anderen Fahrzeugen und ebenfalls die automatische Bewegung des ausgewählten Zuges. Dabei soll der Bediener das Programm auch ohne Kenntnis über die reale Infrastruktur bedienen können.

Auch in unerwarteten Situationen soll die Sicherheit so gut wie unter den Versuchsbedingungen möglich gewahrt bleiben. In diesem Bereich müssen aber aufgrund des fehlenden Platzes im Maßstab 1:160 Abstriche gemacht werden.

Die Arbeit der Software lässt sich als einfacher Regelkreis beschreiben: Die Stellwerkssoftware erhält Informationen über den Zustand der Freimeldesensoren. Auf Basis dieser Informationen wird der Besetztzustand der Gleise ermittelt. Ist eine Fahrstraße korrekt eingestellt, kann ein Fahrbefehl erteilt werden, der solange aufrecht erhalten bleibt, bis die Gleisfreimeldeanlage das vollständige Erreichen des Zielabschnitts meldet.

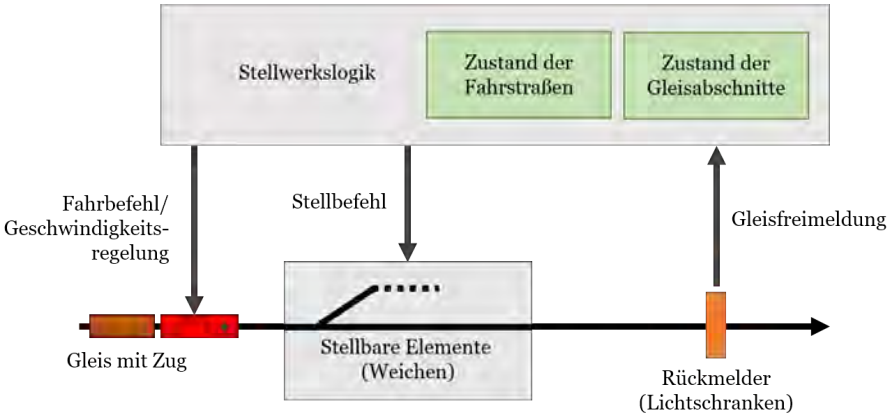

Abbildung 6: Regelkreis der Eisenbahnsicherungstechnik nach [13, S. 14], angepasst auf die Verhältnisse im Versuch

Es soll auch eine Schnittstelle zu anderen Stellwerksinstanzen implementiert werden. Dies wird für einen realistischeren Einsatz benötigt, da auch in Realität kein einzelnes Stellwerk das ganze Streckennetz auf einmal kontrollieren kann. Über diese Schnittstelle sollen Züge an andere Stellwerke – mit einem vorher festgelegten Fahrtziel – übergeben und eine insgesamt gesicherte Fahrstraße aufgebaut werden.

2.2 Elektrische Vorarbeiten

Die Steuerung erfolgt von einem Raspberry Pi aus. Dieser wurde für die Ansteuerung ausgewählt, da er sowohl durch ein vollständiges Linux eine flexible Programmierung in Hochsprachen erlaubt als auch genügend I/O-Ports bereitstellt, über die mit externen Komponenten kommuniziert werden kann. Die einzelnen angesteuerten Elemente werden im Folgenden näher erläutert.

2.2.1 Stellen der Weichen

Die auf der Versuchsanlage bereits vorhandenen elektrischen Weichenantriebe werden mit einer Spannung von 12V angesteuert und haben für jede der beiden Richtungen (im Folgenden als gerade oder abzweigend bezeichnet) je einen Eingang. Um die Endlage zu erreichen, reicht ein kurzer Impuls ($t < 0,5\text{s}$). Diese Impulslänge sollte nicht zu stark überschritten werden, da sonst der Weichenantrieb beschädigt werden kann.

Die Computersteuerung übernimmt diese Ansteuerungsmethode. Zur Versorgung der Antriebe mit 12V ist der Raspberry Pi mittels einem I^2C-Portexpander mit einer Relaiskarte verbunden. Mit einem einzelnen Umschaltrelais lässt sich nur die Richtung der

Weiche einstellen; der Impuls muss durch ein weiteres Einschaltrelais erzeugt werden. Da der gesamte Schaltstrom von 10 Weichen gleichzeitig den zulässigen Maximalstrom des verwendeten Netzteils übersteigen würde, werden je zwei Weichen zu einer Schaltgruppe zusammengefasst und erhalten ein gemeinsames Relais zur Impulssteuerung.

Unter Voraussetzung einer durchlaufenden Nummerierung der Weichen und einer Relaisanzahl von 16, lassen sich für die Weiche w die Nummer des Gruppen-Schaltrelais s wie folgt ermitteln:

$$s = 16 - \left\lfloor \frac{w-1}{2} \right\rfloor$$

Zum Sparen von Energie werden die Ansteuerungsrelais von der Software nur während des Schaltvorgangs aktiviert. In der Schaltsequenz für eine Weiche müssen erst beide Richtungsrelais der Gruppe in die gewünschte Position gebracht werden. Anschließend wird das Gruppenschaltrelais aktiviert und beide Weichen werden in die gewünschte Position bewegt. Beim Abschalten wird in umgekehrter Reihenfolge verfahren und die Weichen verbleiben in ihrer Position. Der Zustand der Richtungsrelais wird durch den Treiber gespeichert und vor dem nächsten Aktivierungsvorgang des Gruppenrelais wiederhergestellt.

Im Versuch findet keine weitergehende Überprüfung der korrekten Endlage der Weichen statt; ein externes Verstellen während der Verwendung der Steuerungssoftware ist daher nicht möglich. Im Modell wirken auch keine großen Kräfte während der Überfahrt eines Zuges, daher kann hier auf eine dauerhafte Überwachung verzichtet werden.

2.2.2 Positionserkennung mit Lichtschranken

Die für die Versuche verwendeten Lichtschranken wurden auf Basis einer modellbahntauglichen Infrarot-Lichtschranke [14] in Anpassung an die bei Raspberry Pis üblichen Betriebsspannungen (5V und 3,3V) in kompakter Bauweise selbst auf Lochrasterplatinen gebaut.

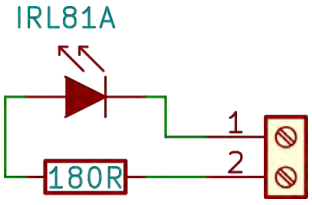

Abbildung 7: Lichtschranke-Sender

Der Sender wird über Pin 2 an +5V Gleichspannung und mit Pin 1 an Masse angeschlos-

sen und ist eine daueraktive Infrarot-LED. Die 5V-Betriebsspannung wurde gewählt, da der vom Raspberry Pi über 3,3V maximal bereitgestellte Strom von 50mA bereits mit wenigen LED überschritten wäre.

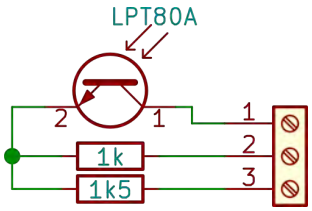

Abbildung 8: Lichtschranke-Empfänger

Der Empfänger wird über Pin 1 an +3,3V und über Pin 3 an Masse angeschlossen. Damit ist der Fototransistor aktiv.

Die Empfänger liefern über einen Datenpin (Pin 2) den aktuellen Zustand (geöffnet oder geschlossen) an den Controller. Eine logische Null am Datenpin zeigt den geschlossenen Zustand an.

Dieses Signal ist für die Zugortung nicht optimal, da wichtige Informationen über den Zustand, dabei vor allem Informationen über den aktuell erfassten Zug und die Bewegungsrichtung, fehlen. Diese müssen softwareseitig ergänzt werden, was zwar unter dem Sicherheitsaspekt nicht die optimale Lösung, aber für die Versuchsreihe ausreichend ist.

Im Versuch wurden die Lichtschranken in der Höhe so angeordnet, dass Wagenkästen gezählt werden. Dies erlaubt nicht den Einsatz aller Wagentypen, da manche Wagentypen wie Personenwaggons durch Anbauteile oder Fenster bei der Zählung störende Impulse erzeugen. Im Versuch werden aus diesem Grund nur Güterwaggons eingesetzt.

2.3 Implementierung der Steuerungssoftware

Die Ansteuerungssoftware wurde in der Programmiersprache C# entwickelt und läuft auf dem schon unter Hardware (Abschnitt 2.2) näher erläuterten Raspberry Pi unter der Mono-Laufzeitumgebung. Diese stellt den Großteil der .NET-Klassenbibliothek für das freie Betriebssystem Linux bereit. Das Programm wird über das Remote-Desktop-Protocol ferngesteuert. Eine Client-Server-Architektur zwischen Bedienanwendung und Stellwerkslogik wäre aber auch als Alternative denkbar, wurde aber nicht umgesetzt.

2.3.1 Modulare Architektur

Das Stellwerksprogramm ist in modulare Komponenten aufgeteilt: Die Infrastrukturtreiber sind grundsätzlich austauschbar und werden vom Hauptprogramm über definierte Schnittstellen angesprochen. Im Versuch kommen zwei Treiber zum Einsatz:

- **SensorDriver**: Der Treiber für die Lichtschranken. Er reagiert auf Änderungen an den Dateneingangspins und ermöglicht das Warten auf ein eintretendes Ereignis.
- **PointDriver**: Übernimmt die Kommunikation mit der Relaiskarte über den I^2C-Bus und die Portexpander zur Ansteuerung der Weichen.

Das Programm benötigt zur korrekten Arbeit Informationen über die Infrastruktur (Gleisnetz und Lichtschranken) und die vorhandenen Züge. Diese Infrastrukturdaten liegen im Json-Format (vgl. A.2) vor und werden vom Hauptprogramm eingelesen; die Daten über die Züge werden bei jedem Programmstart abgefragt.

Die gemeinsam genutzte Bibliothek **Shared** stellt ein Datenmodell für die Infrastruktur und die gesamte Stellwerkslogik bereit.

Die Züge werden nicht direkt von der Stellwerkssoftware aus gesteuert. Jeder Zug hat eine eigene Zugcontroller-Instanz (`trainunit.exe`), der über eine Netzwerk-Verbindung vom Stellwerksprogramm über eine Fahrerlaubnis informiert wird.

Ebenso mag ein monolithisches Stellwerk im Versuchsbetrieb sinnvoll erscheinen, es müssen aber auch Schnittstellen zur Kommunikation mit anderen automatischen Stellwerken vorhanden sein. Für solche Aufgaben ist in Zukunft bei Sicherungssystemen die Verwendung von IP-basierten Netzwerken vorgesehen [15], auch im Versuch werden daher eigene, auf TCP aufbauende Protokolle verwendet (Protokoll-Referenz vgl. A.3).

Im Versuch laufen die verschiedenen Komponenten aber nicht örtlich getrennt auf unterschiedlichen Hosts; die Unterscheidung erfolgt allein über verschiedene Netzwerkports.

2.3.2 Ermitteln der Fahrstraße

Das Gleisnetz kann als ungerichteter Graph visualisiert werden. Blöcke und Weichen entsprechen den Knoten, ihre Verbindungen werden durch Kanten dargestellt. Der Graph darf keine ohne Richtungswechsel befahrbare Zyklen besitzen. Blöcken wird intern eine Richtung zugewiesen, indem sie durch Anfangs- und Endübergänge definiert werden; alle Blöcke im Graph müssen gleich gerichtet sein, damit die Fahrtrichtungsbestimmung korrekt funktioniert.

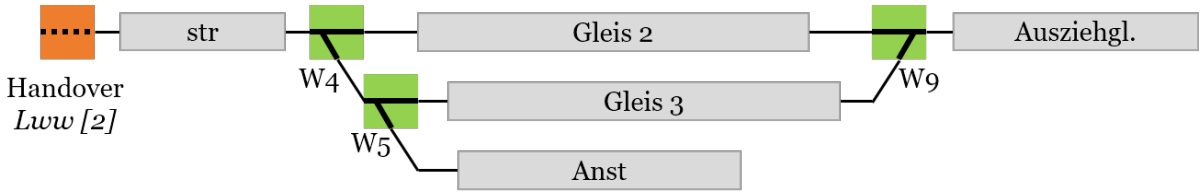

Abbildung 9: Gleisnetzgraph der Versuchsanlage

Als Methode zur Fahrstraßensuche wird eine modifizierte Breitensuche verwendet. An Weichen wird immer in die ohne Richtungswechsel möglichen Richtungen fortgefahren. Wenn von der Weichenspitze ausgegangen wird, stehen also zwei Möglichkeiten zur Verfügung, sonst nur eine. Es wird immer zuerst der gerade und anschließend der abzweigende Weg geprüft.

Natürlich können bereits von einem Zug belegte oder für eine andere Fahrstraße reservierte Abschnitte nicht noch einmal belegt werden, daher werden diese bei der Suche ignoriert.

Richtungswechsel werden beispielsweise bei Fahrten von Gleis 2 zu Gleis 3 benötigt und können nur auf Gleisabschnitten stattfinden. Um dies zu ermöglichen, darf jeder Abschnitt oder jede Weiche maximal zwei Mal in verschiedenen Richtungen durchlaufen werden, andernfalls ergibt sich ein zyklischer Weg. Knoten werden in der Implementierung daher erst als abschließend besucht markiert, wenn sie bereits genau zwei Mal besucht wurden.

