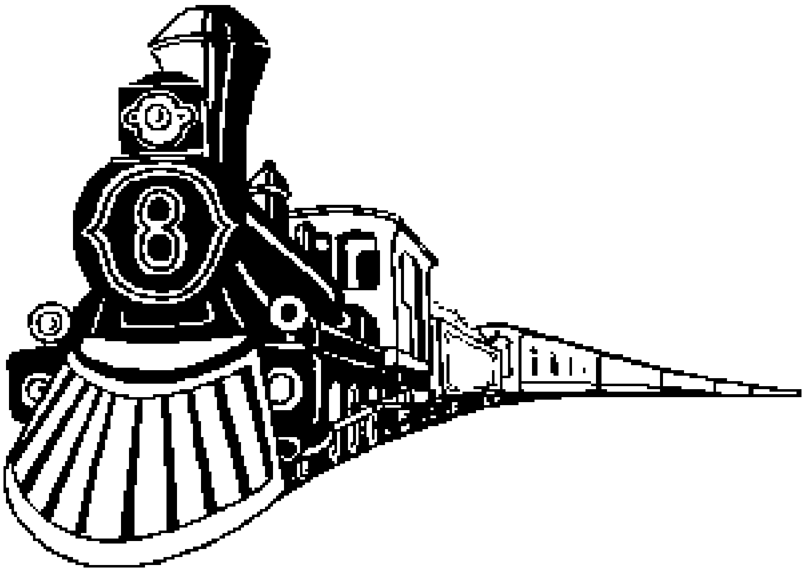

Entwurf und Implementation einer kollisionsvermeidenden Mehrzugsteuerung am Beispiel einer Modell-Straßenbahn

Studienarbeit
(Version 0.99)

eingereicht am 12. März 1997 von

Martin Meyer
Steffens Weg 18
21079 Hamburg
Matr.-Nr. 41 75 349

Zusammenfassung

Es wird eine computergestützte Mehrzugsteuerung für eine Modellbahn entwickelt, mit deren Hilfe ein Straßenbahnbetrieb unter Berücksichtigung des Konzeptes 'Fahrt auf Sicht' simuliert werden kann. Drohende Kollisionen werden innerhalb der Sichtweite erkannt und durch geeignete Maßnahmen vermieden.

Bei der Modellierung der Züge erfolgt eine schichtweise Abstraktion von der Hardware. Die Streckenführung kann benutzerseitig frei vorgegeben werden und wird mit Hilfe eines Graphenmodells beschrieben.

Die aktuelle Fahrzeugposition auf der Gleisanlage wird ermittelt und an den Steuerungsrechner übertragen. Eine drohende Kollision zwischen Zügen wird durch geeignetes Einfärben von Teilen des Gleisplangraphen erkannt.

Für mögliche Kollisionen wird ein Schema zur Einordnung in wenige Typklassen entwickelt, mit dessen Hilfe eine einfache Kollisionsvermeidung möglich wird. Der Benutzer erhält die Möglichkeit, die Züge vom PC aus direkt anzusteuern und wird über Aktionen des Systems informiert.

Inhaltsverzeichnis

Kapitel 1

Einführung

Die vorliegende Arbeit entstand am Arbeitsbereich Technische Informatik im Rahmen eines Projektes zur Implementierung technischer Prozesse. Es geht um die computergestützte Steuerung einer Modellbahnanlage unter Nachbildung eines Straßenbahnbetriebs.

1.1 Steuerungskonzepte für Züge

Im Bereich der Steuerung von Schienenfahrzeugen existieren verschiedene Konzepte zur Verkehrsregelung. Während im Bereich des europäischen Fernverkehrs nahezu ausschließlich eine Block-Signalsteuerung zum Einsatz kommt, bietet sich im innerstädtischen Personennahverkehr über kurze Distanzen, etwa für Schienenbusse und Straßenbahnen, eine einfachere Alternative an: Die Fahrt auf Sicht.

1.1.1 Block-Signalsteuerung

Beim Fernzugverkehr werden im Vergleich zum Straßenbahnbetrieb wesentlich höhere Geschwindigkeiten erreicht. Das führt zu erheblich höheren Brems- und Anhaltewegen, so daß die Zugführer das Fahrzeug nicht mehr innerhalb der einsehbaren Strecke zum Stillstand bringen können. So wird eine zentrale Steuerung erforderlich.

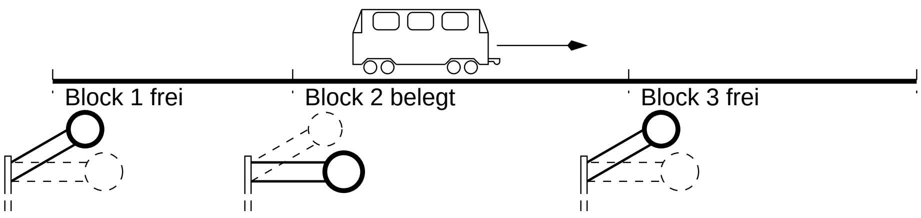

Abbildung 1.1: Schema einer Blocksignalsteuerung

Die Strecke ist in Blockabschnitte eingeteilt, die durch Signale abgetrennt werden. Durch eine zentrale Signalsteuerung wird sichergestellt, daß sich in einem Abschnitt stets nur ein Zug befindet. Eine ausführliche Schilderung findet sich in [10]

1.1.2 Fahrt auf Sicht

Im Straßenbahnbetrieb steht ein großer Teil der Gleise nicht allein dem Schienenverkehr zur Verfügung, sondern wird auch von anderen Verkehrsteilnehmern befahren. Daher ist stets mit Hindernissen auf dem Gleiskörper oder vorübergehend durch andere Verkehrsteilnehmer blockierte Streckenabschnitte zu rechnen. Solche Einflüsse können selbstverständlich nicht durch Signalanlagen erfaßt werden.

Beim Konzept 'Fahrt auf Sicht' steuert der Zugführer sein Fahrzeug autark und kann auf Ereignisse in seinem Gesichtsfeld unmittelbar reagieren. Er kann seine Geschwindigkeit den Strecken- und Sichtverhältnissen anpassen, Hindernisse auf dem Gleiskörper erkennen und drohende Kollisionen abwenden.

Bei Begegnung zweier (Schienen-)Fahrzeuge können die Zugführer sich ggf. durch Zeichen verständigen und sich gegenseitig den Weg freigeben.

Auch kann die Weichenstellung im Bedarfsfall vor Ort vom Zugführer selbst vorgenommen werden.

1.2 Aufgabenstellung

Ziel dieser Arbeit ist es, einen Straßenbahnbetrieb im Modell nachzubilden. Dabei soll eine 'Fahrt auf Sicht' simuliert werden. Technische Randbedingungen der einzelnen Triebwagen, wie etwa erreichbare Höchstgeschwindigkeit, Beschleunigungsvermögen u.ä., sind vorgegeben. Die Streckenführung soll benutzerseitig bestimmt werden können.

Es ist ein Programm zu implementieren, das zum einen die genannten Abläufe simuliert, andererseits aber auch dem Benutzer Möglichkeiten zur interaktiven Steuerung eines Zuges analog den Möglichkeiten eines Zugführers einräumt.

Besonderes Augenmerk soll auf die Erkennung und Vermeidung von Kollisionen zwischen Triebfahrzeugen gelegt werden. Auch dabei soll das Konzept 'Fahrt auf Sicht' nachgebildet werden. Ziel ist es nicht, eine möglichst effiziente Steuerung des Zugbetriebes zu erreichen. Stattdessen soll in dieser Arbeit auf der Grundlage des beschriebenen Konzeptes eine computergestützte Modellbahnsteuerung aufgebaut werden, die einen realitätsnahen Straßenbahnbetrieb mit der bewußt gewählten Beschränkung einer begrenzte Sichtweite modelliert.

1.3 Übersicht

Im folgenden werden zunächst die technischen Voraussetzungen für diese Arbeit beschrieben. Daran anschließend wird die Abbildung der realen Gegebenheiten wie z.B. der Streckenführung, der Triebwagen und den bestehenden Randbedingungen im Rechner erläutert, um auf dieser Basis im folgenden Kapitel Möglichkeiten zur Kollisionserkennung darzustellen. Der fünfte Abschnitt zeigt dann Strategien zur Vermeidung erkannter Kollisionen auf. Abschließend wird ein Versuch unternommen, die aufgeführten Verfahren zu beurteilen und Probleme bei der Anwendung aufzuzeigen.

Kapitel 2

Technische Voraussetzungen

Zur Darstellung des Straßenbahnbetriebs kommt eine Modellbahn in der Spurgröße H0, entsprechend einem Maßstab von 1:87, zum Einsatz. Dabei handelt es sich um ein Digitalsystem der Firma 'Zimo', das eine RS232-Schnittstelle zur computergestützten Bedienung bereitstellt.
So können die Triebfahrzeuge von einem PC als Steuerungsrechner einzeln angesteuert werden.

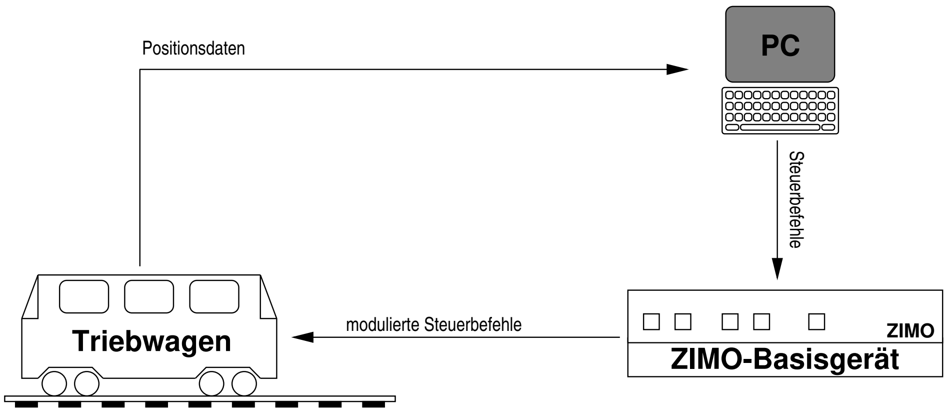

Abbildung 2.1: Datenfluß auf der Anlage

Jeder Triebwagen ermittelt fortlaufend Daten zu seiner Position, die auf Anforderung dem Steuerungsrechner übermittelt werden. Der PC berechnet daraus die aktuelle Position auf der Gleisanlage, führt eine Kollisionsprüfung durch und ergreift ggf. Maßnahmen zur Kollisionsvermeidung.

Die einzelnen Bestandteile sollen im folgenden genauer erläutert werden.

2.1 Das ZIMO-System

Die Ansteuerung der Triebwagen erfolgt vom PC aus über ein Zimo–Basisgerät vom Typ M1000. Dabei geht es vor allem darum, Züge unabhängig von ihrem Standort auf der Gleisanlage einzeln anzusprechen. Notfalls muß allerdings auch die Möglichkeit gegeben sein, den Zugbetrieb auf der Anlage komplett zu unterbrechen, ohne jeden Triebwagen einzeln anhalten zu müssen.