Zur Optimierung des gefundenen Weges wird, anders als bei einer normalen Breitensuche, eine Prioritätswarteschlange verwendet. Weichen, die immer eine höhere Priorität erhalten als Gleisabschnitte, werden so bevorzugt abgearbeitet. Dies führt dazu, dass die Suche einen Weg mit einer geringen Anzahl von Gleisabschnitten bevorzugt, im Gegensatz zu einem Weg mit möglichst wenigen Elementen insgesamt.

Dieser Effekt tritt bei einer kleinen Anordnung wie bei der Versuchsanlage noch nicht auf, lässt sich aber bei größeren Weichenstraßen wie der im Folgenden gezeigten erkennen. Die Fahrstraße soll jeweils vom Startabschnitt bis nach Gleis 2 ermittelt werden.

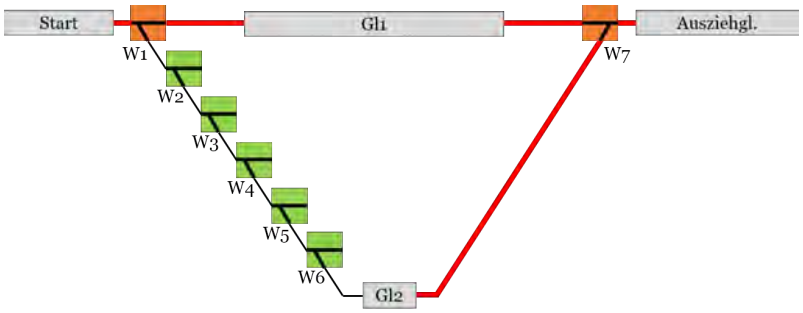

Abbildung 10: Suche ohne Prioritätswarteschlange: Fahrstraße mit möglichst wenigen Fahrwegelementen (7 bei doppelter Zählung von W7), aber mit einem eigentlich nicht benötigten Richtungswechsel

Der hier gefundene Weg enthält zwar die kleinste Anzahl an Fahrwegelementen, ist aber im Betrieb nicht so effizient, da beide Bahnhofsköpfe in die Bewegung miteinbezogen sind und so keine zweite Einfahrt gleichzeitig möglich ist. Ferner verlängern Richtungswechsel die Fahrtzeit, da der Zug erst abbremsen und anschließend wieder beschleunigen muss.

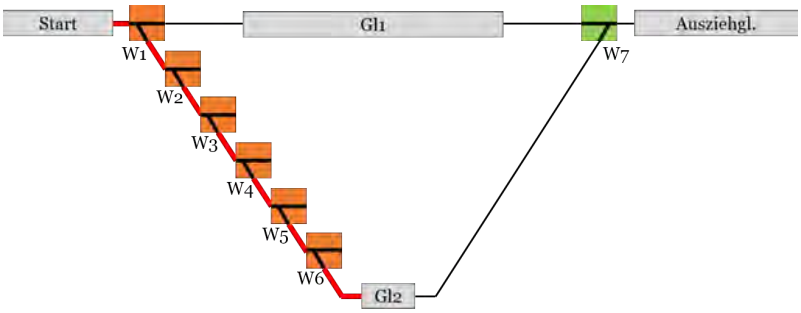

Abbildung 11: Suche mit Prioritätswarteschlange: Direkteste Fahrstraße, dafür mit mehr Fahrwegelementen (8)

Wenn Weichen gegenüber Gleisabschnitten priorisiert werden, ist der Weg für den Betrieb sinnvoller, da der Richtungswechsel wegfällt und über den rechten Bahnhofskopf grundsätzlich eine zweite Einfahrt möglich ist. Die Implementierung des beschriebenen Algorithmus findet sich in der Klasse `PathFinder` (vgl. Listing 15).

2.3.3 Automatische Durchführung der Zugfahrten

An der Modelleisenbahnanlage kommt eine DCC-Digitalsteuerung z21 zum Einsatz. Diese bietet eine UDP-API über ein Ethernet-Interface an. Unter Verwendung der offiziellen Schnittstellendokumentation [16] und in Anlehnung an eine bestehende C#-Bibliothek [17] entstand eine eigene Bibliothek, die die benötigten Funktionen bereitstellt. Diese wird in der Train Unit verwendet.

Um als Basis für die automatische Steuerung festzustellen, wo sich der bewegte Zug gerade befindet, werden die Lichtschranken mit Hilfe von Zählern ausgewertet. Der Zähler hat drei Betriebsmodi:

- **Fehler-Modus**: Jegliche Veränderung des Zustandes führt zu einer `TrafficException`. Dieser Modus ist nach Programmstart der Standard-Modus für alle Lichtschranken.
- **Zähler-Modus**: Es werden entweder beide oder nur einer der beiden Signaltypen *Öffnen* oder *Schließen* gezählt. Der Zählerstand kann anschließend ausgewertet werden.
- **Zielanzahl-Modus**: Es wird wie im Zähler-Modus gezählt. Zusätzlich kann bis zum Erreichen eines bestimmten Zielwertes asynchron gewartet werden.

Nachdem auf Benutzeranweisung, wie in 2.3.2 beschrieben, die günstigste Fahrstraße ermittelt wurde, wird diese als „Reserviert“ markiert und die Weichen laufen in die gewünschte Endlage. Anschließend wird die folgende Sequenz abgewickelt:

1. Alle Lichtschranken, die noch nicht für andere Fahrstraßen konfiguriert wurden, werden in den Fehler-Modus versetzt. Dies ist eigentlich nicht zwingend nötig, wird aber aus Sicherheitsgründen wiederholt.
2. Alle Lichtschranken in der Fahrstraße werden in den Zähler-Modus versetzt und damit der Zähler auf 0 zurückgesetzt. Die Lichtschranken, die die Fahrstraße begrenzen, werden im Fehler-Modus belassen.
3. Die vorletzte Lichtschranke an der Fahrstraße wird in den Zielanzahl-Modus versetzt, der Zielwert entspricht der doppelten Wagenanzahl.
4. Der Train Unit wird seine Fahrerlaubnis mitgeteilt und so lange gewartet, bis an der vorletzten Lichtschranke der Zielwert erreicht wird. Anschließend wird die Fahrerlaubnis des Zuges aufgehoben.
5. Bei allen Zähler-Lichtschranken wird die gezählte Anzahl mit der Zielanzahl verglichen. Unterscheiden sich die Werte, wird eine `TrafficException` geworfen, da auf der Strecke Wagen verloren gegangen sein müssten.

Abbildung 12: Beispiel einer Lichtschrankenkonfiguration einer Zugfahrt. Orange = Fehler-Modus, Blau = Zielanzahl-Modus, Grün = Zähler-Modus

Sollte während der Programmausführung eine `TrafficException` geworfen werden, werden alle von der Software kontrollierten Züge sofort angehalten. Dies ist bei Verlust von Waggons oder einer anderen unerlaubten, nicht von der Software kontrollierten Zugbewegung der Fall.

Sind für das Erreichen eines bestimmten Zielblocks ein oder mehrere Richtungswechsel erforderlich, wird der Fahrweg wie mehrere getrennte Fahrstraßen behandelt und diese jeweils nach der oben beschriebenen Abfolge abgewickelt.

Die Implementierung der Stellwerkslogik findet sich nahezu vollständig in der Klasse `MovementLogic` (vgl. Listing 26).

2.3.4 Übergabe von Zügen an andere Stellwerksbezirke

Vor dem Beginn der Fahrstraßenermittlung kann der Benutzer einen noch freien Zielblock im Zielbezirk auswählen. Die Kommunikation erfolgt über das Interlocking Protocol (vgl. A.3), darüber wird auch dem Zielstellwerk das Fahrtziel mitgeteilt.

Fahrten, die in anderen Stellbezirken enden, werden ähnlich wie Fahrten innerhalb des eigenen Bezirks behandelt. Die Lichtschrankenkonfiguration unterscheidet sich aber in einigen Details.

Beim Übergeben eines Zuges wird die Lichtschranke am Rand der Fahrstraße, die an der Übergabestelle liegt, nicht in den Fehler-, sondern in den Zielanzahlmodus versetzt. Die vorletzte Lichtschranke wird nur in den Zähler-Modus versetzt und erhält keinen Sonderzustand mehr. Wenn der Zug diese letzte Lichtschranke vollständig passiert hat, wird er nicht angehalten, sondern es wird nur die Reservierung der Fahrstraße im eigenen Bezirk aufgehoben.

Wenn von einem anderen Stellwerk ein Zug übernommen wird, so wird die erste Lichtschranke am Rand der Fahrstraße, der an der Übergabestelle liegt, nicht in den Fehler-, sondern in den Zielanzahlmodus versetzt. Ansonsten wird analog zu normalen Fahrstraßen verfahren.

2.4 Bewertung des umgesetzten Sicherungssystems

2.4.1 Sicherungstechnische Mängel im Vergleich mit bestehenden Sicherungssystemen

Nach §2 (2) Eisenbahn-Bau- und Betriebsordnung (EBO) kann bei Sicherungssystemen „von den anerkannten Regeln der Technik [...] abgewichen werden, wenn mindestens die gleiche Sicherheit wie bei Beachtung dieser Regeln nachgewiesen ist“ [1, §2]. Dies ist natürlich bei einer Versuchssteuerung auf einem Modell nicht möglich.

Durch die vereinfachten Versuchsbedingungen sind einige Einschränkungen vorgegeben. Dazu gehören die automatischen Zugbewegungen, die auf (näherungsweise) instantanen Geschwindigkeitswechseln ohne träger Masse basieren. Eine realistische Geschwindigkeitsänderung mit langen Bremswegen findet nicht statt. Jedes Fahrzeug müsste Informationen über das eigene Fahrverhalten kennen und diese auf die vorgegeben Höchstgeschwindigkeiten anwenden. Die ebenfalls benötigten Informationen zur aktuellen Ist-Geschwindigkeit sind technisch im Modell nicht ermittelbar. Auf einer Versuchsanlage könnte hier also maximal eine gute Annäherung erreicht werden.

Fahrzeuge ohne Kontrolle durch das System können nicht automatisch angehalten werden, um Kollisionen zu verhindern. Dieser Aspekt trifft auch bei bisherigen Systemen zu, dort gibt es aber durch etablierte und vorgeschriebene Standards zumindest den Ansatz einer Lösung. Auch aktiv verwendete Flankenschutzmittel können dieses Problem beheben.

Im Versuch findet auch keine Überprüfung auf systemfremde Fahrzeuge, Personen oder Umwelteinflüsse im Gleisbereich statt. Während sich Bahnübergänge grundsätzlich einfach in das System integrieren ließen und die freie Strecke auch heute nicht überwacht wird, stellt die Sicherung der Reisenden ein erheblich größeres Problem dar. Dieses Problem tritt bei allen vollautomatischen Schienenverkehrsmitteln auf und kann auf unterschiedliche Weise gelöst werden. Eine Möglichkeit stellt sicherlich das Verwenden von gegenüber dem Gleisbereich abgeschirmten Bahnsteigen dar, wie es beispielsweise bei der Transportbahn im Flughafen München verwendet wird. Eine andere Möglichkeit ist der Einsatz eines Triebfahrzeugführers ausschließlich zur Überwachung und Abfertigung der Reisenden am Bahnsteig.

Einer der größten Mängel im Vergleich mit bestehenden Sicherungssystemen ist aber das Fehlen von Flankenschutzeinrichtungen wie Schutzweichen oder Gleissperren. Das hier entwickelte System schützt zwar weitgehend vor bewegten Zügen mit Lokomotive, wenn

aber nur einzelne Wagen abgestellt sind, kann es zu gefährlichen Zuständen kommen. Diese werden derzeit von der Steuerungssoftware nicht angewendet, da es kein Testszenario mit Schutzweichen auf der Versuchsanlage gibt; funktionsfähige Gleissperren sind im Maßstab 1:160 nicht mit vertretbarem Aufwand realisierbar.

Bei dem Versuch fehlt aufgrund der einfachen automatischen Zugsteuerung ein zweiter Regelkreis zur Geschwindigkeitsregelung des Triebfahrzeugs. Die Überwachung des Bedieners kann natürlich bei einem zentralen automatischen System entfallen, die Geschwindigkeit muss aber auch fahrzeugseitig überwacht und gesteuert werden. Nur so könnte z.B. bei einem Ausfall der Verbindung zur Kontrollstelle der Zug sicher zum Stehen gebracht werden.

2.4.2 Verbesserungsmöglichkeiten im Hard- und Softwarebereich

Neben den Sicherheitsmängeln gibt es weitere Einschränkungen der Praktikabilität. Ein Bewegen von Fahrzeugen ohne Kontrolle des Systems ist nicht möglich. Automatisches Rangieren ist komplexer als ein reiner automatischer Zugverkehr und sollte daher noch von Menschen gesteuert werden. Das System könnte einen Rangiermodus erhalten, in dem ein Fahrzeug manuell in einem bestimmten, von Gleissensoren eingeschlossenen Bereich bewegt werden kann.

Die Software überprüft auch nicht, ob ein bestimmter Zug in den Zielabschnitt passt. Ist der Zug länger als der Abschnitt, wird das gesamte Sicherungssystem blockiert, da die Fahrstraße nicht vollständig frei ist und die Software den Verbleib des Rest-Zuges nicht überprüfen kann.

Im Versuch waren einige Lichtschranken äußerst fehleranfällig und konnten bereits durch minimale Veränderungen im Umgebungslicht oder der Ausrichtung gestört werden. Erst recht im realen Bahnbetrieb wären Lichtschranken leicht zu verschmutzen und damit außer Betrieb zu setzen. Sowohl im Versuch zur Verbesserung der Stabilität als auch für einen realen Einsatz wären daher andere Ortungsmethoden besser geeignet.

Allgemein ist die Stabilität der Steuerungssoftware zu gering, um dauerhaft einen reibungslosen Betrieb zu garantieren. Wenn während des Programmablaufs eine Exception geworfen wird, egal ob eine interne `TrafficException` oder eine andere, verbleibt das Programm in einem undefinierten Zustand. Die Möglichkeiten, manuell einzugreifen, müssten daher noch erheblich verbessert werden, aktuell bleibt nur ein Neustart des Programms. Dies ist aber nur zielführend, wenn sich der ganze Zug noch in einem einzi-

gen Gleisabschnitt befindet, sonst muss mit der externen manuellen Steuerung, die nicht gleichzeitig mit der Software eingesetzt werden kann, eingegriffen werden. Einmal reservierte Fahrstraßen können nicht mehr freigegeben werden, falsch angelegte Züge weder editiert noch entfernt werden. Eine Netzwerkkommunikation zwischen Bedienanwendung und Stellwerkslogik anstelle der aktuellen Kombination in einer Anwendung, würde möglicherweise zur Verbesserung der Stabilität beitragen, daneben müssten die manuellen Eingriffsmöglichkeiten stark ausgebaut werden.

Abschließend ist festzustellen, dass aufgrund der oben genannten sicherungstechnischen Mängel und anderweitigen Problemen ein Praxiseinsatz des Systems nur mit großem Aufwand und vielen Verbesserungen erfolgen könnte und damit die Bedingung nach §2 EBO nicht erfüllt ist.

3 Ausblick auf zukünftige Entwicklungen

Die Entwicklung der Sicherungstechnik bleibt aber natürlich nicht bei den heute in der Fläche vorhandenen Technologien stehen, sondern entwickelt sich kontinuierlich weiter. Heutige Stellwerke und Triebfahrzeuge sind mit immer mehr Digitaltechnik ausgestattet, es spricht also nichts dagegen, diese auch für die Sicherungstechnik zu verwenden.

Ein Ansatz, der aktuell bereits auf einigen Strecken zum Einsatz kommt, ist ETCS (European Train Control System). Europaweit vereinheitlicht soll es den grenzüberschreitenden Bahnverkehr erleichtern und den flexibleren Einsatz von Triebfahrzeugen ermöglichen. ETCS ist dabei ein eigenes Sicherungssystem und kann nationale Systeme entweder ergänzen oder vollständig ersetzen. Neben der Strecke müssen in der Regel auch alle Fahrzeuge mit ETCS-Bordeinrichtungen ausgestattet werden. Die Ausrüstung kann dabei in verschiedenen Stufen erfolgen [18]:

- **Level 0**: Ein ETCS-Fahrzeug befindet sich auf einer Strecke ohne oder mit nicht funktionalem Sicherungssystem, daher kann nur eine feste Maximalgeschwindigkeit überwacht werden.
- **Level NTC**: Es ist ein nationales Sicherungssystem vorhanden. Die Fahrzeuge müssen neben der ETCS-Bordausrüstung zusätzlich mit Modulen für dieses nationale System ausgestattet sein, die mit den ETCS-Modulen über definierte Schnittstellen Daten austauschen. Die Sicherheit ist damit äquivalent zum nationalen System.
- **Level 1**: Über ETCS wird eine punktförmige Übertragung zum Zug über sogenannte

Eurobalisen (Antennen im Gleisbett, die entweder statische oder dynamische Daten senden) realisiert. Fahrzeugseitig wird damit eine dauerhafte Geschwindigkeitsüberwachung möglich, die auch das Überfahren von Halt zeigenden Signalen verhindert.

- **Level 2**: Die Kommunikation vom Zug zur Streckenausrüstung erfolgt kontinuierlich per Funk (sog. Euroradio). Die Gegenstelle ist das RBC (Radio-Block-Centre), das mit nationalen Stellwerks- und Gleisfreimeldesystemen kommuniziert. Die Zugposition wird vom Zug selbst auf Basis der erkannten Balisen-Positionen ermittelt und an das RBC übermittelt. Im Level 2 können stationäre Signale entfallen.
- **Level 3**: Die Gleisfreimeldung erfolgt nur noch auf Basis der von den Zügen bereitgestellten Positionsinformationen. Daneben muss der Zug selbstständig überprüfen, ob der Zugverband noch vollständig ist und diese Information an das RBC melden. Durch den Verzicht auf stationäre Gleisfreimeldeanlagen kann die Blocklänge wesentlich kleiner ausfallen oder eine feste Blockaufteilung kann ganz entfallen (sog. Moving Block). Damit übernimmt ETCS nahezu die vollständige Funktion eines Stellwerks. Stationäre Signale sind im Level 3 nicht vorgesehen.

Bis einschließlich ETCS Level 2 wird von ETCS nur die Kommunikation zwischen Streckenausrüstung und Fahrzeug übernommen. Die Positionserkennung und die Zugvollständigkeitsprüfung wird nicht von ETCS übernommen, sondern von konventionellen Block- und Stellwerkssystemen.

Ab ETCS Level 1 kommt neben den stationären Signalen Führerstandssignalisierung zum Einsatz. Dabei wird auf dem DMI (Driver-Machine-Interface, Anzeigenelement im Führerraum) der Zustand des nächsten Signals angezeigt. Ein Vorteil der Führerstandssignalisierung ist, dass die Signalbegriffe digital schon früher an das Triebfahrzeug übertragen werden können, als dies mit Lichtzeichen praktikabel ist. Bei hohen Geschwindigkeiten nimmt die Erkennbarkeit von Lichtzeichen ab, auch hier ist die Führerstandssignalisierung auch heute schon unverzichtbar. Nicht zuletzt können durch individuelle Signalisierung für jeden Zug Unfälle vermieden werden. Im Jahr 2003 nahm ein Triebfahrzeugführer am Bahnhof Neufahrn (b Freising) fälschlicherweise ein Signal als für seinen Zug geltend wahr, was zu einer Zugkollision führte [19].

ETCS ist dabei grundsätzlich immer noch so aufgebaut, dass ein Triebfahrzeugführer auf dem Fahrzeug vorhanden ist. Es ist aber auch die Möglichkeit vorgesehen, über ETCS automatischen Zugbetrieb durchzuführen. Diese als „ATO over ETCS“ bezeichnete Erweiterung soll laut DB in Deutschland ab 2021 in Hamburg auf einem Teilstück der S-Bahn-Linie S21 getestet werden, wobei auch ein kurzes Teilstück vollautomatisch ohne

Fahrgäste und anwesendem Triebfahrzeugführer gefahren werden soll [20].

ETCS wird bis jetzt nur auf wenigen Strecken Deutschlands eingesetzt, darunter die im Jahr 2017 eröffneten, neu gebauten Abschnitte der Strecke München – Berlin. Die Gesamtkosten der Neubaustrecke Nürnberg – Halle/Leipzig (Projekte VDE 8.1/8.2) betrugen ca. 7 Mrd. € [21], davon wurden ca. 120 Mio. € für die leit- und sicherungstechnische Ausrüstung (darunter auch ETCS L2oS, Level 2 ohne Signale) verwendet [22]. Das dabei erreichte Verhältnis der sicherungstechnischen Kosten zu den Gesamtkosten ist dabei wesentlich kleiner als bei der Versuchsanlage mit etwa 100€ Aufwand für die Sicherungstechnik, was wahrscheinlich durch die verhältnismäßig geringeren Modellgleisbaukosten begründet ist.

A Anhang

A.1 Hardwareaufbau

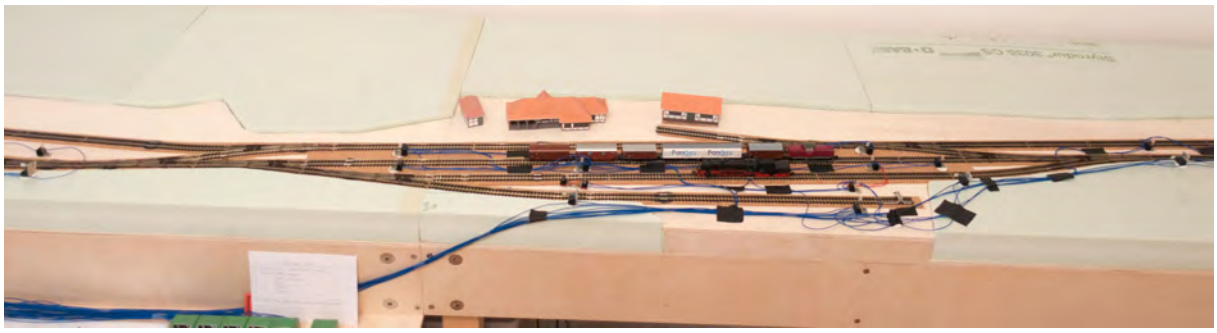

Abbildung 13: Überblick über die Versuchsanlage

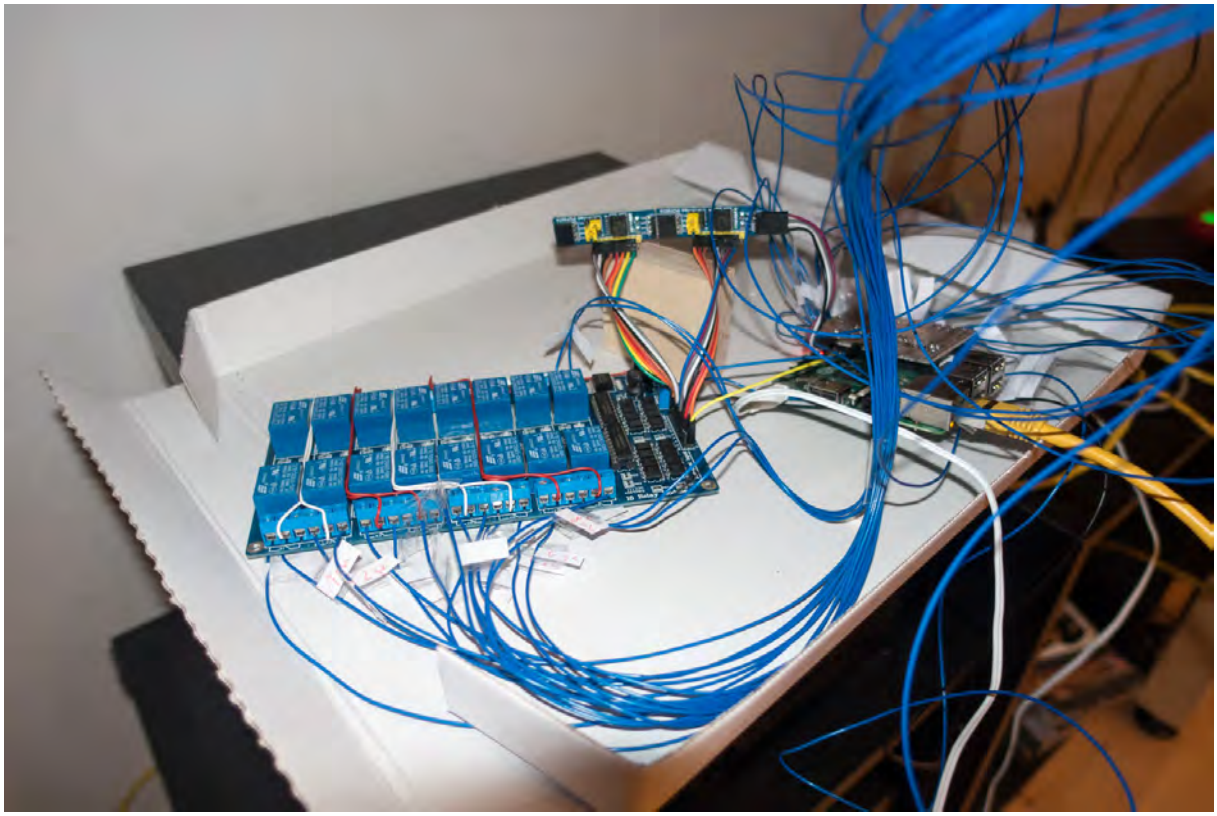

Abbildung 14: Hardware zur Weichenansteuerung mit Relaiskarte und I^2C-Portexpander.

A.2 JSON-Dateiformat

Die Programmdaten liegen im Ordner `data/` als Dateien im JSON-Dateiformat vor. Zusätzlich zur normalen JSON-Syntax sind aufgrund der verwendeten Parser-Bibliothek `Json.NET` auch Kommentare verwendbar.

Fahrzeugdaten

Für jedes Triebfahrzeug wird eine Datei im Ordner `data/locos/` angelegt. In diesen Dateien werden die Fahrzeugdaten für die DCC-Steuerung gespeichert; sie werden von der Train Unit (`trainunit.exe`) eingelesen.

```
{
    "Name": "V100",
    "DccAddress": 4, /* DCC-Lokadresse */
    "DccPrepareFunctions": [0, 4], /* DCC-Funktionen, die zu Beginn
       aktiviert werden. 4 = Deaktivieren der Massenträgheitssimulation.
        */
    "DccMaxFs": 40 /* Höchstgeschwindigkeit (0 - 128) */
}
```

Infrastrukturdaten

Die Informationen zum Gleisnetz sind in der Datei `layout.json` enthalten. Diese ist aufgeteilt in die Blockdefinitionen, Weichendefinitionen und die Definition der Verbindungen zwischen diesen.