Dementsprechend lassen sich die vom Zimo-Basisgerät erkannten Befehle in drei Gruppen unterteilen:

- **Fahrbefehle** dienen der Ansteuerung einzelner Triebfahrzeuge. Mit ihnen werden die aktuelle Fahrtgeschwindigkeit, die Fahrtrichtung sowie einige Zusatzfunktionen gesteuert.

- **Konditionierungsbefehle** dienen zur Festlegung von Parametern einzelner Züge, die nicht ständigen Änderungen unterworfen sind. Dies sind z.B. das Beschleunigungs- oder Bremsverhalten, die Begrenzungen für Mindest- und Höchstgeschwindigkeiten der einzelnen Züge sowie weitere Zusatzfunktionen.

- **Sammelbefehle** erreichen alle auf der Anlage befindlichen Züge. Z. Zt. ist lediglich ein Befehl für Einrichtung und Aufhebung eines Sammelstops implementiert, um alle Züge auf der Anlage unverzüglich anzuhalten.

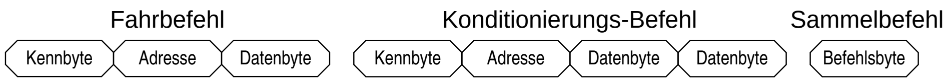

Abbildung 2.2: Aufbau der ZIMO-Befehle

Während Sammelbefehle nur aus einem einzigen Byte – dem Befehl selbst – bestehen, folgen auf die Kennung der Fahr- und Konditionierungsbefehle, wie in Abbildung 2.2 gezeigt, die Adresse des anzusprechenden Fahrzeugs und den betreffenden Fahrzeugdaten.

Diese Steuerbefehle werden im PC generiert und über die serielle Schnittstelle zum Zimo–Basisgerät gesendet. Aufgabe des Basisgerätes ist es, diese Daten an die Züge weiterzuleiten, indem die Steuersignale geeignet codiert[1] und der Versorgungsspannung überlagert werden (Abbildung 2.3).

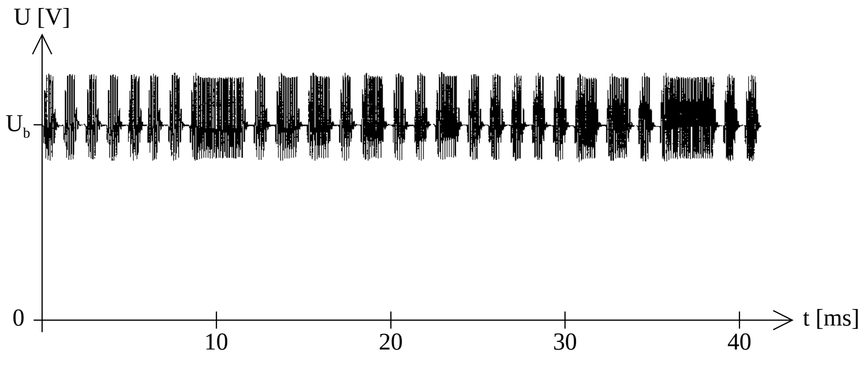

Abbildung 2.3: Betriebsspannung mit überlagerten Steuersignalen

[1] Verwendet wird Pulse Code Modulation; verschieden lange Pulse einer festen Frequenz werden nacheinander ausgesendet

Auf jedem Zug befindet sich ein Zimo-Empfängerbaustein mit einer systemweit eindeutigen Adresse, der die Steuersignale decodiert und, wenn das betreffende Fahrzeug angesprochen ist, den Befehl ausführt.

Zur Sicherung der Datenübertragung über den störanfälligen Weg Schiene – Radsatz sieht das Zimo-System eine ständige zyklische Wiederholung aller aktuellen Fahr- und Konditionierungsbefehle vor. Dies führte in der Praxis zu Problemen bei Ansteuerungen der Sonderfunktionen und zu einer Einschränkung der erreichbaren Datenrate auf 10 Fahr-/Konditionierunsbefehle pro Sekunde.

Um diese Nachteile zu umgehen wurde die automatische Befehlswiederholung im Zimo–Basisgerät entfernt. Im Fehlerfall ist es nun Aufgabe des Steuerungsrechners, den betreffenden Befehl zu wiederholen.

2.2 Positionsermittlung

Im Rahmen eines Projektes des Arbeitsbereiches TECH wurde im Jahr 1994 eine Methode entwickelt, um eine Positionsermittlung auf der Gleisanlage zu ermöglichen.

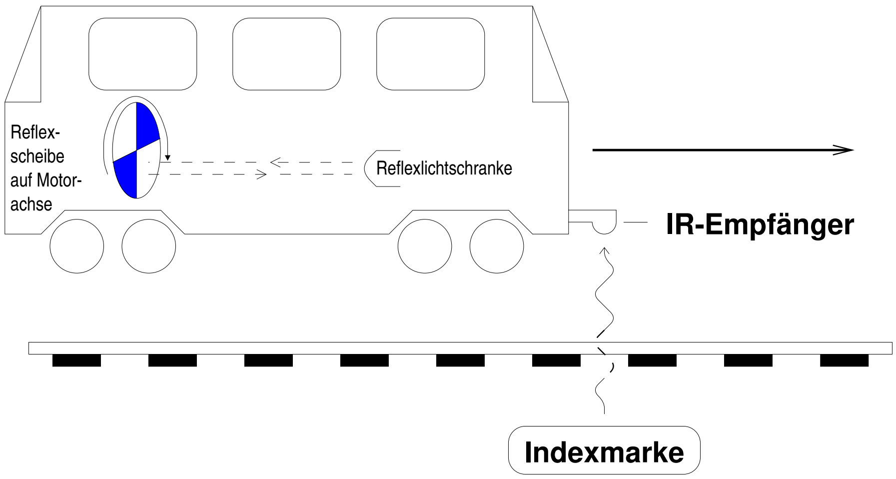

Abbildung 2.4: Positionsermittlung auf dem Triebwagen

Bei dem in Abbildung 2.4 dargestellten Verfahren wird bei jedem Triebfahrzeug auf der Motordrehscheibe ein Kontrastmuster angebracht, so daß mittels einer Reflexlichtschranke die Motorumdrehungen in geeigneten (nicht normierten) Einheiten gezählt werden können. Wird hierbei die Fahrtrichtung berücksichtigt, so kann man Abstandsmessungen mit hinreichender Genauigkeit durchführen[2]. Gleichzeitig wurden auf der Gleisanlage Initialisierungspunkte ('Indexmarken') installiert, die ihre systemweit eindeutige Kennung ständig über eine Infrarot-Diode aussenden. Beim Passieren eines Zuges wird das Signal von einem IR-Empfänger auf dem Triebwagen erkannt und gespeichert. Weiterhin wird beim Passieren einer Indexmarke der o.g. Abstandszähler zurückgesetzt.

[2]Das Verfahren liefert prinzipiell Genauigkeiten im Millimeterbereich. Allerdings treten – etwa bedingt durch Schlupf der Räder oder häufige Richtungsänderungen – weitere Fehler auf.

Der Abstand zwischen den Initialisierungspunkten muß so gewählt werden, daß ein Überlauf des Abstandszählers in jedem Fall ausgeschlossen werden kann.

Auf Anforderung können nun dem Steuerungsrechner die Kennung der letzten Indexmarke, der Abstand zu dieser Marke in Zähleinheiten und Informationen zur Zählrichtung übermittelt werden.

In Bild 2.5 wird dieses Verfahren am Beispiel deutlich: Das Fahrzeug hat die Indexmarke mit Kennung '0' passiert und wird an der bezeichneten Stelle einen Zählerstand von 100 erreicht haben. Solange keine Richtungsänderung eintritt, wird der Abstandszähler weiterlaufen, bis die Marke '2' erreicht wird.

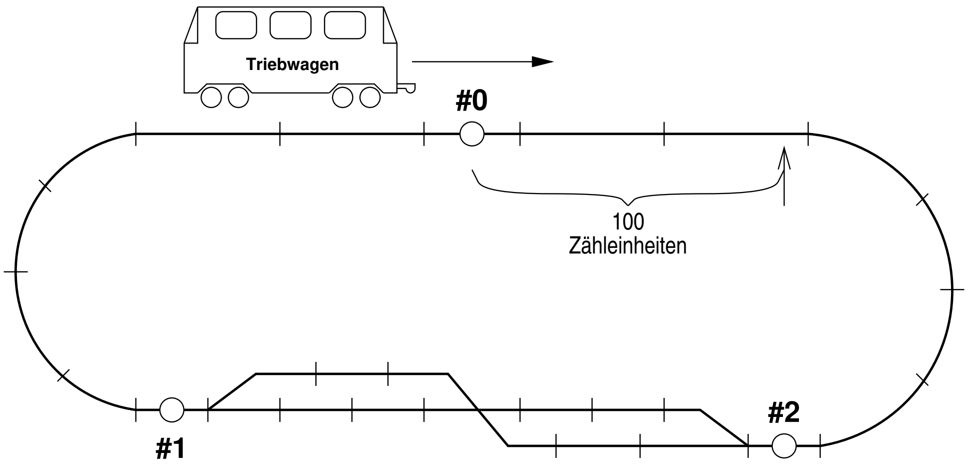

Abbildung 2.5: Beispiel zur Positionsermittlung

Würde das Triebfahrzeug jedoch vorher die Fahrtrichtung ändern, so kehrte sich die Zählrichtung einfach um, bis bei der Marke '0' erneut ein Reset zur Kalibrierung des Abstandszählers ausgelöst würde. Eine auftretende Ungenauigkeit – etwa durch Schlupf der Antriebsräder – könnte einen Überlauf des Abstandszählers zur Folge haben. Diese Möglichkeit muß bei der weiteren Verarbeitung der Daten berücksichtigt werden [3].

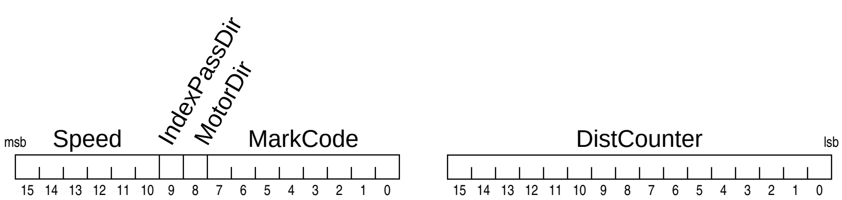

Abbildung 2.6: Auf der Lok gespeicherte Dateworte

Die folgenden Daten werden während der Fahrt ermittelt und stehen auf der Lok zum Abruf bereit. Der Aufbau der entsprechenden Datenworte ist in Bild

[3]Ein Zählerüberlauf wird zwar auf der Lok erkannt, aber nicht an den Steuerungsrechner gemeldet. Einzige Möglichkeit, diesen Sonderfall zu erkennen, ist daher momentan nur das Auftreten von Zählerständen nahe des Überlaufbereiches. Bei der Positionierung der Indexmarken ist dafür Sorge zu tragen, daß derartig hohe Zählerstände im Normalbetrieb nicht erreicht werden.