Bei Weichen stehen die Anschlüsse `Entry`, `OutStraight` (gerades Gleis) und `OutDiverging` (abzweigendes Gleis) zur Verfügung. Bei Blöcken sind die Anschlüsse `Begin` und `End` verwendbar.

Der `Type` des Blocks muss aus den folgenden Werten gewählt werden:

- 0: Bahnhofsgleis (Station)
- 1: Streckengleis (Line)
- 2: Übergabe an ein anderes Stellwerk (Handover)

```
{
  "Blocks": [
    {
      "Type": 1,
      "Name": "Strecke"
    },
    {
      "Type": 0,
      "Name": "GleisA"
    },
    {
```

```
12        "Type": 0,
13        "Name": "GleisB"
14      }
15    ],
16    "Points": [
17      {
18        "Name": "W1",
19        "Id": 1
20      }
21    ],
22    "Connections": [
23      {
24        "src": "Strecke",
25        "src_dir": "End",
26        "dest": "W1",
27        "dest_dir": "Entry",
28      },
29      {
30        "src": "W1",
31        "src_dir": "OutStraight",
32        "dest": "GleisB",
33        "dest_dir": "Begin",
34      },
35      {
36        "src": "W1",
37        "src_dir": "OutDiverging",
38        "dest": "GleisA",
39        "dest_dir": "Begin",
40      }
41    ]
42  }
```

Ebenfalls zu den Infrastrukturdaten gehört die Zuordnung von Blöcken zu den Sensor-Eingängen der Lichtschranken. Diese ist in der Datei `sensors.json` enthalten. Der Eingang wird als Hardware-Pinnummer des Raspberry Pi eingegangen. `_b` und `_e` sind Suffixe für Beginn und Ende eines Blocks. Für jeden Block muss eine Lichtschranke für den Anfang und das Ende vorhanden sein. Einzige Ausnahme sind Blöcke mit dem Typ `Handover`.

```
1  {
2      "ZAK_b": 8,
3      "ZAK_e": 10,
```

```
4     "str_b": 10,
5     "str_e": 12
6 }
```

A.3 Netzwerkprotokolle

In der Steuerungssoftware kommen zwei simple Netzwerkprotokolle aufbauend auf TCP zur Verwendung. Befehle und Antworten werden als Zeichenketten mit UTF-8-Kodierung ausgetauscht.

Interlocking Protocol

Dieses Protokoll wird für die Kommunikation zwischen verschiedenen Stellwerksinstanzen verwendet. Jede Instanz hat einen Server, der von anderen Instanzen kontaktiert werden kann. Die Portnummer ist $35000 + StellwerksId$. Es stehen die folgenden Kommandos zur Verfügung:

- `list;end` gibt eine Liste aller unbelegten Blöcke der Serverinstanz zurück. Die Einträge sind mit Semikola getrennt.
- `reserve;<sourceInterface>;<targetBlock>;<loco>;<wagonCount>;end` stellt die Anfrage, eine Fahrstraße zu reservieren. `<sourceInterface>` ist die Netzwerkadresse (`Host:Port`) des anfragenden Stellwerks. `<targetBlock>` ist der Name des gewünschten Zielblocks. `<loco>` ist die Netzwerkadresse (`Host:Port`) der Train Unit und `<wagonCount>` die Wagenanzahl des zu übergebenden Zuges.

Train Control Protocol

Dieses Protokoll wird für die Übermittlung der Fahrerlaubnis an den Zug verwendet. Jede Train Unit hat einen Server, der vom Stellwerksprogramm angesprochen wird. Die Portnummer des Servers kann frei gewählt werden und muss dem Steuerungsprogramm beim Verbindungsaufbau mitgeteilt werden. Es stehen die folgenden Kommandos zur Verfügung:

- `name;end` gibt den Namen der verwendeten Lokomotive zurück.
- `ma;<drive>;end` sendet einen Fahrtbefehl an den Zug. `<drive>` ist dabei die gewünschte Fahrtrichtung an: 0 (Rückwärts), 1 (Halt) oder 2 (Vorwärts).

A.4 Quellcodeauszüge

Listing 1: Quellcode /PointDriver/GenericI2CDriver.cs

```
1  using Raspberry;
2  using Raspberry.IO.GeneralPurpose;
3  using Raspberry.IO.InterIntegratedCircuit;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace PointDriver
11 {
12     internal class GenericI2CDriver : IDisposable
13     {
14         private I2cDriver driver;
15         private I2cDeviceConnection connection0, connection1;
16         private byte state0, state1;
17
18         public GenericI2CDriver()
19         {
20             if (Board.Current.IsRaspberryPi)
21             {
22                 driver = new I2cDriver(ConnectorPin.P1Pin03.ToProcessor(), ConnectorPin.P1Pin05.ToProcessor());
23                 connection0 = driver.Connect(0x20);
24                 connection1 = driver.Connect(0x21);
25             }
26
27             state0 = state1 = 0xFF; // Alle High
28         }
29
30         public void Write(ushort relais, bool on)
31         {
32             if (!Board.Current.IsRaspberryPi)
33                 return;
34
35             var c = connection0;
36             var s = state0;
37             var r = relais;
38             if (relais > 8)
39             {
40                 c = connection1;
41                 s = state1;
42                 r -= 8;
43             }
44             if (relais > 16)
45                 throw new Exception("No relais with id " + relais);
46
47             byte mask = (byte)(0x01 << (r - 1));
48             if (on)
49                 s = (byte)(s & ~mask);
50             else
51                 s = (byte)(s | mask);
52
53             c.Write(s);
54
55             if (relais > 8)
56                 state1 = s;
57             else
58                 state0 = s;
59         }
60
61         public void Reset(bool resetState)
62         {
63             if (!Board.Current.IsRaspberryPi)
64                 return;
65
66             if (resetState)
67                 state0 = state1 = 0xFF;
68             connection0.Write(0xFF);
69             connection1.Write(0xFF);
70         }
71
72         public void Dispose()
73         {
74             driver.Dispose();
75         }
76     }
77 }
```

Listing 2: Quellcode /PointDriver/I2CPointDriver.cs

```
1  using Shared;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading;
7  using System.Threading.Tasks;
8
9  namespace PointDriver
10 {
11     public class I2CPointDriver : IPointDriver
12     {
13         private GenericI2CDriver driver;
14
15         public I2CPointDriver()
16         {
17             driver = new GenericI2CDriver();
18         }
19
```

```
20          public void Reset()
21          {
22              driver.Reset(true);
23          }
24
25          public async Task<bool> SetPointAsync(Point point, bool straight)
26          {
27              var r = point.Id;
28              var s = 16 - (int)Math.Floor((r - 1) / 2f);
29
30              driver.Write((ushort)r, straight);
31              await Task.Delay(400);
32              driver.Write((ushort)s, true);
33              await Task.Delay(600);
34              driver.Write((ushort)s, false);
35
36              await Task.Delay(600);
37              driver.Reset(false);
38
39              return true;
40          }
41      }
42  }
```

Listing 3: Quellcode /SensorDriver/GenericGpioDriver.cs

```
1   using Raspberry.IO.GeneralPurpose;
2   using Shared;
3   using System;
4   using System.Collections.Generic;
5   using System.Linq;
6   using System.Text;
7   using System.Threading;
8   using System.Threading.Tasks;
9
10  namespace SensorDriver
11  {
12      public class GenericGpioDriver
13      {
14          GpioConnectionDriver driver;
15
16          public event EventHandler<EventArgs<bool>> Changed;
17
18          public GenericGpioDriver()
19          {
20              if (!Raspberry.Board.Current.IsRaspberryPi)
21                  return;
22
23              driver = new GpioConnectionDriver();
24          }
25
26          public void Subscribe(int pinnumber, CancellationToken tok)
27          {
28              if (!Raspberry.Board.Current.IsRaspberryPi)
29                  return;
30
31              var pin = MapPin(pinnumber);
32
33              Task t = new Task(() =>
34              {
35                  bool last = true;
36
37                  while (!tok.IsCancellationRequested)
38                  {
39                      var x = driver.Read(pin.ToProcessor());
40                      if (last != x)
41                          Changed?.Invoke(pinnumber, new EventArgs<bool>(x));
42                      last = x;
43                      Thread.Sleep(100);
44                  }
45              }, tok);
46              t.Start();
47          }
48
49          private ConnectorPin MapPin(int pin)
50              => (ConnectorPin)Enum.Parse(typeof(ConnectorPin), "P1Pin" + pin);
51      }
52  }
```

Listing 4: Quellcode /SensorDriver/GpioSensorDriver.cs

```
1   using Shared;
2   using System;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.Text;
6   using System.Threading;
7   using System.Threading.Tasks;
8
9   namespace SensorDriver
10  {
11      public class GpioSensorDriver : ISensorDriver
12      {
13          private GenericGpioDriver driver;
14
15          public GpioSensorDriver()
16          {
17              driver = new GenericGpioDriver();
18              driver.Changed += (s, e) =>
19              {
20                  var id = (int)s;
21                  if (!e.Data)
22                      Enter?.Invoke(this, new EventArgs<int>(id));
```

```
23                  else
24                      Leave?.Invoke(this, new EventArgs<int>(id));
25              };
26          }
27
28          public event EventHandler<EventArgs<int>> Leave;
29          public event EventHandler<EventArgs<int>> Enter;
30
31          public void Subscribe(int id)
32          {
33              driver.Subscribe(id, new CancellationToken());
34          }
35      }
36  }
```

Listing 5: Quellcode /Shared/Drivers/ILocoDriver.cs

```
1  namespace Shared
2  {
3      public interface ILocoDriver
4      {
5          void ForceStop();
6          void MoveLoco(Loco loco, MovementDirection direction);
7      }
8  }
```

Listing 6: Quellcode /Shared/Drivers/IPointDriver.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      /// <summary>
10     /// Schnittstelle für Weichentreiber
11     /// </summary>
12     public interface IPointDriver
13     {
14         Task<bool> SetPointAsync(Point point, bool straight);
15
16         /// <summary>
17         /// Setzt den Treiber auf den Ausganszustand zurück.
18         /// </summary>
19         void Reset();
20     }
21 }
```

Listing 7: Quellcode /Shared/Drivers/ISensorDriver.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      public interface ISensorDriver
10     {
11         void Subscribe(int id);
12
13         event EventHandler<EventArgs<int>> Leave;
14         event EventHandler<EventArgs<int>> Enter;
15     }
16
17     public class EventArgs<T> : EventArgs
18     {
19         public T Data { get; private set; }
20
21         public EventArgs(T data)
22         {
23             Data = data;
24         }
25     }
26 }
```

Listing 8: Quellcode /Shared/Networking/BaseClient.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Net.Sockets;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace Shared.Networking
9  {
10     public abstract class BaseClient
11     {
12         private string server;
13         private int port;
14
```

```
15          public BaseClient(string server, int port)
16          {
17              this.server = server;
18              this.port = port;
19          }
20
21          protected string Send(string message)
22          {
23              var client = new TcpClient(server, port);
24
25              byte[] data = Encoding.ASCII.GetBytes(message);
26
27              var stream = client.GetStream();
28              stream.Write(data, 0, data.Length);
29
30              data = new byte[256];
31              string result = "";
32              int c = 0;
33              while ((c = stream.Read(data, 0, data.Length)) != 0)
34                  result += Encoding.ASCII.GetString(data, 0, c);
35
36              stream.Close();
37              client.Close();
38
39              return result;
40          }
41      }
42  }
```

Listing 9: Quellcode /Shared/Networking/BaseServer.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Net;
5   using System.Net.Sockets;
6   using System.Text;
7   using System.Threading.Tasks;
8
9   namespace Shared.Networking
10  {
11      // Basiert auf https://docs.microsoft.com/en-us/dotnet/framework/network-programming/asynchronous-server-socket-
            example
12      public abstract class BaseServer
13      {
14          private Socket listener;
15          private int port;
16
17          private const int BUFFER_SIZE = 1024;
18          private string result;
19          private byte[] buffer;
20
21          public BaseServer(int port)
22          {
23              buffer = new byte[BUFFER_SIZE];
24              this.port = port;
25          }
26
27          public event EventHandler Listening;
28          public event EventHandler<EventArgs<string>> ConnectionError;
29
30          public void StartListening()
31          {
32              IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
33              IPEndPoint localEndPoint = new IPEndPoint(ipAddress, port);
34
35              listener = new Socket(ipAddress.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
36
37              try
38              {
39                  listener.Bind(localEndPoint);
40                  listener.Listen(100);
41                  listener.BeginAccept(new AsyncCallback(AcceptCallback), listener);
42                  Listening?.Invoke(this, null);
43              }
44              catch (Exception e)
45              {
46                  ConnectionError?.Invoke(this, new EventArgs<string>(e.Message));
47              }
48          }
49
50          public void AcceptCallback(IAsyncResult ar)
51          {
52              Socket listener = (Socket)ar.AsyncState;
53              Socket client = listener.EndAccept(ar);
54
55              client.BeginReceive(buffer, 0, buffer.Length, 0,
56                  new AsyncCallback(ReadCallback), client);
57          }
58
59          public void ReadCallback(IAsyncResult ar)
60          {
61              Socket client = (Socket)ar.AsyncState;
62
63              int bytesRead = client.EndReceive(ar);
64
65              if (bytesRead > 0)
66              {
67                  // There  might be more data, so store the data received so far.
68                  result += Encoding.ASCII.GetString(buffer, 0, bytesRead);
69
70                  if (result.IndexOf(";end") > -1)
71                  {
72                      var res = ProcessRequest(result);
73                      result = "";
74                      Send(client, res);
75                  }
```

```
76                  else
77                  {
78                      // Not all data received. Get more.
79                      client.BeginReceive(buffer, 0, buffer.Length, 0,
80                      new AsyncCallback(ReadCallback), client);
81                  }
82              }
83          }
84
85          protected abstract string ProcessRequest(string result);
86
87          private void Send(Socket socket, string data)
88          {
89              byte[] byteData = Encoding.ASCII.GetBytes(data);
90
91              socket.BeginSend(byteData, 0, byteData.Length, 0, new AsyncCallback(SendCallback), socket);
92          }
93
94          private void SendCallback(IAsyncResult ar)
95          {
96              try
97              {
98                  Socket handler = (Socket)ar.AsyncState;
99                  int bytesSent = handler.EndSend(ar);
100
101                 handler.Shutdown(SocketShutdown.Both);
102                 handler.Close();
103
104                 // Wieder auf neuen Client warten
105                 listener.BeginAccept(new AsyncCallback(AcceptCallback), listener);
106             }
107             catch (Exception e)
108             {
109                 Console.WriteLine(e.ToString());
110             }
111         }
112     }
113 }
```

Listing 10: Quellcode /Shared/Networking/InterlockingClient.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared.Networking
8  {
9      public class InterlockingClient : BaseClient
10     {
11         public InterlockingClient(string server, int port) : base(server, port)
12         {
13         }
14
15         public int Reserve(string sourceInterface, string target_block, Train train)
16         {
17             var res = Send($"reserve;{sourceInterface};{target_block};{train.Locomotive.HostPort};{train.WagonCount};end"
                   );
18             if (int.TryParse(res, out int result))
19                 return result;
20             return -1;
21         }
22
23         public string[] ListAvailable()
24         {
25             var res = Send("list;end");
26             return res.Split(';');
27         }
28     }
29 }
```

Listing 11: Quellcode /Shared/Networking/InterlockingServer.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5
6  namespace Shared.Networking
7  {
8      public class InterlockingServer : BaseServer
9      {
10         private IState state;
11         private HashSet<int> occupiedHandles;
12         private int lastHandle;
13         private LocoDispatcher locoDispatcher;
14         private MovementLogic driver;
15
16         public InterlockingServer(int port, IState state, LocoDispatcher locoDispatcher, MovementLogic driver) : base(
               port)
17         {
18             this.state = state;
19             occupiedHandles = new HashSet<int>();
20             this.locoDispatcher = locoDispatcher;
21             this.driver = driver;
22         }
23
24         protected override string ProcessRequest(string result)
25         {
26             var blocks = state.Blocks.Where(b => !b.Reserved && b.Occupied == null && b.Type == BlockType.Station);
27             var bnames = blocks.Select(b => b.Name);
28
```

```
29              var cmd = result.Split(';');
30              if (cmd.Last() != "end" || cmd.Length < 2)
31                  return "-1";
32              switch(cmd[0])
33              {
34                  case "list":
35                      return string.Join(";", bnames);
36                  case "reserve":
37                      var sourceInterface = cmd[1];
38                      var targetBlockName = cmd[2];
39                      var locoAddress = cmd[3];
40                      var wagonCount = int.Parse(cmd[4]);
41
42                      if (!bnames.Contains(targetBlockName))
43                          return "-1";
44
45                      var loco = locoDispatcher.AssignLoco(locoAddress);
46                      var train = new Train(loco) { WagonCount = wagonCount };
47
48                      var sourceBlock = state.Blocks.First(b => b.Interface == sourceInterface);
49                      var targetBlock = state.Blocks.First(b => b.Name == targetBlockName);
50
51                      var tcs = new TaskCompletionSource<bool>();
52                      driver.OvertakeTrain(sourceBlock, targetBlock, train, tcs);
53                      if (!tcs.Task.Wait(3000))
54                          return "-1";
55
56                      var idx = ++lastHandle;
57                      occupiedHandles.Add(idx);
58                      return idx.ToString();
59              }
60              return "-1";
61          }
62      }
63  }
```

Listing 12: Quellcode /Shared/Networking/LocoClient.cs

```
1  using System;
2
3  namespace Shared.Networking
4  {
5      internal class LocoClient : BaseClient
6      {
7          public LocoClient(string server, int port) : base(server, port)
8          {
9          }
10
11         public void MoveLoco(MovementDirection direction)
12             => Send($"ma;{(int)direction};end");
13
14         public string GetName()
15             => Send($"name;end");
16     }
17 }
```

Listing 13: Quellcode /Shared/Networking/LocoDispatcher.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared.Networking
8  {
9      public class LocoDispatcher : ILocoDriver
10     {
11         Dictionary<Loco, LocoClient> knownLocos;
12
13         public LocoDispatcher()
14         {
15             knownLocos = new Dictionary<Loco, LocoClient>();
16         }
17
18         public Loco AssignLoco(string hostPort)
19         {
20             var parts = hostPort.Split(':');
21             var host = parts[0];
22             var port = int.Parse(parts[1]);
23             var client = new LocoClient(host, port);
24
25             var name = client.GetName();
26             var loco = new Loco(name, hostPort);
27
28             knownLocos.Add(loco, client);
29             return loco;
30         }
31
32         public void MoveLoco(Loco loco, MovementDirection direction)
33         {
34             var l = knownLocos[loco];
35             l.MoveLoco(direction);
36         }
37
38         public void ForceStop()
39         {
40             foreach (var loco in knownLocos.Values)
41                 loco.MoveLoco(MovementDirection.Stop);
42         }
43     }
44 }
```

Listing 14: Quellcode /Shared/Path/PathDriver.cs

```
using Shared;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Shared.Path
{
    public class PathDriver
    {
        private IPointDriver driver;

        public PathDriver(IPointDriver driver)
        {
            this.driver = driver;
        }

        public void Reset()
        {
            driver.Reset();
        }

        public async Task DoPath(PathResult path)
        {
            var ac = path.GetActions();
            Console.WriteLine(string.Join(Environment.NewLine, ac.Select(a => a.ToString())));

            await InternalDoPath(ac);
        }

        private async Task InternalDoPath(IEnumerable<PointAction> actions)
        {
            foreach (var action in actions)
            {
                await driver.SetPointAsync(action.Point, action.Straight);
                await Task.Delay(400);
            }
        }
    }
}
```

Listing 15: Quellcode /Shared/Path/PathFinder.cs

```
using Shared;
using Shared.Util;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Shared.Path
{
    public class Pathfinder
    {
        private IState layout;

        private const int BLOCK_PRIO = 0, POINT_PRIO = 10;

        public Pathfinder(IState layout)
        {
            this.layout = layout;
        }

        public PathResult FindPath(Block source, Block destination)
        {
            // Zuerst nach einem Weg ohne Richtungswechel suchen, dann mit
            var way = GetFromAToB(source, destination, false);
            if (way == null)
                way = GetFromAToB(source, destination, true);

            // Split-up of multi-movement paths
            bool split = false;
            var visited = new HashSet<IRail>();
            for (int i = 0; i < way.Count; i++)
            {
                if (!visited.Add(way[i]))
                {
                    way = way.Take(i).ToList();
                    split = true;
                    break;
                }
            }

            Console.WriteLine($"== Path from {source.Name} to {way.LastOrDefault().Name}");
            if (way == null)
            {
                Console.WriteLine("No path found!");
                return new PathResult(MovementDirection.Stop, null, false);
            }

            if (way[0] is Block b && b.End == way[1])
                return new PathResult(MovementDirection.Forward, way.ToArray(), split);
            return new PathResult(MovementDirection.Backward, way.ToArray(), split);
        }

        private List<IRail> GetFromAToB(Block start, Block dest, bool changesAllowed)
        {
            if (!layout.Blocks.Contains(start) || !layout.Blocks.Contains(dest))
                throw new ArgumentException("Start- oder Zielblock nicht im Layout enthalten!");

            var visited = new CountSet<IRail>(changesAllowed ? 2 : 1);
            var queue = new PrioQueue<IRail>();
```

```
61              var parents = new Dictionary<IRail, IRail>();
62              var ways = new Dictionary<IRail, IRail[]>();
63
64              queue.Enqueue(start, BLOCK_PRIO);
65              parents[start] = null;
66              ways[start] = new IRail[0];
67
68              while (queue.Any())
69              {
70                  var rail = queue.Dequeue();
71
72                  if (rail == dest)
73                      return ways[dest].SkipWhile(r => r == null).Concat(new[] { dest }).ToList();
74
75                  var last = parents[rail];
76                  foreach (var n in rail.GetNext(last, changesAllowed).Where(r => r != null))
77                  {
78                      if (n.Reserved || (n is Block blk && blk.Occupied != null)
79                          || visited.ContainsMax(n) || queue.Contains(n))
80                          continue;
81
82                      parents[n] = rail;
83                      ways[n] = ways[last ?? start].Concat(new[] { last, rail }).ToArray();
84                      queue.Enqueue(n, n is Point ? BLOCK_PRIO : POINT_PRIO);
85                  }
86
87                  visited.Add(rail);
88              }
89
90              return null;
91          }
92      }
93  }
```

Listing 16: Quellcode /Shared/Path/PathResult.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace Shared.Path
8   {
9       public class PathResult
10      {
11          public MovementDirection Direction { get; private set; }
12
13          public IRail[] Path { get; private set; }
14
15          public bool Continue { get; private set; }
16
17          public PathResult(MovementDirection direction, IRail[] path, bool needContinue)
18          {
19              Direction = direction;
20              Path = path;
21              Continue = needContinue;
22          }
23
24          public IEnumerable<int> GetSensors(Dictionary<Block, SensorIds> sensorMapping)
25          {
26              IRail last = null;
27              foreach (var r in Path)
28              {
29                  if (r is Block block && sensorMapping.ContainsKey(block))
30                  {
31                      var nidx = Array.IndexOf(Path, block) + 1;
32                      var next = nidx < Path.Length ? Path[nidx] : null;
33
34                      if (last == block.Begin || next == block.End)
35                      {
36                          yield return sensorMapping[block].Begin;
37                          yield return sensorMapping[block].End;
38                      }
39                      else if (last == block.End || next == block.Begin)
40                      {
41                          yield return sensorMapping[block].End;
42                          yield return sensorMapping[block].Begin;
43                      }
44                      else
45                          throw new Exception("Programmfehler bei der Verarbeitung von Block " + block.Name);
46                  }
47                  last = r;
48              }
49          }
50
51          public IEnumerable<PointAction> GetActions()
52          {
53              IRail last = null;
54              foreach (var r in Path)
55              {
56                  if (r is Point point)
57                  {
58                      var next = Path[Array.IndexOf(Path, point) + 1];
59                      if (last == point.OutDiverging || next == point.OutDiverging)
60                          yield return new PointAction(point, false);
61                      else if (last == point.OutStraight || next == point.OutStraight)
62                          yield return new PointAction(point, true);
63                      else
64                          throw new Exception("Programmfehler bei der Verarbeitung von Weiche " + point.Id);
65                  }
66                  last = r;
67              }
68          }
69      }
70  }
```

Listing 17: Quellcode /Shared/Util/CountSet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Shared.Util
{
    internal class CountSet<T>
    {
        private Dictionary<T, int> dict;
        private int maxCount;

        public CountSet(int maxCount)
        {
            dict = new Dictionary<T, int>();
            this.maxCount = maxCount;
        }

        public bool Add(T obj)
        {
            dict.TryGetValue(obj, out int val);
            dict[obj] = ++val;
            return val <= maxCount;
        }

        public bool ContainsMax(T obj)
        {
            dict.TryGetValue(obj, out int val);
            return val >= maxCount;
        }
    }
}
```

Listing 18: Quellcode /Shared/Util/PrioQueue.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Shared.Util
{
    internal class PrioQueue<T> where T : class
    {
        private SortedList<int, Queue<T>> list = new SortedList<int, Queue<T>>();

        public bool Any() => list.Any();

        public void Enqueue(T obj, int rank)
        {
            list.TryGetValue(rank, out Queue<T> q);
            q = q ?? new Queue<T>();
            q.Enqueue(obj);
            list[rank] = q;
        }

        public bool Contains(T obj)
            => list.Any(e => e.Value.Contains(obj));

        public T Dequeue()
        {
            if (!list.Any())
                return null;

            var val = list.ElementAt(0);
            if (val.Value.Count == 1)
                list.RemoveAt(0);
            return val.Value.Dequeue();
        }
    }
}
```