2.6 dokumentiert.

- Abstand zur letzten Indexmarke in einer nicht normierten, triebwagenspezifischen Zähleinheit ('DistCounter', 16 Bit)
- Kennung der letzten passierten Indexmarke ('MarkCode', 8 Bit)
- aktuelle Motordrehrichtung ('MotorDir', 1 Bit)
- Motordrehrichtung bei Passieren der letzten Indexmarke ('IndexPassDir', 1 Bit)
- aktuelle Geschwindigkeit ('Speed', 6 Bit – aus Platzmangel noch nicht implementiert)

Eine weitere Beschränkung für die Ansteuerung der Züge ergibt sich durch die Indexmarken auf der Gleisanlage: Da deren Kennung seriell ausgesendet wird, darf ein Triebfahrzeug seine Fahrtrichtung nicht ändern, während sich der IR-Empfänger, der dieses Signal erkennen soll, über der Indexmarke befindet.

2.3 Übertragung der Positionsdaten

Im vorigen Abschnitt wurde die Ermittlung von Positionsdaten beschrieben. Diese Daten sollen zur weiteren Verarbeitung an den Steuerungsrechner übertragen werden. Da das Zimo-System hierzu keine direkte Möglichkeit bietet, wurde im Vorfeld dieser Studienarbeit eine Lösung erarbeitet, die auf einfache Weise eine Datenübertragung über eine Infrarot-Strecke mit einer Reichweite von bis zu 6 Metern gestattet.

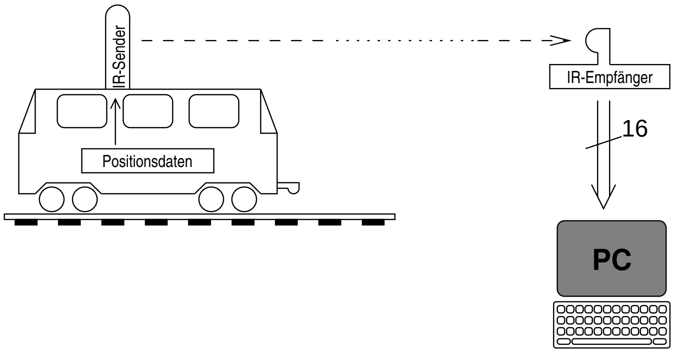

Abbildung 2.7: Datenübertragung über IR-Strecke

Aus Gründen der Platzersparnis mußte senderseitig ein möglichst einfach zu realisierendes Codierungsverfahren gewählt werden, das in Bild 2.8 dargestellt ist:

Dabei wird auf jeder Lok eine Infrarot-Diode mit einem 38 kHz-Takt angesteuert. Eine logische Null wird durch eine, eine Eins durch zwei Folgen von je 8 Pulsen[4] codiert. Aufgrund von technischen Randbedingungen kann der IR-Empfänger maximal ein Pulspaket in 800 μs empfangen, so daß zwischen den einzelnen Paketen eine Pause von 24 Takten erforderlich wird.

[4]Die Länge der einzelnen Pulsfolgen wurde entsprechend der Leistungsfähigkeit des IR-Empfängers empirisch bestimmt.

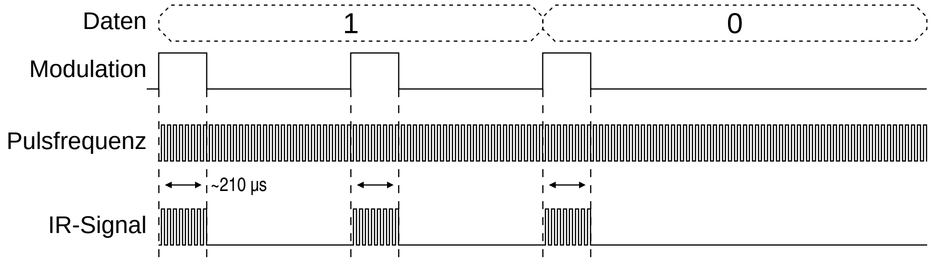

Abbildung 2.8: Codierung der IR-Daten auf Senderseite

Zum Empfang dieser Signale wird ein Baustein der Fa. Sharp aus der Fernsehtechnik eingesetzt. Dieser Empfangsbaustein liefert ohne zusätzlichen Schaltungsaufwand ein TTL-Ausgangssignal, das nach Empfang einer IR-Pulsfolge für ca. $400\mu s$ abfällt. Nach weiteren $400\mu s$ ist der Baustein empfangsbereit für die nächste Pulsfolge. Das Signal kann – wie in Abbildung 2.9 schematisch dargestellt – auf digitale Weise decodiert werden. Eine geeignete Schaltung wurde unter Verwendung von feldprogrammierbaren Logikbausteinen (FPGAs) realisiert und ist in Anhang C gezeigt.

Nach dem dargestellten Verfahren sind zur Übertragung eines Datenbits zwei Pulspakete erforderlich. Bedingt durch das Zeitverhalten des Sharp-Empfängerbausteins errechnet sich die dafür mindestens benötigte Zeit zu

$$T = 2 * (\overbrace{400\mu s}^{Low-Phase} + \overbrace{400\mu s}^{Karenzphase}) = 1,6ms$$

und die maximal erreichbare Datenrate zu

$$\varrho = \frac{1}{T} = 625\frac{Bit}{sec.}$$

Zur Aufnahme der Daten steht eine 16 Bit breite Datenleitung zum Steuerungsrechner zur Verfügung, die über eine rechnerseitig installierte zusätzliche Steckkarte einen schnellen Datentransfer sichert[5]. Alternativ ist aber auch – etwa zu Testzwecken – eine serielle Kommunikation über eine RS232-Schnittstelle möglich.

Die Infrarot-Datenübertragung über längere Distanzen ist z.T. erheblichen Störeinflüssen ausgesetzt. Während sich gegenseitige Störungen der verschiedenen IR-Quellen auf der Anlage durch unterschiedliche Trägerfrequenzen und Codierungsverfahren sowie Begrenzung der Sendeleistung bei kurzer Reichweite ausschließen lassen, sind Störungen durch Tageslichteinflüsse, z.B. direkte Sonneneinstrahlung oder diffuse Reflektionen, sehr schwer zu begrenzen. Vor diesem Hintergrund wäre eine Verbesserung des Übertragungsprotokolls unter Anwendung fehlererkennender und -korrigierender Codes hilfreich. Ein solches Vorhaben erforderte allerdings zumindest senderseitig, also auf der Lok, eine hardwaremäßige Implementation entsprechender Algorithmen, für die z.Zt. der

[5]Dies mag angesichts der vergleichsweise langsamen Datenrate zur IR-Übertragung unnötig erscheinen, doch ist zum einen mittelfristig eine Verbesserung der Übertragungstechnik – auch aus Gründen der Übertragungssicherheit – erforderlich, zum anderen bietet die zusätzliche Steckkarte auch programmtechnisch einige Vorteile.

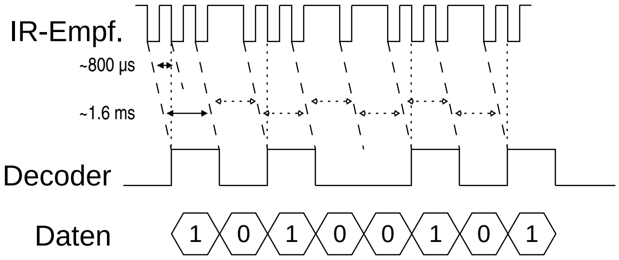

Abbildung 2.9: Decodierung der IR-Daten auf Empfängerseite

Platz fehlt. Somit bleibt als einzige Möglichkeit der Fehlererkennung eine softwaremäßige Plausibilitätskontrolle der empfangenen Daten bei der weiteren Verarbeitung im Steuerungsrechner.

2.4 Gleisplanung

Um unter den gegebenen Randbedingungen sinnvolle Ergebnisse zu erzielen, wurde zum Testbetrieb die in Abbildung 2.10 skizzierte einfache Streckenführung aufgebaut. Auf jedem Punkt dieser Strecke ist ein aufgrund der beschränkten Reichweite zu fordernder Maximalabstand zum IR-Empfängerbaustein von 6m eingehalten.

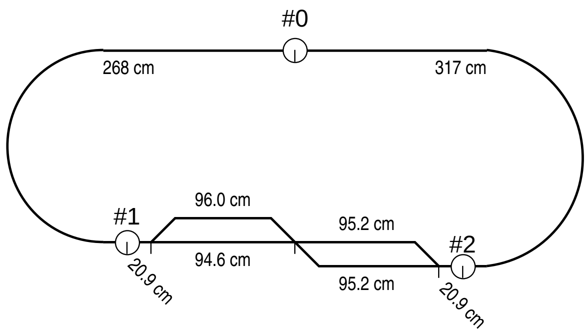

Abbildung 2.10: Gleisanlage mit Angabe der Streckenlängen

Zum gegebenen Zeitpunkt ist leider davon auszugehen, daß eine computergestützte Weichensteuerung nicht möglich ist, da die hierfür erforderliche kommerzielle Hardware nicht zur Verfügung stand. Eine betreffende Erweiterungsmöglichkeit wurde aber selbstverständlich berücksichtigt und softwaremäßig weitgehend vorbereitet.

Etwas aufwendiger ist die Realisierung von Kehrschleifen. Um die in Abschnitt 2.3 erwähnte Plausibilitätskontrolle zu ermöglichen, wurde beim Entwurf angenommen, daß innerhalb einer Kehrschleife oder eines vergleichbaren Gleisabschnitts mindestens eine Indexmarke positioniert wird[6], so daß ein Zug

[6]Darüberhinaus erfordert das Zimo-System ein zusätzliches Kehrschleifen-Modul, das zum Zeitpunkt dieser Arbeit nicht zur Verfügung stand.

niemals eine Kennung lesen kann, die mit seiner zuletzt gespeicherten identisch ist – es sei denn, er hat seine Fahrtrichtung geändert.

Unter Einhaltung dieser Randbedingungen ist die Streckenführung beliebig variierbar, solange sie dem Steuerungsrechner bekanntgemacht wird und die Streckenteile zusammenhängend sind[7]. Zur Beschreibung des Gleisplans wird im folgenden Kapitel ein Modell angeregt. Selbstverständlich können auch andere Beschreibungsmethoden verwendet werden, wenn eine entsprechende Routine zum Einlesen der Daten bereitsteht.

[7]Handelt es sich um unzusammenhängende Streckenabschnitte, so können diese o.B.d.A. getrennt voneinander als einzelne Gleispläne dargestellt werden.