Listing 19: Quellcode /Shared/Block.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Shared
{
    [System.Diagnostics.DebuggerDisplay("Block: {Name}")]
    public class Block : IRail
    {
        public BlockType Type { get; private set; }

        public string Name { get; private set; }

        public Train Occupied { get; set; }

        public IRail Begin { get; set; }

        public IRail End { get; set; }

        public string Interface { get; set; }

        public bool Reserved { get; set; }
```

```
25
26          public IEnumerable<IRail> GetNext(IRail last, bool changesAllowed)
27              => new[] { Begin, End }.Where(r => changesAllowed || r != last);
28
29          public Block(string name, BlockType type)
30          {
31              Name = name;
32              Type = type;
33          }
34      }
35  }
```

Listing 20: Quellcode /Shared/BlockType.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      public enum BlockType
10     {
11         Station,
12         Line,
13         Handover,
14     }
15 }
```

Listing 21: Quellcode /Shared/IRail.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      public interface IRail
10     {
11         string Name { get; }
12
13         bool Reserved { get; set; }
14
15         IEnumerable<IRail> GetNext(IRail last, bool changesAllowed);
16     }
17 }
```

Listing 22: Quellcode /Shared/IState.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      public interface IState
10     {
11         Dictionary<Block, SensorIds> SensorMapping { get; }
12
13         string StwName { get; }
14
15         int StwId { get; }
16
17         List<Block> Blocks { get; }
18
19         List<Point> Points { get; }
20
21         string HostPort { get; }
22     }
23
24     public struct SensorIds
25     {
26         public int Begin { get; set; }
27
28         public int End { get; set; }
29     }
30 }
```

Listing 23: Quellcode /Shared/Loco.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      public class Loco
10     {
```

```
11          public string Name { get; set; }
12
13          public bool TurnDirection { get; set; }
14
15          public string HostPort { get; set; }
16
17          public Loco(string name, string hostPort)
18          {
19              Name = name;
20              HostPort = hostPort;
21          }
22      }
23  }
```

Listing 24: Quellcode /Shared/LogicException.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      public class LogicException : Exception
10     {
11         public LogicException()
12         {
13         }
14
15         public LogicException(string message) : base(message)
16         {
17         }
18     }
19 }
```

Listing 25: Quellcode /Shared/MovementDirection.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      public enum MovementDirection
10     {
11         Backward,
12         Stop,
13         Forward
14     }
15 }
```

Listing 26: Quellcode /Shared/MovementLogic.cs

```
1  using Shared;
2  using Shared.Networking;
3  using Shared.Path;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Text;
8  using System.Threading;
9  using System.Threading.Tasks;
10
11 namespace Shared
12 {
13     public class MovementLogic
14     {
15         private IState state;
16
17         private PathDriver pathDriver;
18         private Pathfinder pathFinder;
19         private ISensorDriver sensorDriver;
20         private ILocoDriver locoDriver;
21
22         private Dictionary<int, SensorCounter> waiters;
23
24         private Dictionary<Block, InterlockingClient> clientCache;
25
26         public event EventHandler MovementFinished, PathSetFailed, StateChanged;
27
28         public List<IEnumerable<IRail>> reservations;
29
30         public MovementLogic(IState state, ISensorDriver sensorDriver, IPointDriver pointDriver, ILocoDriver locoDriver)
31         {
32             this.state = state;
33             pathDriver = new PathDriver(pointDriver);
34             pathFinder = new Pathfinder(state);
35             this.sensorDriver = sensorDriver;
36             this.locoDriver = locoDriver;
37
38             reservations = new List<IEnumerable<IRail>>();
39
40             waiters = new Dictionary<int, SensorCounter>();
41             clientCache = new Dictionary<Block, InterlockingClient>();
42
43             var sensors = new HashSet<int>();
```

```
44              foreach (var s in state.SensorMapping)
45              {
46                  if (sensors.Add(s.Value.Begin))
47                      InitCounter(s.Value.Begin, s.Key);
48                  if (sensors.Add(s.Value.End))
49                      InitCounter(s.Value.End, s.Key);
50              }
51
52              this.sensorDriver.Enter += (s, e) => DoNotify(e, true);
53              this.sensorDriver.Leave += (s, e) => DoNotify(e, false);
54
55              FailAll(); // Alle Lichtschranken scharf stellen (Keine Zugbewegung erwartet)
56          }
57
58          public async void DoTrain(Block start, Block end, string remoteTarget = null)
59          {
60              if (remoteTarget != null)
61              {
62                  if (!DoRemoteInterlocking(end, start.Occupied, remoteTarget))
63                      throw new LogicException("Fern-Fahrstraße konnte nicht eingestellt werden.");
64              }
65
66              void finished(bool b) { if (b == false) PathSetFailed?.Invoke(this, null); }
67              await InternalDoTrain(start, end, finished);
68          }
69
70          public async void OvertakeTrain(Block start, Block end, Train train, TaskCompletionSource<bool> tcs)
71              => await InternalDoTrain(start, end, (b) => tcs.TrySetResult(b), train);
72
73          private async Task InternalDoTrain(Block start, Block end, Action<bool> finished, Train train = null)
74          {
75              PathResult result;
76              var begin = start;
77              do
78              {
79                  result = pathFinder.FindPath(begin, end);
80                  finished?.Invoke(result.Path != null);
81                  if (result.Path == null)
82                      return;
83
84                  ReservePath(result, begin.Occupied);
85                  await pathDriver.DoPath(result);
86
87                  var mode = PathMode.Internal;
88                  if (result.Path.First() is Block bf && bf.Type == BlockType.Handover)
89                      mode = PathMode.Overtake; // Nur bei der wirklich ersten Aktion den Overtake-Modus setzen
90                  else if (result.Path.Last() is Block bl && bl.Type == BlockType.Handover)
91                      mode = PathMode.Handover; // Nur bei der wirklich letzten Aktion den Handover-Modus setzen
92
93                  await InternalMovement(begin, result, mode, train);
94                  begin = (Block)result.Path.Last();
95              } while (result.Continue);
96
97              MovementFinished?.Invoke(this, null);
98          }
99
100         // Führt eine einzelne Zugbewegung durch
101         private async Task InternalMovement(Block start, PathResult result, PathMode mode, Train train)
102         {
103             var direction = result.Direction;
104
105             var tra = train ?? start.Occupied;
106             var loco = tra.Locomotive;
107             direction = loco.TurnDirection ?
108                 (direction == MovementDirection.Forward ? MovementDirection.Backward : MovementDirection.Forward) :
                        direction;
109             if (mode == PathMode.Internal || mode == PathMode.Handover)
110                 locoDriver.MoveLoco(loco, direction);
111
112             FailAll(); // Alle Lichtschranken "scharf"
113
114             // Lichtschranken auf der Strecke konfigureieren
115             var skipCount = mode == PathMode.Overtake ? 0 : 1; // Bei Overtake: Erste LS gehört zur Strecke
116             var rawpebs = result.GetSensors(state.SensorMapping).Skip(skipCount);
117
118             var takeCount = mode == PathMode.Handover ? 0 : 1; // Bei Handover: Die letzte LS gehört zur Strecke
119             var pebs = rawpebs.Take(rawpebs.Count() - takeCount).Distinct().ToArray();
120
121             // Erwartete Sensor-Toggle-Anzahl
122             var wcount = 2 * tra.WagonCount;
123
124             var waiters = pebs.Select(p => GetCounter(p));
125             var route = waiters.Take(pebs.Length - 1);
126             AcceptRoute(route, wcount);
127
128             var ew = GetCounter(pebs.Last());
129             ew.ClearWaiter();
130
131             await ew.WaitToggles(wcount);
132
133             if (mode != PathMode.Handover)
134                 locoDriver.MoveLoco(loco, MovementDirection.Stop);
135
136             CheckRoute(route, wcount); // Überprüfen, ob wir irgendwo Wagen verloren haben
137
138             await Task.Delay(1000); // Letzte Zugbewegung abwarten
139
140             var end = (Block)result.Path.Last();
141
142             if (mode != PathMode.Handover)
143                 end.Occupied = tra;
144
145             start.Occupied = null;
146             ClearReservedPath(result.Path);
147             FailAll(); // Alle Lichtschranken "scharf"
148
149             StateChanged?.Invoke(this, null);
150         }
151
152         #region Reserving
```

```
153         internal void ReservePath(PathResult result, Train train)
154         {
155             foreach (var rail in result.Path.Distinct())
156             {
157                 if (rail.Reserved || (rail is Block b && b.Occupied != null && b.Occupied != train))
158                     throw new LogicException("Fahrwegelement kann nicht reserviert werden");
159                 rail.Reserved = true;
160             }
161             reservations.Add(result.Path);
162             StateChanged?.Invoke(this, null);
163         }
164 
165         internal void ClearReservedPath(IEnumerable<IRail> path)
166         {
167             foreach (var rail in path)
168                 rail.Reserved = false;
169             reservations.RemoveAll(l => l.SequenceEqual(path));
170             StateChanged?.Invoke(this, null);
171         }
172         #endregion
173 
174         #region Sensors
175         private void DoNotify(EventArgs<int> e, bool state)
176         {
177             var id = e.Data;
178 
179             if (waiters.ContainsKey(id))
180             {
181                 try
182                 {
183                     waiters[id].NotifyEdge(state);
184                 }
185                 catch (TrafficException)
186                 {
187                     Console.WriteLine("TrafficException thrown: Unexpected signal edge on id " + id);
188                     locoDriver.ForceStop();
189                     throw;
190                 }
191             }
192         }
193 
194         private SensorCounter GetCounter(int id) => waiters[id];
195 
196         private void InitCounter(int id, Block b)
197         {
198             var tsc = new SensorCounter(id, b);
199             waiters[id] = tsc;
200             sensorDriver.Subscribe(id);
201         }
202 
203         private void FailAll()
204         {
205             foreach (var w in waiters.Values)
206             {
207                 if (w.Block.Reserved)
208                     continue;
209                 w.ClearWaiter();
210                 w.FailOnEdge();
211             }
212         }
213 
214         private void AcceptRoute(IEnumerable<SensorCounter> waiters, int wagonCount)
215         {
216             foreach (var w in waiters)
217             {
218                 w.ClearWaiter();
219                 w.CountToggles();
220             }
221         }
222 
223         private void CheckRoute(IEnumerable<SensorCounter> waiters, int wagonCount)
224         {
225             foreach (var w in waiters)
226             {
227                 if (w.Counter != wagonCount)
228                 {
229                     Console.WriteLine("Unerwartete Abweichung von der erwarteten Wagenanzahl");
230                     throw new TrafficException("Unerwartete Abweichung von der erwarteten Wagenanzahl");
231                 }
232             }
233         }
234         #endregion
235 
236         #region RemoteInterlocking
237         public InterlockingClient GetClient(Block handover)
238         {
239             if (handover.Type != BlockType.Handover)
240                 return null;
241             if (!clientCache.TryGetValue(handover, out InterlockingClient ic))
242             {
243                 var parts = handover.Interface.Split(':');
244                 ic = new InterlockingClient(parts[0], int.Parse(parts[1]));
245                 clientCache[handover] = ic;
246             }
247             return ic;
248         }
249 
250         public bool DoRemoteInterlocking(Block handover, Train train, string targetBlock)
251         {
252             var cl = GetClient(handover);
253             var handle = cl.Reserve(state.HostPort, targetBlock, train);
254             if (handle == -1)
255                 return false;
256             return true;
257         }
258         #endregion
259 
260         private enum PathMode
261         {
262             Internal,
```

```
263             Overtake,
264             Handover
265         }
266     }
267 }
```

Listing 27: Quellcode /Shared/Point.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      [System.Diagnostics.DebuggerDisplay("Point: {Name}")]
10     public class Point : IRail
11     {
12         public IRail Entry { get; set; }
13
14         public IRail OutStraight { get; set; } // gerade
15
16         public IRail OutDiverging { get; set; } // abzweigend
17
18         public string Name { get; private set; }
19
20         public int Id { get; private set; }
21
22         public bool Reserved { get; set; }
23
24         public IEnumerable<IRail> GetNext(IRail last, bool changesAllowed)
25         {
26             if (last == Entry)
27                 return new[] { OutStraight, OutDiverging };
28             else
29                 return new[] { Entry };
30         }
31
32         public Point(string name, int id)
33         {
34             Name = name;
35             Id = id;
36         }
37     }
38 }
```

Listing 28: Quellcode /Shared/PointAction.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      [System.Diagnostics.DebuggerDisplay("{Point.Name} -> {Straight}")]
10     public class PointAction
11     {
12         public Point Point { get; private set; }
13
14         public bool Straight { get; private set; }
15
16         public PointAction(Point point, bool straight)
17         {
18             Point = point;
19             Straight = straight;
20         }
21
22         public override string ToString()
23         {
24             return $"{Point.Name} -> {Straight}";
25         }
26     }
27 }
```

Listing 29: Quellcode /Shared/SensorCounter.cs

```
1  using Shared;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace Shared
9  {
10     public class SensorCounter
11     {
12         private TaskCompletionSource<bool> tcs;
13         private bool countingToggles;
14         private bool countingEdgeState;
15         private int targetCount;
16         private int id;
17
18         private bool failOnEdge;
19
20         public int Counter { get; private set; }
```