Kapitel 3

Modellbildung

Basierend auf den dargestellten technischen Voraussetzungen soll im folgenden ein Ansatz zur Modellierung eines Straßenbahnbetriebs erarbeitet werden. Dabei sollen nur ausgewählte, für die Problemstellung relevante Aspekte der Realität nachgebildet werden.

Aufgabenstellung ist es nicht, einen bestehenden Schienenverkehr in irgendeiner Weise auf Effizienz zu optimieren, sondern es soll, wie in Abschnitt 1.2 dargelegt, im Ansatz eine 'Fahrt auf Sicht' nachgebildet werden.

Bei der Modellbildung muß also beachtet werden, daß Ereignisse, die außerhalb der Sichtweite eines Fahrzeugführers stattfinden, ignoriert werden sollen. Daraus entstehende lokale Verklemmungen werden nicht im Vorfeld vermieden, sondern, selbst um den Preis von vorübergehenden Fahrtunterbrechungen für betroffene Züge, in Kauf genommen.

Die Modellierung des Gesamtsystems erfolgt auf drei Säulen:

1. Ein **Modell des Gleisplans** soll die Streckenführung darstellen und eine Grundlage zur Beschreibung der Positionen einzelner Triebfahrzeuge liefern, so daß darauf aufbauend eine Kollisionsprüfung möglich wird.

2. Durch eine **Modellierung der Züge** soll eine weitgehend hardwareunabhängige Ansteuerung der einzelnen Triebfahrzeuge ermöglicht und das Fahrverhalten realer Straßenbahnen bei 'Fahrt auf Sicht' im Modell nachgebildet werden.

3. Darauf aufbauend soll eine **modellierte Betriebsüberwachung** durch einen Steuerungsrechner drohende Kollisionen zwischen Triebfahrzeugen erkennen und geeignete Maßnahmen zur Kollisionsvermeidung veranlassen. Entsprechende Konzepte werden in Kapitel 4 behandelt.
   In Erweiterung des ursprünglichen Konzeptes soll darüberhinaus eine interaktive Mehrzugsteuerung durch den Benutzer möglich sein, um kritische Situationen für Demonstrationszwecke zu provozieren. Diese Besonderheit soll jedoch nicht Thema der vorliegenden Arbeit sein; einige Hinweise zur Benutzungsoberfläche lassen sich Anhang A entnehmen.

Zur Modellierung wurde ein objektorientierter Ansatz gewählt und in C++ formuliert. Damit können Besonderheiten der zugrundeliegenden Hardware weitgehend verborgen werden.

3.1 Das Gleisplanmodell

An ein Gleisplanmodell müssen eine Reihe von Anforderungen gestellt werden, die im folgenden aufgeführt sind.

1. Es müssen beliebige Streckenführungen darstellbar sein.
2. Es muß eine Beschreibungsmethode für die Bezeichnung von Orten auf der Gleisanlage klar definiert sein.
3. Mit dieser Methode muß jeder beliebige Ort auf der Gleisanlage eindeutig zu bezeichnen sein.
4. Umgekehrt muß jeder mit dieser Methode beschriebene Ort (innerhalb des definierten Bereiches) auf der Gleisanlage eindeutig lokalisierbar sein.
5. Das Modell der Gleisanlage muß eine der Problemstellung angemessene Genauigkeit bieten[1].
6. Dynamische Veränderungen der Streckenführung zur Laufzeit[2] müssen auf vorab definierte Weise möglich sein.
7. Statische Veränderungen der Streckenführung müssen in der genannten Beschreibungsmethode implementiert werden können.
8. Die Kennung der einzelnen Indexmarken soll nach außen verborgen bleiben.

3.1.1 Notwendigkeit eines Gleisplanmodells

Allein die Angabe der in Abschnitt 2.2 vorgestellten Positionsdaten – Abstand zur letzten Indexmarke und deren Kennung – erfüllt diese Anforderungen nicht und bringt daher für die weitere Verarbeitung Probleme mit sich, die zur Motivation eines neuen Gleisplanmodells im folgenden kurz aufgezeigt werden sollen.

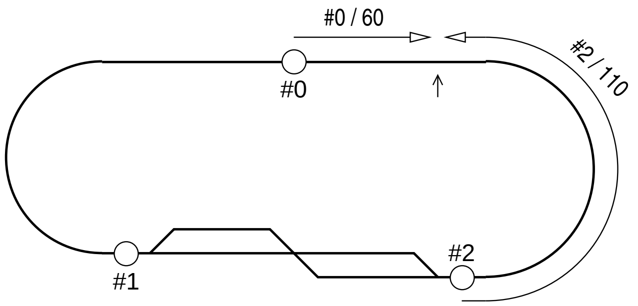

Abbildung 3.1: Position kann auf mehrere Arten codiert werden

[1] Da bei der Modellierung die Spurgröße der tatsächlichen Gleisanlage nicht festliegt, kann hier kein absoluter Wert angegeben werden. Für die zu dieser Arbeit genutzte Anlage im Maßstab 1:87 (Spurgröße H0) ist eine Genauigkeit im Millimeterbereich angemessen.

[2] Beispielsweise Umstellung von Weichen, Sperrung von Fahrstrecken usw.

- zu Forderung 3

  Wie anhand von Grafik 3.1 deutlich wird, kann dieselbe Position auf unterschiedliche Weise codiert werden:

  Wenn ein Punkt auf der Gleisanlage von zwei oder mehr Indexmarken aus erreichbar ist, so ist zu jeder dieser Indexmarken die Angabe der Kennung mit entsprechendem Abstand möglich und legitim. Damit wäre ein gegebener Ort nicht mehr eindeutig zu beschreiben.

- zu Forderung 4

  In Grafik 3.2 ist dargestellt, daß die Angabe des Abstandes zur letzten Indexmarke und deren Kennung nicht geeignet ist, jede Position auf der Gleisanlage eindeutig zu beschreiben.

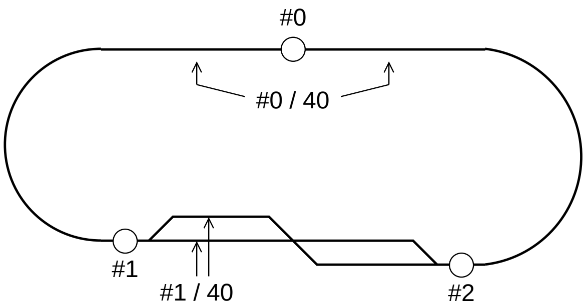

Abbildung 3.2: Positionsangabe nicht eindeutig

  Vielmehr können in Abhängigkeit sonstiger Faktoren[3] verschiedene Positionen mit denselben Daten bezeichnet werden. So wäre mit einer Beschreibung der Ort auf der Gleisanlage nicht mehr eindeutig festgelegt.

- zu Forderung 6

  Auf dem Triebfahrzeug sind keinerlei Informationen über die Streckenführung und die aktuelle Weichenstellung bekannt. Daher ist es nicht möglich, allein auf diesen Daten eine Plausibilitätskontrolle für die ermittelte Position durchzuführen.

Es muß also eine geeignetere Weise gefunden werden, die statische und dynamische Streckenführung sowie die Positionen der einzelnen Triebfahrzeuge darzustellen. Im folgenden wird ein Graphenmodell zur Streckenbeschreibung eingeführt, das diesen Forderungen genügt.

3.1.2 Das Graphenmodell

Der Gleisplan wird beschrieben als Netzwerk von **Knoten** und **Kanten**. Als Knoten werden alle ausgezeichneten Punkte auf der Strecke, wie z.B. Verzweigungen, Kreuzungen, Indexmarken, Streckenendpunkte u.ä. gewählt; darüberhinaus können beliebige weitere Punkte als Knoten beschrieben werden. Knoten werden ohne räumliche Ausdehnung angenommen.

[3] z.B. Weichenstellung

Kanten sind die zwischen zwei benachbarten Knoten liegenden Gleisabschnitte mit einer beliebigen, aber festen Länge. Um eine eindeutige Ortsangabe zu gewährleisten, muß für jede Kante eine Zählrichtung definiert werden. Bei der späteren Beschreibung des Gleisplans ist es zweckmäßig, aber nicht zwingend erforderlich, diese Zählrichtung auf der gesamten Anlage einheitlich festzulegen, sofern nicht etwaige Kehrschleifen eine Ausnahme erfordern.
Zur Verdeutlichung ist in Bild 3.3 ein willkürlich gewählter Gleisplan in Knoten und Kanten aufgeteilt.

Definition 3.1 (Knoten und Kante)

*Ein **Knoten** ist ein auf der Gleisanlage beliebig festzulegender Punkt ohne räumliche Ausdehnung.*

*Eine **Kante** ist ein zusammenhängender und verzweigungsfreier Teil des Gleiskörpers mit fester Länge $l > 0$, der an jedem Ende mit genau einem Knoten verbunden ist und auf dem eine Zählrichtung festgelegt ist.*

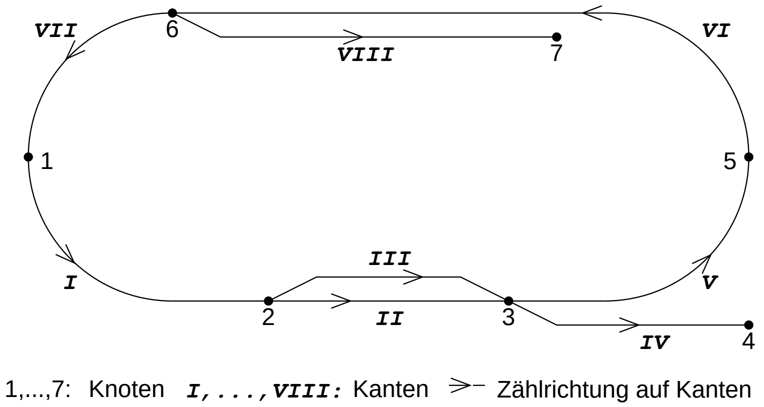

Abbildung 3.3: Aufteilung eines Gleisplans in Knoten und Kanten

Sowohl Knoten als auch Kanten können darüberhinaus noch eine Reihe von zusätzlichen Eigenschaften zugeordnet werden. Repräsentiert etwa ein Knoten eine Kreuzung oder Verzweigung, so werden Informationen zur aktuellen Weichenstellung benötigt, für Kanten können besondere Streckeneigenschaften wie etwa abschnittsweise Höchstgeschwindigkeiten relevant sein. Darüberhinaus darf ein Triebfahrzeug über Indexmarken und – aus Sicherheitsgründen – auch im Bereich von Weichen seine Fahrtrichtung nicht ändern[4].

Durch den objektorientierten Ansatz des Modells sind hier nahezu beliebige Erweiterungsmöglichkeiten gegeben, so daß eine Implementation dieses Modells immer unvollständig bleiben muß. Auf die gegenwärtig realisierten Möglichkeiten wird in Anhang A eingegangen.