```
21
22          public bool CountingInitialized { get; private set; }
23
24          public Block Block { get; private set; }
25
26          public SensorCounter(int id, Block b)
27          {
28              this.id = id;
29              tcs = new TaskCompletionSource<bool>();
30              Block = b;
31          }
32
33          public async Task WaitToggles(int toggleCount)
34          {
35              CountToggles();
36              targetCount = toggleCount;
37
38              await tcs.Task;
39          }
40
41          public async Task WaitEdges(bool state, int count)
42          {
43              CountEdges(state);
44              targetCount = count;
45
46              await tcs.Task;
47          }
48
49          public void CountToggles()
50          {
51              if (CountingInitialized)
52                  throw new InvalidOperationException("Already listening and counting");
53
54              CountingInitialized = true;
55              countingToggles = true;
56              targetCount = -1;
57          }
58
59          public void CountEdges(bool state)
60          {
61              if (CountingInitialized)
62                  throw new InvalidOperationException("Already listening and counting");
63
64              CountingInitialized = true;
65              countingToggles = false;
66              countingEdgeState = state;
67              targetCount = -1;
68          }
69
70          public void FailOnEdge()
71          {
72              if (CountingInitialized)
73                  throw new InvalidOperationException("Already listening and counting");
74
75              CountingInitialized = true;
76              failOnEdge = true;
77          }
78
79          public void NotifyEdge(bool state)
80          {
81              if (!CountingInitialized)
82                  return;
83
84              if (failOnEdge)
85                  throw new TrafficException("Unerwartete Signalflanke an Lichtschranke");
86
87              if (countingToggles || countingEdgeState == state)
88                  Counter++;
89
90              if (targetCount > 0 && Counter >= targetCount)
91              {
92                  CountingInitialized = false;
93                  tcs.SetResult(true);
94                  Counter = 0;
95              }
96          }
97
98          public void ClearWaiter()
99          {
100             CountingInitialized = false;
101             countingToggles = false;
102             targetCount = -1;
103             failOnEdge = false;
104             Counter = 0;
105         }
106     }
107 }
```

Listing 30: Quellcode /Shared/TrafficException.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      public class TrafficException : Exception
10     {
11         public TrafficException()
12         {
13         }
14
15         public TrafficException(string message) : base(message)
16         {
17         }
```

```
18      }
19  }
```

Listing 31: Quellcode /Shared/Train.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Shared
8  {
9      public class Train
10     {
11         public int WagonCount { get; set; }
12
13         public Loco Locomotive { get; set; }
14
15         public Train(Loco loco)
16         {
17             Locomotive = loco;
18         }
19     }
20 }
```

Listing 32: Quellcode /stwgui/AssignForm.cs

```
1  using Shared;
2  using Shared.Networking;
3  using System;
4  using System.Collections.Generic;
5  using System.Windows.Forms;
6
7  namespace stwgui
8  {
9      public partial class AssignForm : Form
10     {
11         public int WagonCount { get; set; }
12
13         public Loco Loco { get; set; }
14
15         private LocoDispatcher dispatcher;
16
17         public AssignForm(LocoDispatcher dispatcher) : this()
18         {
19             this.dispatcher = dispatcher;
20         }
21
22         private AssignForm()
23         {
24             InitializeComponent();
25         }
26
27         private void okButton_Click(object sender, EventArgs e)
28         {
29             WagonCount = int.Parse(cTextBox.Text);
30
31             try
32             {
33                 Loco = dispatcher.AssignLoco(aTextBox.Text);
34                 Loco.TurnDirection = tCheckBox.Checked;
35             }
36             catch
37             {
38                 MessageBox.Show("Adresse ungültig");
39                 return;
40             }
41
42             DialogResult = DialogResult.OK;
43             Close();
44         }
45     }
46 }
```

Listing 33: Quellcode /stwgui/MainForm.cs

```
1  using PointDriver;
2  using SensorDriver;
3  using Shared;
4  using Shared.Networking;
5  using System;
6  using System.Collections.Generic;
7  using System.IO;
8  using System.Windows.Forms;
9
10 namespace stwgui
11 {
12     public partial class MainForm : Form
13     {
14         private IState state;
15         private StateRenderer renderer;
16         private MovementLogic driver;
17         private LocoDispatcher locoDispatcher;
18
19         public MainForm()
20         {
21             InitializeComponent();
22
```

```
23              var args = Environment.GetCommandLineArgs();
24              if (args.Length < 2)
25              {
26                  MessageBox.Show("Bitte den Daten-Basispfad als Parameter angeben!");
27                  Load += (s, e) => Close();
28                  return;
29              }
30              var lpath = Path.Combine(args[1], "layout.json");
31              var spath = Path.Combine(args[1], "sensors.json");
32              if (!File.Exists(lpath) || !File.Exists(spath))
33              {
34                  MessageBox.Show("Eine oder mehrere der benötigten Datendateien wurden nicht gefunden!");
35                  Load += (s, e) => Close();
36                  return;
37              }
38              state = new StateLoader(lpath, spath);
39
40              Text += $" {state.StwName} (Stw {state.StwId})";
41
42              renderer = new StateRenderer(state);
43              renderer.Dock = DockStyle.Fill;
44              hostPanel.Controls.Add(renderer);
45
46              renderer.Assigned += (s, e) =>
47              {
48                  stateLabel.Text = "Bereit";
49                  renderer.Refresh();
50              };
51
52              locoDispatcher = new LocoDispatcher();
53              driver = new MovementLogic(state, new GpioSensorDriver(), new I2CPointDriver(), locoDispatcher);
54
55              var server = new InterlockingServer(35000 + state.StwId, state, locoDispatcher, driver);
56              server.StartListening();
57
58              driver.MovementFinished += (s, e) =>
59              {
60                  BeginInvoke((Action)(() => {
61                      renderer.ResetSelection();
62                  }));
63              };
64              driver.StateChanged += (s, e) =>
65              {
66                  BeginInvoke((Action)(() => {
67                      renderer.ResetSelection();
68                      stateLabel.Text = "Bereit";
69                  }));
70              };
71              driver.PathSetFailed += (s, e) =>
72              {
73                  BeginInvoke((Action)(() => {
74                      MessageBox.Show("Fehler beim Festelegen der Fahrstraße. Fahrweg ist bereits belegt oder reserviert.")
                                ;
75                      renderer.ResetSelection();
76                  }));
77              };
78          }
79
80          private void moveButton_Click(object sender, EventArgs e)
81          {
82              if (renderer.PathFrom == null || renderer.PathTo == null)
83              {
84                  MessageBox.Show("Bitte erst Start- und Zielblock auswählen!");
85                  return;
86              }
87              string remoteTarget = null;
88              if (renderer.PathTo.Type == BlockType.Handover)
89              {
90                  var cl = driver.GetClient(renderer.PathTo);
91                  var ts = new TargetSelector(cl);
92                  ts.ShowDialog();
93                  remoteTarget = ts.Target;
94              }
95
96              stateLabel.Text = "Warte auf Fahrstraßenfestlegung";
97              driver.DoTrain(renderer.PathFrom, renderer.PathTo, remoteTarget);
98          }
99
100         private void resetButton_Click(object sender, EventArgs e)
101         {
102             renderer.ResetSelection();
103         }
104
105         private void assignButton_Click(object sender, EventArgs e)
106         {
107             var pf = new AssignForm(locoDispatcher);
108             if (pf.ShowDialog() != DialogResult.OK)
109                 return;
110
111             renderer.Assign = new Train(pf.Loco) { WagonCount = pf.WagonCount + 1 };
112             renderer.Refresh();
113             stateLabel.Text = "Zuweisen: " + pf.Loco.Name;
114         }
115     }
116 }
```

Listing 34: Quellcode /stwgui/Program.cs

```
1  using System;
2  using System.Windows.Forms;
3
4  namespace stwgui
5  {
6      static class Program
7      {
8          /// <summary>
9          /// Der Haupteinstiegspunkt für die Anwendung.
```

```
10          /// </summary>
11          [STAThread]
12          static void Main()
13          {
14              Application.EnableVisualStyles();
15              Application.SetCompatibleTextRenderingDefault(false);
16              Application.Run(new MainForm());
17          }
18      }
19  }
```

Listing 35: Quellcode /stwgui/StateLoader.cs

```
1   using Newtonsoft.Json;
2   using Newtonsoft.Json.Linq;
3   using Shared;
4   using System.Collections.Generic;
5   using System.IO;
6   using System.Linq;
7   using System.Reflection;
8
9   namespace stwgui
10  {
11      public class StateLoader : IState
12      {
13          public string StwName { get; private set; }
14
15          public int StwId { get; private set; }
16
17          public List<Block> Blocks { get; private set; }
18
19          public List<Point> Points { get; private set; }
20
21          public Dictionary<Block, SensorIds> SensorMapping { get; private set; }
22
23          public string HostPort => "localhost:" + (35000 + StwId).ToString();
24
25          public StateLoader(string layoutJson, string sensorsJson)
26          {
27              // Deserialize Layout
28              var jLayout = (JObject)JsonConvert.DeserializeObject(File.ReadAllText(layoutJson));
29              var jbs = (JArray)jLayout.GetValue("Blocks");
30              var blocks = jbs.OfType<JObject>().Select(t => t.ToObject<Block>()).ToList();
31              var jps = (JArray)jLayout.GetValue("Points");
32              var points = jps.OfType<JObject>().Select(t => t.ToObject<Point>()).ToList();
33
34              var map = blocks.OfType<IRail>().Concat(points).ToDictionary(r => r.Name);
35
36              var jcs = (JArray)jLayout.GetValue("Connections");
37              foreach (var jc in jcs.OfType<JObject>())
38              {
39                  var jsrc = jc.GetValue("src").ToString();
40                  var jsrcDir = jc.GetValue("src_dir").ToString();
41                  var jdest = jc.GetValue("dest").ToString();
42                  var jdestDir = jc.GetValue("dest_dir").ToString();
43
44                  var src = map[jsrc];
45                  var dest = map[jdest];
46                  SetRail(src, jsrcDir, dest);
47                  SetRail(dest, jdestDir, src);
48              }
49
50              StwName = (string)jLayout.GetValue("Stw_Name");
51              StwId = (int)jLayout.GetValue("Stw_Id");
52              Points = points;
53              Blocks = blocks;
54
55              // Deserialize SensorMapping
56              SensorMapping = new Dictionary<Block, SensorIds>();
57              var jMapping = (JObject)JsonConvert.DeserializeObject(File.ReadAllText(sensorsJson));
58              foreach (JProperty prop in jMapping.Children())
59              {
60                  var n = prop.Name.Substring(0, prop.Name.Length - 2);
61                  var t = prop.Name.Substring(prop.Name.Length - 2, 2);
62                  var bl = Blocks.FirstOrDefault(b => b.Name == n);
63
64                  if (bl.Type == BlockType.Handover)
65                      continue;
66
67                  SensorMapping.TryGetValue(bl, out var pi);
68                  if (t == "_b")
69                      pi.Begin = prop.Value.ToObject<int>();
70                  if (t == "_e")
71                      pi.End = prop.Value.ToObject<int>();
72                  SensorMapping[bl] = pi;
73              }
74          }
75
76          private void SetRail(IRail src, string prop, IRail dest)
77              => src.GetType().GetProperty(prop, BindingFlags.Public | BindingFlags.Instance)?.SetValue(src, dest);
78      }
79  }
```

Listing 36: Quellcode /stwgui/StateRenderer.cs

```
1   using Shared;
2   using System;
3   using System.Collections.Generic;
4   using System.Drawing;
5   using System.Linq;
6   using System.Windows.Forms;
7
8   namespace stwgui
```

```
 9 {
10     internal class StateRenderer : Panel
11     {
12         private IState state;
13 
14         const int ITEM_HEIGHT = 30;
15 
16         public Train Assign { get; set; }
17         public Block PathFrom { get; set; }
18         public Block PathTo { get; set; }
19 
20         public event EventHandler Assigned;
21 
22         public StateRenderer(IState state)
23         {
24             this.state = state;
25             DoubleBuffered = true;
26         }
27 
28         private StateRenderer()
29         {
30         }
31 
32         protected override void OnPaint(PaintEventArgs e)
33         {
34             if (state == null)
35                 return;
36 
37             e.Graphics.Clear(DefaultBackColor);
38             var y = 0;
39             var width = e.Graphics.ClipBounds.Width;
40             var font = DefaultFont;
41 
42             foreach (var block in state.Blocks)
43             {
44                 var brush = block.Occupied != null ? Brushes.Red : Brushes.Green;
45                 brush = block.Reserved && block.Occupied == null ? Brushes.DarkOrange : brush;
46                 if (PathFrom == block)
47                     brush = Brushes.DarkRed;
48                 else if (PathTo == block)
49                     brush = Brushes.DarkGreen;
50                 if (Assign != null && block.Type != BlockType.Station)
51                     brush = Brushes.LightGray;
52 
53                 e.Graphics.FillRectangle(brush, 0, y, width, ITEM_HEIGHT);
54                 e.Graphics.DrawRectangle(Pens.Black, 0, y, width, ITEM_HEIGHT);
55 
56                 e.Graphics.DrawString(block.Name, font, Brushes.Black, 0, y);
57 
58                 var text = "[" + block.Type.ToString() + "]";
59                 var size = e.Graphics.MeasureString(text, font);
60                 e.Graphics.DrawString(text, font, Brushes.Black, width - size.Width, y);
61 
62                 if (block.Occupied != null)
63                     e.Graphics.DrawString(block.Occupied.Locomotive.Name
64                         + " [" + block.Occupied.WagonCount + "] "
65                         + (block.Occupied.Locomotive.TurnDirection ? "-->" : "<--"),
66                         font, Brushes.Black, 0, y + ITEM_HEIGHT / 2);
67 
68                 y += ITEM_HEIGHT;
69             }
70         }
71 
72         protected override void OnMouseClick(MouseEventArgs e)
73         {
74             var idx = e.Y / ITEM_HEIGHT;
75             if (idx >= state.Blocks.Count || idx < 0)
76                 return;
77 
78             var block = state.Blocks[idx];
79 
80             if (Assign != null && block.Type == BlockType.Station)
81             {
82                 block.Occupied = Assign;
83                 Assign = null;
84                 Assigned?.Invoke(this, null);
85             }
86             else if (PathFrom == null)
87             {
88                 if (block.Occupied == null || block.Reserved)
89                     return;
90 
91                 PathFrom = block;
92                 Invalidate();
93             }
94             else if (PathTo == null)
95             {
96                 if (block.Occupied != null || block.Reserved)
97                     return;
98 
99                 PathTo = block;
100                Invalidate();
101            }
102            base.OnMouseClick(e);
103        }
104 
105        public void ResetSelection()
106        {
107            PathFrom = null;
108            PathTo = null;
109            Invalidate();
110        }
111    }
112 }
```

Listing 37: Quellcode /stwgui/TargetSelector.cs