Somit kann der Gleisplan durch Angabe eines endlichen, zusammenhängenden, und gerichteten Graphen ($G = (V, E)$) mit V als der Menge aller auf der

[4]Da im Modellbahnbetrieb ein 'aufschneiden' der Weichen möglich ist, könnte es bei einer Fahrtrichtungsumkehr zu einer Entgleisung kommen

Gleisanlage definierten Knoten[5] und $E \subseteq V \times V$ als Menge aller auf der Gleisanlage definierten Kanten beschrieben werden.

Da ein Graph unabhängig von der Art seiner Darstellung definiert ist (vgl. [5]), werden Einzelheiten der Streckenführung wie z.B. Kurven oder Steigungen durch dieses Modell nicht abgedeckt. Stattdessen werden die daraus resultierenden für die Aufgabenstellung relevanten Randbedingungen[6] bei der Implementation als zusätzliche Attribute berücksichtigt.

Knoten und Kanten werden jeweils fortlaufend durchnumeriert, so daß sie paarweise verschiedene Indizes erhalten. Da zudem eine eindeutige Zählrichtung auf den Kanten festgelegt wurde, ist die Position eines Triebfahrzeugs mit Angabe der Kantennummer und des relativen Abstandes vom Kantenursprung eineindeutig beschrieben.

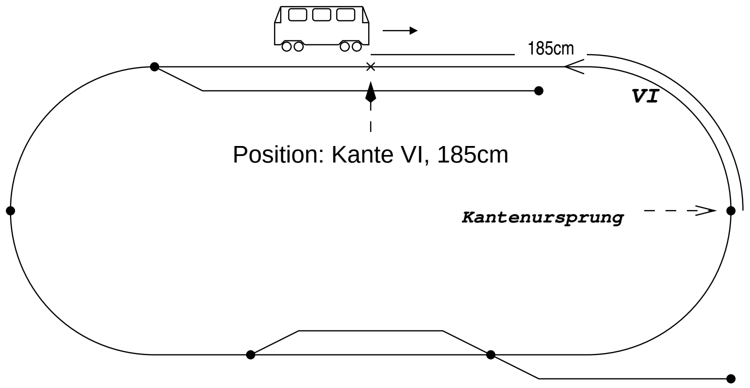

Abbildung 3.4: Positionsangabe im Graphenmodell

Da das beschriebene Modell die erreichbare Genauigkeit in keiner Weise einschränkt, ist die Präzision der Positionsangabe allein durch die Randbedingungen des in Abschnitt 2.2 erläuterten Verfahrens zur Ermittlung der Positionsdaten auf dem Triebwagen begrenzt. Die dort gewonnenen Daten müssen aufbereitet und in das Graphenmodell tranformiert werden. Ein geeigneter Algorithmus wird in Abschnitt 3.1.4 beschrieben. Weitere Details zur Implementation finden sich in Anhang A.

3.1.3 Gleisplanbeschreibung mit dem Graphenmodell

Im folgenden soll anhand des bereits in Bild 2.10 vorgestellten Gleisplans der Testanlage eine Gleisplanbeschreibung exemplarisch durchgeführt werden. Die Position der Indexmarken wurde bereits bei der Streckenkonzeption geeignet gewählt, so daß alle Randbedingungen eingehalten werden.

Zunächst werden alle ausgezeichneten Punkte auf der Strecke[7] als Knoten definiert, so daß die dazwischenliegenden Streckenabschnitte verzweigungsfrei sind. Diese Streckenabschnitte werden als Kanten definiert. Nun wird eine der Problemstellung angemessene Maßeinheit zur Positionsangabe festgelegt. Bei

[5]Da ein Knoten stets auf dem Gleiskörper definiert wird, ist seine Valenz immer größer als Null.

[6]beispielsweise Geschwindigkeitsbeschränkungen auf Streckenabschnitten

[7]Das sind hier Indexmarken, Weichen und die Kreuzung

der Größe der vorliegenden Anlage und den hardwareseitig gegebenen Randbedingungen erscheint eine Angabe der Strecken in Zehntel-Millimeter geeignet. Wird die Maßeinheit so klein gewählt, daß die absoluten Zahlenwerte für die Kantenlänge zu groß werden, so bestünde die Möglichkeit, beliebige weitere Knotenpunkte einzufügen um zu lange Kanten zu unterteilen. Dies ist im vorliegenden Fall jedoch nicht erforderlich.

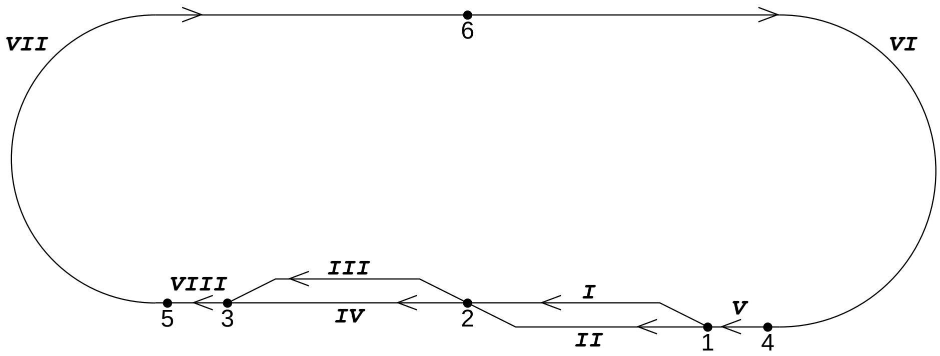

Abbildung 3.5: Gleisplan der Testanlage mit Knoten und Kanten

Knoten und Kanten werden nun fortlaufend numeriert, so daß die jeweiligen Indizes paarweise verschieden sind. Den Kanten wird dann die Länge des jeweiligen Streckenabschnitts zugeordnet und es wird eine Zählrichtung definiert[8]. So erhält man einen endlichen, zusammenhängenden und gerichteten Graphen mit den in Bild 3.5 bezeichneten Knoten und Kanten.

Kanten-index	Länge (in /10 mm)	Anfangs-knoten	End-knoten
I	9520	2	1
II	9520	2	1
III	9460	3	2
IV	9600	3	2
V	2090	1	4
VI	31700	4	6
VII	26800	6	5
VIII	2090	5	3

Abbildung 3.6: Kanten im Gleisplan der Testanlage

Analog dem hier gezeigten Vorgehen kann zu jedem zusammenhängenden Gleisplan ein entsprechender Graph gefunden werden.

3.1.4 Umrechnung der Positionsdaten

Anhand der bereits in Abschnitt 2.4 und 3.1.3 angesprochenen Test-Gleisanlage (vgl. Bild 2.10 und 3.5) soll beispielhaft die Umrechnung der vom Triebfahrzeug

[8]Zur Festlegung einer Zählrichtung reicht es aus, anzugeben bei welchem der beiden benachbarten Knoten der Kantenursprung sein soll

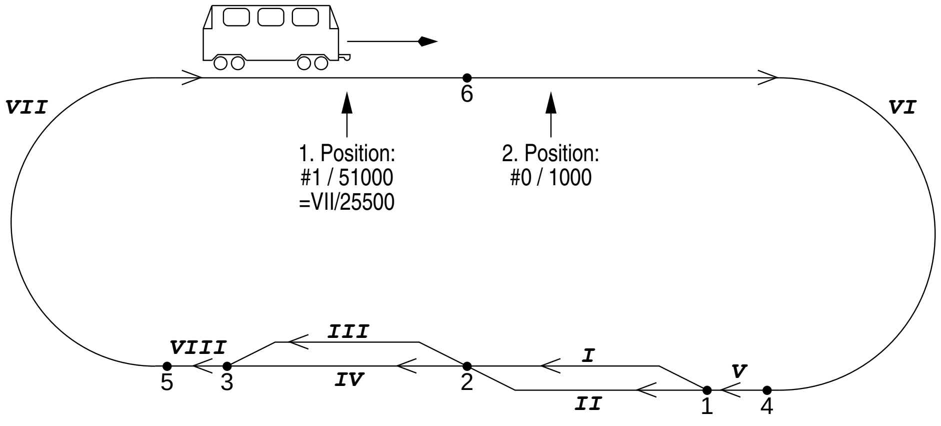

Abbildung 3.7: Beispiel zur Positionsumwandlung

empfangenen Positionsdaten in das Graphenmodell besprochen werden. Da die Ausgangsdaten keine eineindeutige Angabe zulassen, werden noch zusätzliche Informationen in Form der letzten bekannten Position, der seither verstrichenen Zeit sowie Fahrtrichtung und Fahrtgeschwindigkeit des Triebfahrzeugs benötigt. Die (willkürlich gewählte) Ausgangssituation ist in Bild 3.7 gezeigt.

Die genannten Zählerstände sind fiktiv und sollen den Wert des auf der Lok befindlichen Abstandszählers angeben. In diesem Beispiel entsprächen zwei Zähleinheiten auf der Lok einer Länge von $1\frac{mm}{10}$ im Graphenmodell. Die entsprechende Normalisierung der Daten ist eine Aufgabe bei der Modellierung der Züge.

Bei der dargestellten Situation ergäbe sich etwa der folgende Ablauf:

1. Die letzte bekannte Position wurde mit Indexmarke 1 und Zählerstand 51000 empfangen. Diese wurde in das Graphenmodell zu 'Kante VII, $25500\frac{mm}{10}$' umgerechnet.

2. Die neue Position wurde mit Indexmarke 0 und Zählerstand 1000 empfangen. Fahrtrichtung und Geschwindigkeit des Triebfahrzeugs sind seither unverändert geblieben; die verstrichene Zeit ist bekannt.

3. Anhand der Fahrtgeschwindigkeit und der zwischen den Positionsangaben verstrichenen Zeit wird ein Erwartungswert für die zurückgelegte Distanz berechnet.

4. Ausgehend von der letzten bekannten Position werden die nachfolgenden Knoten und Kanten im Graphenmodell sukzessive in Fahrtrichtung durchlaufen[9], und zwar, da sich die Indexmarke verändert hat, bis ein Knoten erreicht wird, der die neue Indexmarke repräsentiert. Die Gesamtlänge der nachgefahrenen Strecke wird dabei protokolliert. Wird die erwartete Distanz um eine vorgegebene Toleranzgrenze überschritten, so gilt die neu empfangene Position als fehlerhaft und der Vorgang bricht mit einem Fehlercode ab.

[9] unter Berücksichtigung der aktuellen Weichenstellung

Im Beispiel wäre der nächste Knoten in Fahrtrichtung Knoten 6, der auch die gesuchte Indexmarke trägt.

Anschließend wird eine dem Zählerstand entsprechende Distanz durchlaufen und die im Graphenmodell erreichte Position als vorläufige Position gespeichert. Im Beispiel wäre diese Distanz nach der o.a. Umrechnung $1000/2 = 500\frac{mm}{10}$, so daß sich als Position 'Kante VI, $500\frac{mm}{10}$ ergibt.

Wäre die Indexmarke unverändert geblieben, so hätte, ausgehend von der letzten bekannten Position, eine Distanz überfahren werden müssen, die der Differenz der beiden Zählerstände entspräche.

5. Für die Gesamtlänge der nachgefahrenen Strecke ergibt sich im Beispiel:

$$\underbrace{\overbrace{26800\frac{mm}{10}}^{\substack{\text{Kantenlänge}\\ \text{Kante VII}}} - 25500\frac{mm}{10}}_{\text{Distanz bis Knoten 6}} + \underbrace{500}_{\substack{\text{Distanz auf}\\ \text{Kante VI}}} = 1800\frac{mm}{10}$$

Diese Länge wird mit der unter Punkt 3 berechneten erwarteten Distanz verglichen. Wird die erwartete Distanz um eine vorgegebene Toleranzgrenze unterschritten[10], so wird die Strecke weiter nachgefahren, bis entweder die gesuchte Position unter Einhaltung der Toleranzgrenzen erreicht wird, oder das Verfahren in Schritt 4 wegen Überschreitung der Toleranzgrenze abbricht. So wird während der Umrechnung der Positionsdaten gleichzeitig eine Plausibilitätskontrolle durchgeführt, die grobe Fehler bei der Datenübertragung ausschließt.

Wird das Verfahren mit Fehlercode abgebrochen, so konnten die empfangenen Positionsdaten nicht mit der erwarteten Position in Einklang gebracht werden. Daher wird ein Übertragungsfehler als wahrscheinlich angenommen. In diesem Fall wird die sich aus der unter Schritt 3 berechneten Distanz ergebende Position als vorläufige Arbeitsgrundlage für die Kollisionserkennung angenommen. Um den Einfluß von Ungenauigkeiten bei der Positionsermittlung zu minimieren, gelten jedoch bei der nächstfolgenden Positionsumwandlung erneut die letzten sicher empfangenen und umgewandelten Daten als Ausgangspunkt.

Da Geschwindigkeits- und insbesondere auch Fahrtrichtungsänderungen der Triebfahrzeuge durch Trägheit der Fahrzeugmasse Ungenauigkeiten in der berechneten Position verursachen können, sollte bei jeder Geschwindigkeits- und Fahrtrichtungsänderung auch eine Positionsermittlung angestrebt werden[11].

3.2 Modellierung der Züge

Eine direkte Ansteuerung der Triebfahrzeuge wäre sehr starken Einflüssen der zugrundeliegenden Hardware ausgesetzt. Daher liegt es nahe, das gewünschte Verhalten der Züge in einem geeigneten Modell nachzubilden und Auswirkungen störender Seiteneffekte weitgehend zu verbergen.
Die Anforderungen an ein solches Modell sollen kurz zusammengefaßt werden:

[10] Eine Überschreitung der Toleranzgrenze wurde bereits durch eine Abbruchbedingung in Schritt 4 ausgeschlossen

[11] In der gegenwärtigen Implementation wird mit jedem Fahrbefehl auch die Übertragung der auf der Lok gespeicherten Positionsdaten veranlaßt

1. Es soll eine **einheitliche Schnittstelle** für alle Züge bereitgestellt werden, die eine Ansteuerung auf hochsprachlicher Programmierebene gestattet.
2. **Hardwaretechnische Besonderheiten** einzelner Triebfahrzeuge sollen verborgen werden.
3. Eine **Kapselung der Kommunikationswege zu und von den Triebfahrzeugen** wird angestrebt. Da jedoch keine Mechanismen zur Fehlererkennung zur Verfügung stehen, kann dieser Anspruch nur bedingt erfüllt werden.
4. Die auf den Triebfahrzeugen ermittelten fahrzeugabhängigen **Daten** zu Position und Fahrt sind zu **normalisieren**.
5. Im Rahmen der Aufgabenstellung sollen Methoden entwickelt werden, die mögliche Aktionen eines Lokführers ausführen.

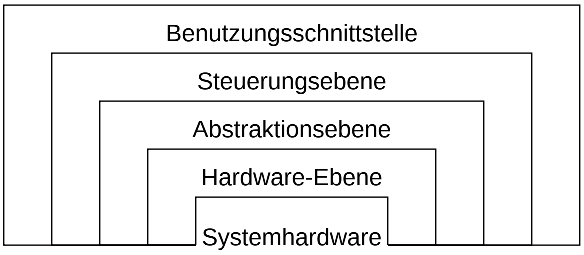

Abbildung 3.8: 5-Schichten-Modell

Diese Forderungen legen einen schichtweisen Modellaufbau nahe, wie er in Bild 3.8 skizziert ist. Die Aufgaben der dargestellten Ebenen sollen im folgenden erläutert werden:

1. Systemhardware
   Aufsatzpunkt ist die Hardware des Modellbahnsystems mit am Arbeitsbereich TECH entwickelten zusätzlichen Modulen zur Positionsermittlung, die über Sonderfunktionen des Zimo-Systems angesprochen werden, sowie der in Abschnitt 2.3 vorgestellte, ebenfalls am AB TECH entwickelte IR-Empfänger mit PC-Schnittstelle.
2. Hardware-Ebene
   Aufgabe dieser Schicht ist es, den Zugriff auf die Hardware zu ermöglichen. Dies geschieht durch Bereitstellung von Funktionen zur Steuerung des Zimo-Systems über RS232 und damit koordinierten Zugriffen über eine Multi-Port-IO-Karte auf den IR-Empfänger. Die entsprechenden Befehle werden in Abschnitt 2.1 sowie in Anhang B erläutert.

   Alle Fahrt- und Positionsdaten liegen in nicht normalisierter Form vor und können nicht zum Vergleich zwischen den Triebfahrzeugen herangezogen werden.

   Nach Möglichkeit sollte bereits auf dieser Ebene eine Fehlererkennung durchgefürt werden. Da jedoch bis dato die erforderliche hardwareseitige Unterstützung auf der Lok fehlt, kann zunächst nur eine rudimentäre

Kontrolle erfolgen, indem mit jedem Zimo-Fahrbefehl gleichzeitig die Sonderfunktion zur Übertragung der gespeicherten Positionsdaten aktiviert wird. Werden daraufhin Datenpakete empfangen, so wurde dieser Fahrbefehl auf der Lok erkannt.

Eine inhaltliche Prüfung der Positionsdaten kann aber z.Zt. nur durch eine Plausibilitätskontrolle auf der Abstraktionsebene erfolgen.

3. Abstraktionsebene
   Um einen hardwareunabhängigen Zugriff zu ermöglichen, müssen folgende Leistungen erbracht werden:

   - Normalisierung der Positionsdaten durch Umrechnung in das Graphenmodell wie in Abschnitt 3.1.4 beschrieben.
     Da die zugrundeliegenden Daten fehlerhaft sein können, muß gleichzeitig eine Plausibilitätskontrolle auf Basis der letzten bekannten Position und der seitdem zurückgelegten Strecke durchgeführt werden.
   - Normalisierung der Fahrdaten (Fahrtrichtung und Geschwindigkeit).
   - Bereitstellung von Steuerungsfunktionen auf Basis der normalisierten Daten, die gleichzeitig die Einhaltung der hardwaretechnischen Randbedingungen gewährleisten.
     Damit kann ein Zug unabhängig vom Typ des eingesetzten Triebfahrzeugs gesteuert werden.

   Auf Basis dieser Leistungen wird eine Schnittstelle bereitgestellt, die eine Steuerung einzelner Züge hardwareunabhängig und auf hoher Ebene gestattet.

4. Steuerungsebene
   Während die unteren Schichten die Ansteuerung einzelner Züge sichern, liegt die Aufgabe dieser Schicht in der Organisation der Interaktion mehrerer Züge untereinander. Dies umfaßt entsprechend der Aufgabenstellung Mechanismen zur Kollisionserkennung und -vermeidung, wie sie im nachfolgenden Kapitel vorgestellt werden.

5. Benutzungsschnittstelle
   Auf das beschriebene System wird eine Bentzungsoberfläche aufgesetzt. Dadurch sollen externe Vorgaben ermöglicht werden.

   Aufgabe dieser Schicht ist zum einen eine Kontrolle der Benutzereingaben, zum anderen eine verständliche Darstellung der Vorgänge im System. Dazu gehören Mitteilungen über Position der Züge, erkannte Kollisionen und erfolgte Maßnahmen. Diese Oberfläche stellt eine Erweiterung dar und ist nicht mehr Teil des Modells. Daher soll auf eine eingehende Erörterung in dieser Arbeit verzichtet werden.

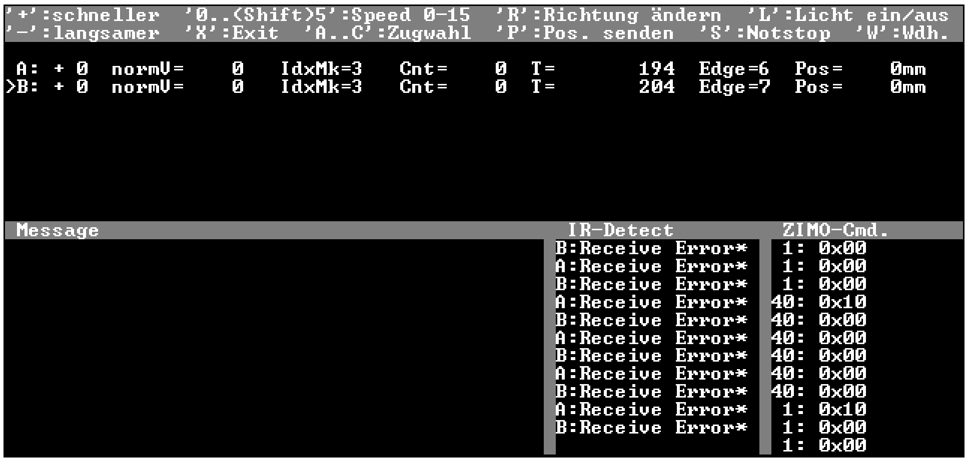

Abbildung 3.9: Benutzungsoberfläche

Kapitel 4

Kollisionen

Im vorangegangenen Kapitel wurden die Konzepte bei der Modellbildung vorgestellt. In diesem Kapitel sollen nun, aufsetzend auf den beschriebenen Schnittstellen, konkrete Algorithmen zur Erkennung und zur Klassifikation von Kollisionen besprochen werden. Damit werden die Voraussetzungen für schnelle und effiziente Gegenmaßnahmen geschaffen.

Um den Straßenbahnbetrieb realitätsnah nachzubilden, sollen entsprechend dem Modell 'Fahrt auf Sicht' nur Kollisionen erkannt und vermieden werden, wenn der Zugführer auf dem betreffenden Fahrzeug das Hindernis wahrnehmen kann. Ereignisse außerhalb der Sichtweite können das Verhalten eines Fahrzeugs nur indirekt über weitere beteiligte Züge beeinflussen.