```
using Shared.Networking;
using System;
using System.Windows.Forms;

namespace stwgui
{
    public partial class TargetSelector : Form
    {
        public string Target { get; set; }

        private TargetSelector()
        {
            InitializeComponent();
        }

        public TargetSelector(InterlockingClient client) : this()
        {
            var options = client.ListAvailable();
            comboBox1.Items.AddRange(options);
        }

        private void okButton_Click(object sender, EventArgs e)
        {
            Target = (string)comboBox1.SelectedItem;
        }
    }
}
```

Listing 38: Quellcode /trainunit/DccLocoProps.cs

```
using System;
using System.Collections.Generic;

namespace Shared
{
    internal class DccLocoProps
    {
        public string Name { get; set; }

        public int DccAddress { get; private set; }

        public byte[] DccPrepareFunctions { get; set; }

        public byte DccMaxFs { get; private set; }

        public bool TurnDirection { get; set; }

        public DccLocoProps(string name, int dccAddress, byte dccMaxFs, byte[] dccFunctions)
        {
            Name = name;
            DccAddress = dccAddress;
            DccPrepareFunctions = dccFunctions;
            DccMaxFs = dccMaxFs;
        }
    }
}
```

Listing 39: Quellcode /trainunit/LocoServer.cs

```
using Shared;
using Shared.Networking;
using System;
using System.Collections.Generic;
using System.Linq;

namespace LocoDriver
{
    internal class LocoServer : BaseServer
    {
        private Z21LocoDriver driver;
        private DccLocoProps loco;

        public LocoServer(int port, DccLocoProps loco) : base(port)
        {
            driver = new Z21LocoDriver("192.168.2.15", 21105);
            driver.RegisterLoco(loco);
            this.loco = loco;
        }

        protected override string ProcessRequest(string result)
        {
            var cmd = result.Split(';');
            if (cmd.Last() != "end" || cmd.Length < 2)
                return "-1";
            if (cmd[0] == "ma")
            {
                var direction = (MovementDirection)int.Parse(cmd[1]);
                Console.WriteLine("Got MA: " + direction.ToString());
                driver.MoveLoco(loco, direction);
                return "0";
            }
            else if (cmd[0] == "name")
                return loco.Name;
            return "-1";
        }
    }
}
```

Listing 40: Quellcode /trainunit/Program.cs

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using Shared;
using System;
using System.Collections.Generic;
using System.IO;

namespace LocoDriver
{
    static class Program
    {
        public static void Main(string[] args)
        {
            if (args.Length < 2)
            {
                Console.WriteLine("trainunit.exe <path to loco file> <port>");
                return;
            }

            var jloco = (JObject)JsonConvert.DeserializeObject(File.ReadAllText(args[0]));
            var loco = jloco.ToObject<DccLocoProps>();
            loco.DccPrepareFunctions = ((JArray)jloco.GetValue("DccPrepareFunctions")).ToObject<byte[]>();

            Console.WriteLine("Train controller: " + loco.Name);

            var port = int.Parse(args[1]);

            var server = new LocoServer(port, loco);
            server.ConnectionError += (s, e) => Console.WriteLine("Connection Error: " + e.Data);

            server.StartListening();

            Console.WriteLine($"Server started on port {port}! Press any key to exit...");
           Console.ReadKey();
        }
    }
}
```

Listing 41: Quellcode /trainunit/Z21LocoDriver.cs

```
using Shared;
using System;
using System.Collections.Generic;

namespace LocoDriver
{
    internal class Z21LocoDriver
    {
        private Z21.Z21Station z21;

        public Z21LocoDriver(string address, int port)
        {
            z21 = new Z21.Z21Station(address, port, false);
        }

        public void RegisterLoco(DccLocoProps loco)
        {
            var a = (short)loco.DccAddress;
            // An der Zentrale anmelden -> erhält Nachrichten zu Loks
            z21.GetLocoInfo(a);
            foreach (var fn in loco.DccPrepareFunctions)
                z21.SetLocoFunction(a, (byte)fn, Z21.FunctionType.On);
        }

        public void MoveLoco(DccLocoProps loco, MovementDirection move)
        {
            var a = (short)loco.DccAddress;
            var dir = move != MovementDirection.Backward ? Z21.Direction.Vorwärts : Z21.Direction.Rückwärts;
            byte fs = move != MovementDirection.Stop ? loco.DccMaxFs : (byte)0;
            z21.SetLocoDrive(a, dir, fs);
        }

        public void ForceStop()
        {
            z21.SetStop();
        }
    }
}
```

Abbildungsverzeichnis

Alle Aufnahmen und Grafiken: Manuel Huber

Literaturverzeichnis

[1] Bundesrepublik Deutschland, „Eisenbahn-Bau- und Betriebsordnung", 2017.

[2] Dr. Freiherr v. Roell, „Signalwesen", in *Enzyklopädie des Eisenbahnwesens*, 2. Aufl. Berlin; Wien: Urban & Scharzenberg, 1921, Bd. 9, S. 56. [Online]: http://www.zeno.org/Roell-1912/K/roell-1912--091-0056 (zuletzt abgerufen am 29.10.2018).

[3] Deutscher Bundestag (Hrsg.), „Antwort der Bundesregierung auf die Kleine Anfrage der Abgeordneten Stephan Kühn, Dr. Anton Hofreiter, Winfried Hermann, weiterer Abgeordneter und der Fraktion BÜNDNIS 90/DIE GRÜNEN – Drucksache 17/4749", 2011. [Online]: http://dip21.bundestag.de/dip21/btd/17/049/1704966.pdf (zuletzt abgerufen am 30.10.2018).

[4] P. Thomas, „Punktförmige Zugbeeinflussung : Wie sicher ist das deutsche Schienennetz?", *Frankfurter Allgemeine Zeitung*, 2013. [Online]: http://www.faz.net/aktuell/technik-motor/technik/punktfoermige-zugbeeinflussung-wie-sicher-ist-das-deutsche-schienennetz-12369658.html (zuletzt abgerufen am 30.10.2018).

[5] Untersuchungszentrale der Eisenbahn-Unfalluntersuchungsstelle des Bundes, „Untersuchungsbericht. Zugkollision, Überleitstelle Hordorf / Strecke Magdeburg Hbf - Halberstadt am 29.01.2011", 2011. [Online]: https://www.eisenbahn-unfalluntersuchung.de/SharedDocs/Downloads/EUB/Untersuchungsberichte/2011/023_Hordorf.pdf?___blob=publicationFile (zuletzt abgerufen am 30.10.2018).

[6] DB Netz AG (Hrsg.), *Richtlinie 483. Punktförmige Zugbeeinflussungsanlagen bedienen; Allgemeiner Teil*, Berlin, 2014, S. Vf. [Online]: https://fahrweg.dbnetze.com/resource/blob/1356006/edc44d10b7f9d973e899eed6bfd25f41/rw_483-0101-data.pdf (zuletzt abgerufen am 20.09.2017).

[7] Bayerischer Landtag (Hrsg.), „Schriftliche Anfrage des Abgeordneten Dr. Martin Runge BÜNDNIS 90/DIE GRÜNEN. Drucksache 16/4700", 2010. [Online]: https://www.bayern.landtag.de/www/ElanTextAblage_WP16/Drucksachen/Schriftliche%20Anfragen/16_0004700.pdf (zuletzt abgerufen am 14.10.2018).

[8] „Deutsche Bahn AG. Regionalbereich Süd". [Online]: https://fahrweg.dbnetze.com/fahrweg-de/unternehmen/kontakt/regionale_ap/sued-1368496 (zuletzt abgerufen am 14.10.2018).

[9] Untersuchungszentrale der Eisenbahn-Unfalluntersuchungsstelle des Bundes, „Zwischenbericht. Zugkollision, 09.02.2016, Bad Aibling - Kolbermoor", 2017. [Online]: https://www.eba.bund.de/SharedDocs/Downloads/EUB/Zwischenberichte/2016/01_Zwischenbericht_Bad_Aibling.pdf?___blob=publicationFile&v=2 (zuletzt abgerufen am 04.09.2017).

[10] DB Netz AG (Hrsg.), *Richtlinie 436. Zug- und Rangierfahrten im Zugleitbetrieb durchführen*, Frankfurt am Main, 2015. [Online]: https://fahrweg.dbnetze.com/resource/blob/1355892/9e7f4084ba2db8f336b9852caf1a8f55/rw_436_neuherausgabe-data.pdf (zuletzt abgerufen am 31.08.2017).

[11] S. Carstens, *Mechanische Stellwerke 1. Funktion, Bauteile und Anordnung.* Fürstenfeldbruck: Verlagsgruppe Bahn, 2011, S. 72.

[12] M. Huber, „Die Wieslauterbahn - Bahnhof Bundenthal in Vorbild und Modell". [Online]: https://wieslautertalbahn.manuelhu.de/ (zuletzt abgerufen am 24.09.2018).

[13] U. Maschek, *Sicherung des Schienenverkehrs. Grundlagen und Planung der Leit- und Sicherungstechnik*, 4. Aufl. Wiesbaden: Springer Vieweg, 2018.

[14] C. Lütgens, „Grundlagen und Schaltplan - Die Theorie hinter der zweiten Lichtschranke". [Online]: http://www.christian-luetgens.de/eisenbahn/elektronik/ls2/grundlagen/Grundlagen.htm (zuletzt abgerufen am 27.11.2017).

[15] A. Freese und O. Matthäi, „Aktuelle Trends von IT/TK und LST in Eisenbahnprojekten", S. 28f., 2017. [Online]: https://www-docs.b-tu.de/fg-eisenbahn/public/Lehre/LV/Kolloquium/2017/LST_ITK_Bahn.pdf (zuletzt abgerufen am 06.05.2018).

[16] „Z21 LAN Protokoll Spezifikation". [Online]: http://www.z21.eu/content/download/1668/18229/file/Z21_LAN_Protokoll_V1.08.pdf (zuletzt abgerufen am 20.08.2017).

[17] „superhandy333/z21". [Online]: https://github.com/superhandy333/z21 (zuletzt abgerufen am 20.08.2017).

[18] ERA * UNISIG * EEIG ERTMS USERS GROUP, „ERTMS/ETCS. System Requirements Specification. Chapter 2. Basic System Description. SUBSET-026-2", S. 12–24, 2016. [Online]: https://www.era.europa.eu/filebrowser/download/493_en (zuletzt abgerufen am 26.09.2018).

[19] E. Anders und O. Lemke, „Informeller Vergleich zweier Why-Because-Analysen (am Beispiel des S-Bahn-Unfalls von Neufahrn)", S. 2, 2005. [Online]: http://ifev.rz.tu-bs.de/Mitarb/Lemke/Nefahrn_WBA-Vergleich_1.0.1.pdf (zuletzt abgerufen am 14.10.2018).

[20] J.-O. Voß, „Themendienst. Digitale S-Bahn Hamburg: Referenzprojekt für die Anwendung neuer Technologien auf der Schiene", 2018. [Online]: https://www.deutschebahn.com/resource/blob/3183678/a276f2c3ef8c9de0112c408355d141f0/TD-Digitale-S-Bahn-Hamburg-data.pdf (zuletzt abgerufen am 20.09.2018).

[21] W. Feldwisch, „Die Verkehrsprojekte Deutsche Einheit (VDE) – Schienenprojekte", in *ETR Spezial. Neu- und Ausbaustrecke Nürnberg – Berlin (VDE 8). Das größte Bahnprojekt Deutschlands*, 2017, S. 68–73. [Online]: https://www.eurailpress.de/fileadmin/user_upload/ETR_VDE8_verlinkt.pdf (zuletzt abgerufen am 08.10.2018).

[22] „Bauleistung - Änderung einer Konzession/eines Auftrags während ihrer/seiner Laufzeit - Verhandlungsverfahren", 2016. [Online]: http://ted.europa.eu/udl?uri=TED:NOTICE:225708-2016:PDF:DE:HTML (zuletzt abgerufen am 08.10.2018).

Erklärung zur Seminararbeit

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Neufahrn, den ....
Unterschrift