4.1 Kollisionserkennung

Entsprechend dem Modell wird jedem Zug eine Sichtweite zugeordnet, die prinzipiell mit der Streckenführung variieren kann[1]. Die in Fahrtrichtung voraus liegende Strecke wird markiert, indem die betreffenden Knoten und Teile von Kanten entlang der Fahrstrecke eingefärbt werden, bis ein Hindernis angetroffen wird oder die Sichtweite erreicht ist. Ein Hindernis liegt vor, wenn ein Streckenabschnitt entweder bereits von einem anderen Zug markiert oder aus anderen Gründen unpassierbar ist. Im Modell entspricht dieser Vorgang dem Ereignis 'Zugführer erkennt Hindernis'. Damit ist eine Kollisionsgefahr erkannt. Kann aber die gesamte Sichtweite markiert werden, so ist die vorausliegende Strecke frei. Das beschriebene Prinzip der Kollisionserkennung wird in Bild 4.1 veranschaulicht. Ein konkretes Verfahren soll im folgenden dargestellt werden. Ein entsprechendes Ablaufdiagramm findet sich in Abbildung 4.3.

Zunächst werden alle auf der Gleisanlage vorhandenen Markierungen des betreffenden Zuges gelöscht. Dann wird, beginnend bei der aktuellen Triebwagenposition, die vorausliegende Strecke sukzessive überprüft und markiert. Wird eine Verzweigung passiert, so wird der zu markierende Zweig anhand der aktuellen Weichenstellung festgestellt. So werden nur diejenigen Gleisabschnitte berücksichtigt, die aller Voraussicht nach auch tatsächlich überfahren werden. Das Verfahren bricht ab, sobald ein bereits von einem anderen Zug markierter

[1]Derzeit ist lediglich eine Abhängigkeit der Sichtweite von der Fahrtgeschwindigkeit, nicht jedoch von Besonderheiten der Strecke implementiert.

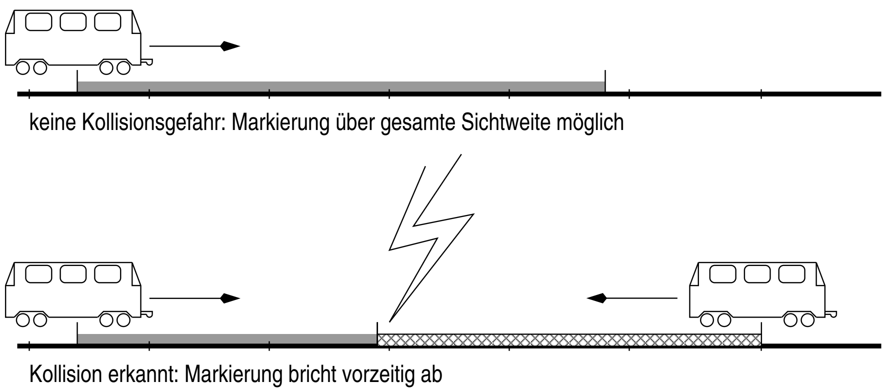

Abbildung 4.1: Kollisionserkennung durch Markierung

Streckenabschnitt angetroffen oder ein anderes Hindernis erkannt wird, spätestens jedoch sobald die volle Sichtweite markiert wurde. Diejenigen Teile der Kante, die außerhalb der Sichtweite liegen, werden nicht markiert.

Um eine Konsistenz der Daten zu gewährleisten, dürfen fremde Markierungen nicht überschrieben werden. Vielmehr wird jeder Zug vor einem neuen Durchlauf seine vorhandenen – veralteten – Markierungen löschen.

Eine Besonderheit sei am Rande kurz erwähnt: Die Position eines Triebfahrzeugs wird stets nur als Punkt angegeben[2]. Je nach Fahrtrichtung muß also noch die Länge des Zuges positiv oder negativ in Betracht gezogen werden. Vor Beginn des Vorgangs wird die eigentliche Sichtweite also noch um einen entsprechenden Betrag korrigiert. Diese Einzelheit ist aber für das weitere Vorgehen ohne Belang. Zur Vereinfachung wird daher im folgenden die Position eines Zuges punktförmig im arithmetischen Mittel seiner Länge angenommen.

Bei der Implementation wurde sichergestellt, daß Beginn und Ende einer markierten Strecke nie in einem Knoten, sondern stets auf einem Kantenstück liegen[3]. Damit wird, wie im folgenden Abschnitt deutlich werden wird, die Klassifikation bestimmter Kollisionstypen vereinfacht.

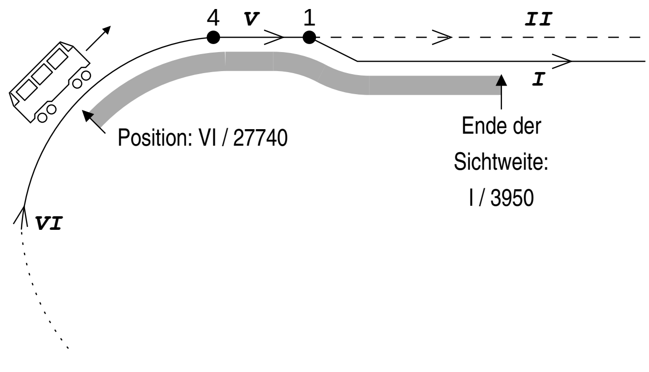

Abbildung 4.2: Markierung von Streckenabschnitten

[2]Dieser Punkt ist eigentlich die exakte Position der IR-Empfangsdiode, die die Signale der auf der Gleisanlage angebrachten Indexmarken aufnimmt (vgl. hierzu Abschnitt 2.2).

[3]Dies kann immer erreicht werden, da ein Knoten keine räumliche Ausdehnung besitzt.

In Bild 4.2 ist eine mögliche Situation für einen (willkürlich gewählten) Ausschnitt der bereits in den Abschnitten 2.4 und 3.1.3 angesprochenen Test-Gleisanlage (vgl. Bild 2.10 und 3.5) gezeigt. Dabei ergäbe sich etwa der folgende Ablauf:

1. Alte Markierungen auf der Gleisanlage werden entfernt. Die Position des Triebfahrzeugs wird mit 'Kante VI, $27740\frac{mm}{10}$' festgestellt. Die Sichtweite sei (willkürlich gewählt) 1000 mm. In den Gleisplanbeschreibungsdaten (vgl. Abbildung 3.6) findet sich die Kantenlänge der Kante VI mit $31700\frac{mm}{10}$ angegeben. Somit ergibt sich eine Restlänge auf der Kante von 396 mm.

2. Das verbleibende Teilstück der Kante VI wird komplett markiert. Die Restsichtweite wird entsprechend vermindert und beträgt nun
$$1000\ mm - 396\ mm = 604\ mm$$

3. Der anschließende Knoten Nr. 4 wird ebenfalls markiert.

4. Als folgende Kante wird Kante V mit einer Länge von $2090\frac{mm}{10}$ erkannt.

5. Kante V wird komplett markiert und die Restsichtweite um die Kantenlänge auf
$$604\ mm - 209\ mm = 395\ mm$$
vermindert.

6. Der anschließende Knoten Nr. 1 wird ebenfalls markiert.

7. Aus der aktuellen Weichenstellung wird Kante I als Folgekante erkannt. Die Kantenlänge beträgt $9520\frac{mm}{10}$.

8. Die Kantenlänge übersteigt die verbleibende Restsichtweite. Kante I wird also nur über die Länge der verbleibenden Restsichtweite bis zur Position $3950\frac{mm}{10}$ markiert. Da alle Markierungen erfolgreich waren, ist der einsehbare Streckenabschnitt als frei erkannt.

Kann eine Abschnittsmarkierung nicht abgeschlossen werden, so sind zwei Ursachen möglich:

- Bei der Markierung wurde ein unpassierbarer Streckenabschnitt oder ein Streckenendpunkt angetroffen.

- Bei der Markierung wurde ein bereits von einem anderen Zug markierter Bereiche angetroffen.

In beiden Fällen wird die Markierung abgebrochen und es ist eine Sonderbehandlung vonnöten. Im ersten Fall können Informationen über die Art der Sperrung (dauerhaft / vorübergehend) und Hinweise zu möglichen Aktionen im betreffenden Knoten-/Kantenobjekt erwartet werden[4]. Im zweiten Fall ist eine Kollisionsbehandlungsroutine aufzurufen, die das weitere Vorgehen bestimmt. Für diese Routine sind jedoch zunächst weitere Daten zu sammeln, so daß eine geeignete Behandlung ermöglicht wird.

[4] Diese Möglichkeit ist z.Zt. noch nicht implementiert

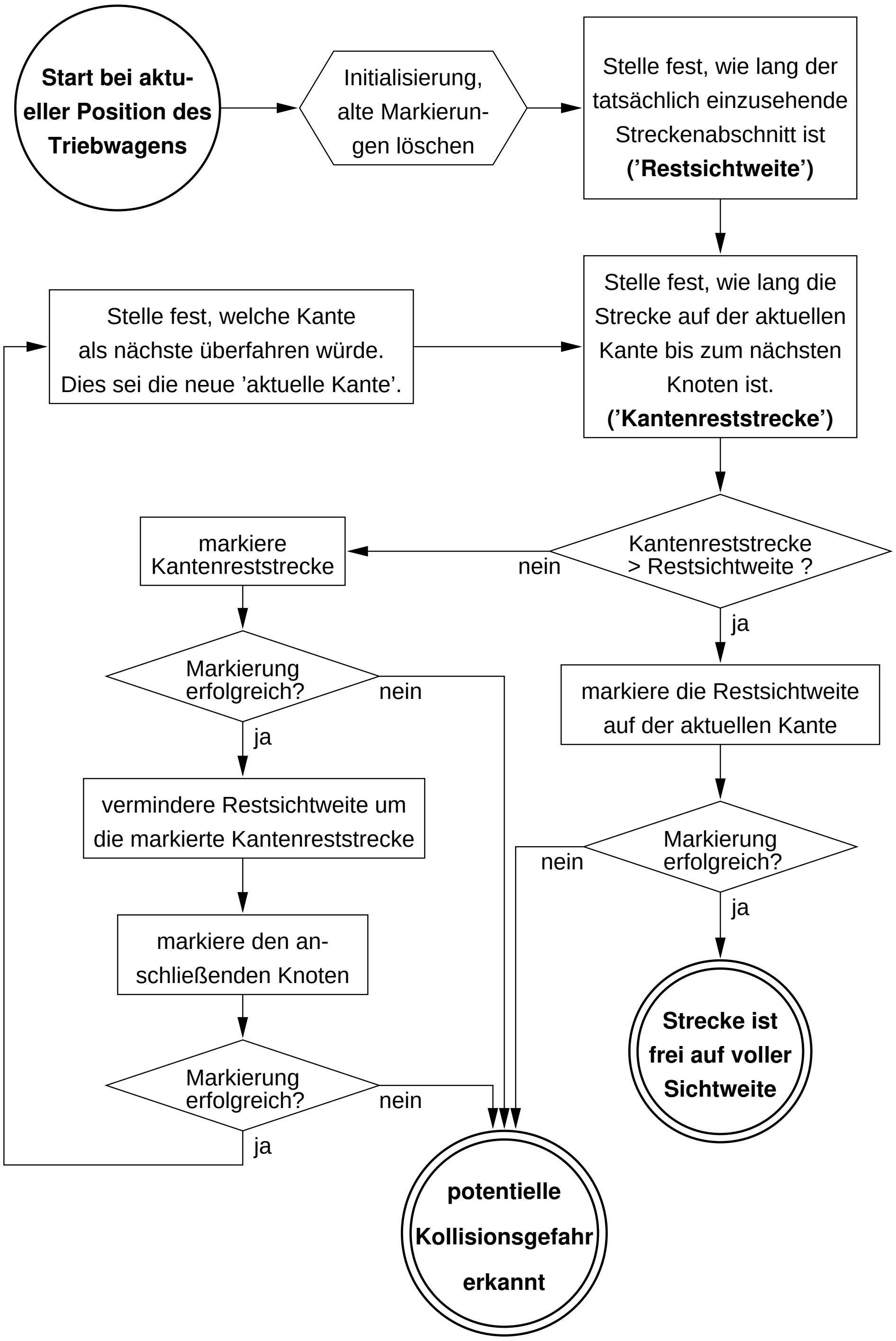

Abbildung 4.3: Ablaufdiagramm für ein Markierungsverfahren

4.2 Kollisionsklassifikation

Sobald eine potentielle Kollision erkannt wird, können gleichzeitig einige Aussagen über den Kollisionstyp getroffen werden, die helfen sollen, diese Kollision zu vermeiden. Damit lassen sich die möglichen Situationen in Kollisionsklassen gruppieren. Innerhalb der Klassen können dann – unabhängig vom konkreten Spezialfall – dieselben Maßnahmen zur Kollisionsvermeidung eingeleitet werden. Die folgenden Kriterien werden für die Klassifikation der Kollisionen herangezogen (vgl. Abb. 4.4):

1. Fahrtrichtung der beteiligten Züge
   Mögliche Alternativen:

   (a) Die Züge fahren in gleicher Richtung

   (b) Die Fahrtrichtung der Züge ist entgegengesetzt

   Zur Bewertung wird die gespeicherte Fahrtrichtung der Züge unabhängig von der tatsächlichen Fahrtgeschwindigkeit (die auch 0 betragen kann) herangezogen.

2. Ort der Kollisionserkennung
   Mögliche Alternativen:

   (a) Der Konflikt tritt in einem Knoten auf

   (b) Der Konflikt tritt auf einer Kante auf

   Nach dem oben gesagten kann ein Knoten nie Anfangs- oder Endpunkt einer markierten Strecke sein. Daher muß, wenn die Kollision in einem Knoten erkannt wird, die Fahrtstrecke des Kollisionsgegners über einen kreuzenden oder abzweigenden Gleisabschnitt führen[5]. Wird die Kollision dagegen auf einer Kante erkannt, so kann die gegnerische Fahrtstrecke direkt über die Position des markierenden Triebfahzeugs führen (vgl. Abb. 4.4, Fall 2). Im Vorgriff auf die Behandlungsroutinen besteht im ersten Fall die Möglichkeit, zu warten bis der Kollisionsgegner den betreffenden Knoten passiert hat[6]; im zweiten Fall wird die Kollision durch einfaches Warten nicht vermieden.

3. Direkte Ausweichmöglichkeiten der beiden Züge
   Mögliche Alternativen:

   (a) Zwischen beiden Zügen liegt mindestens eine Verzweigung

   (b) Zwischen beiden Zügen liegt keine Verzweigung

   Liegt zwischen dem markierenden Triebfahrzeug und dem Ort der Kollisionserkennung mindestens eine Verzweigung, so bestehen für die Kollisionsvermeidung noch weitere Alternativen. So kann z.B., wenn die Züge in entgegensetzter Richtung fahren, die Kollision auch mit Hilfe einer Weichenumstellung vermieden oder auf den unter 2 genannten Fall 'Kollisionserkennung im Knoten' zurückgeführt werden[7]. Zur Erkennung dieses

[5] Ein Markierungskonflikt dieser Art kann nur in Verzweigungsknoten auftreten

[6] vorausgesetzt, seine Geschwindigkeit ist größer als 0

[7] Da eine Weichensteuerung noch nicht implementiert ist, ist dieser Fall z.Zt. noch ohne praktische Bedeutung

Falles wird das Markierungsverfahren geringfügig erweitert, so daß, sobald ein Verzweigungsknoten markiert wird, ein entsprechendes Attribut gespeichert wird. Dabei ist die Richtung der Verzweigung zu beachten.

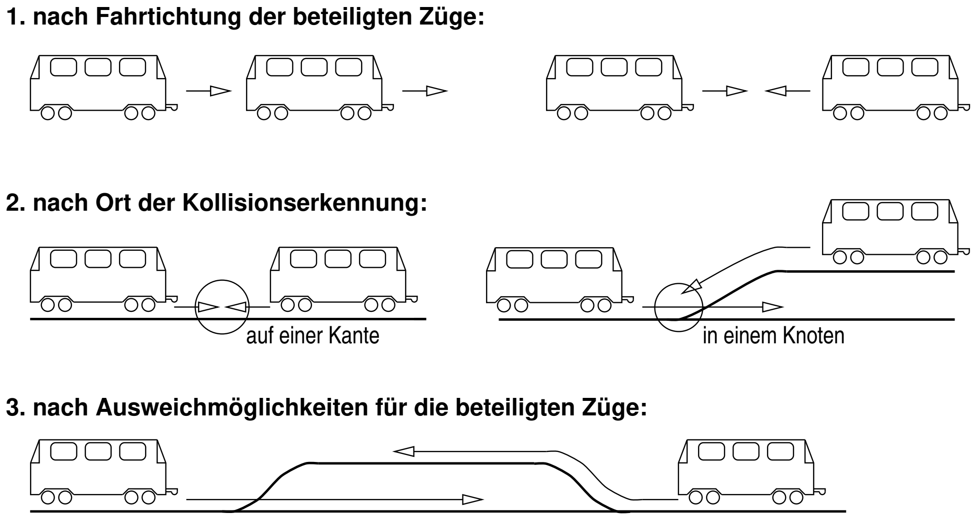

Abbildung 4.4: Kollisionsklassifikation

Die genannten Kriterien decken jedes für sich das gesamte Spektrum möglicher Kollisionen ab. Die genannten 3 Kriterien mit je 2 möglichen Alternativen führen so zu $2^3 = 8$ verschiedenen Kollisionsklassen, für die Kollisionsbehandlungsroutinen bereitgestellt werden müßten. In der Praxis wird jedoch nicht jede dieser Routinen benötigt, denn nicht jede mögliche Kombinationen der Kriterien ist sinnvoll: Wenn beispielsweise nach Kriterium 2 eine Kollision in einem Knoten festgestellt wurde, ist immer sichergestellt, daß nach Kriterium 3 mindestens eine Verzweigung zwischen markierendem Zug und dem Ort der Kollisionserkennung liegt – nämlich der Ort der Kollisionserkennung selbst.

Unter Berücksichtigung dieser Redundanzen werden im folgenden Kapitel zu den genannten Kollisionsklassen geeignete Behandlungsroutinen entworfen.

Aufgrund technischer Randbedingungen der zugrundeliegenden Hardware kann ein zusätzlicher Sonderfall auftreten: Es kann vorkommen, daß, z.B. durch Verschmutzung der Schienen, einige Züge zeitweise nicht auf Fahrbefehle reagieren und somit auch keine Positionsdaten aussenden. Dieser Fall wird vom Steuerungsrechner erkannt. Dabei können große Unsicherheiten über die tatsächliche Geschwindigkeit und damit auch über die tatsächliche Position des Zuges entstehen, so daß, wenn solche Züge in Kollisionsgefahr geraten, der gesamte Zugbetrieb auf der Anlage vermittels eines Nothalt-Signals vorübergehend angehalten werden muß. Nachdem der Benutzer das Problem manuell beseitigt hat, kann die Anlage wieder für den Zugbetrieb freigegeben werden.

Kapitel 5

Kollisionsvermeidung

Im vorangegangenen Kapitel wurden Kriterien entwickelt, nach denen Kollisionen eingeordnet werden können. Anhand dieser Kriterien können, wie beschrieben, verschiedene Klassen von Kollisionen entworfen werden. Zu den vorgestellten Kollisionsklassen sollen nun Algorithmen gefunden werden, die in der Lage sind, eine Kollision des betreffenden Typs sicher zu vermeiden. Dabei soll nicht etwa der insgesamt zu betreibende Aufwand minimiert werden, sondern die Entscheidungen sollen getreu der Aufgabenstellung 'Fahrt auf Sicht' auf Grundlage der dem jeweiligen Zugführer bekannten Daten getroffen werden. D.h. eine Interaktion erfolgt immer nur mit dem direkt benachbarten Kollisionsgegner; Ereignisse außerhalb der Sichtweite bleiben unberücksichtigt.

Möglicherweise ist ein Triebfahrzeug in seiner Reaktion von anderen (z.B. unmittelbar nachfolgenden) Zügen abhängig. Diese Abhängigkeiten werden das Verhalten des betroffenen Zugführers beeinflussen, soweit sie sich auf seine Sichtweite beziehen, und müssen daher auch bei der Kollisionsbehandlung in Betracht gezogen werden. In den meisten Fällen wird es mehrere mögliche, gleichermaßen geeignete Aktionen zur Kollisionsvermeidung geben, so daß diese Abhängigkeiten als zusätzliches Kriterium zur Auswahl der letztendlich eingesetzten Routine herangezogen werden.

5.1 Maßnahmen nach Kollisionserkennung

Nach Auftreten eines Markierungskonflikts können über die in Anhang A beschriebenen Schnittstellen zur Triebwagensteuerung weitere Maßnahmen eingeleitet werden, um eine drohende Kollision zu vermeiden. Die erforderlichen Aktionen werden im Rahmen einer Kollisionsbehandlung eingeleitet[1], deren schematischer Ablauf in Abbildung 5.1 dargestellt ist. Die einzelnen Schritte sollen im folgenden erläutert werden.

1. Eine Kollisionsbehandlung wird erforderlich, sobald die in Abschnitt 4.1 beschriebene Markierung eines Streckenabschnitts nicht komplett abgeschlossen werden konnte. Dabei wird gleichzeitig eine Klassifikation vorgenommen, so daß der Kollisionstyp bekannt ist.

[1] u.U. sind auch gar keine Maßnahmen erforderlich

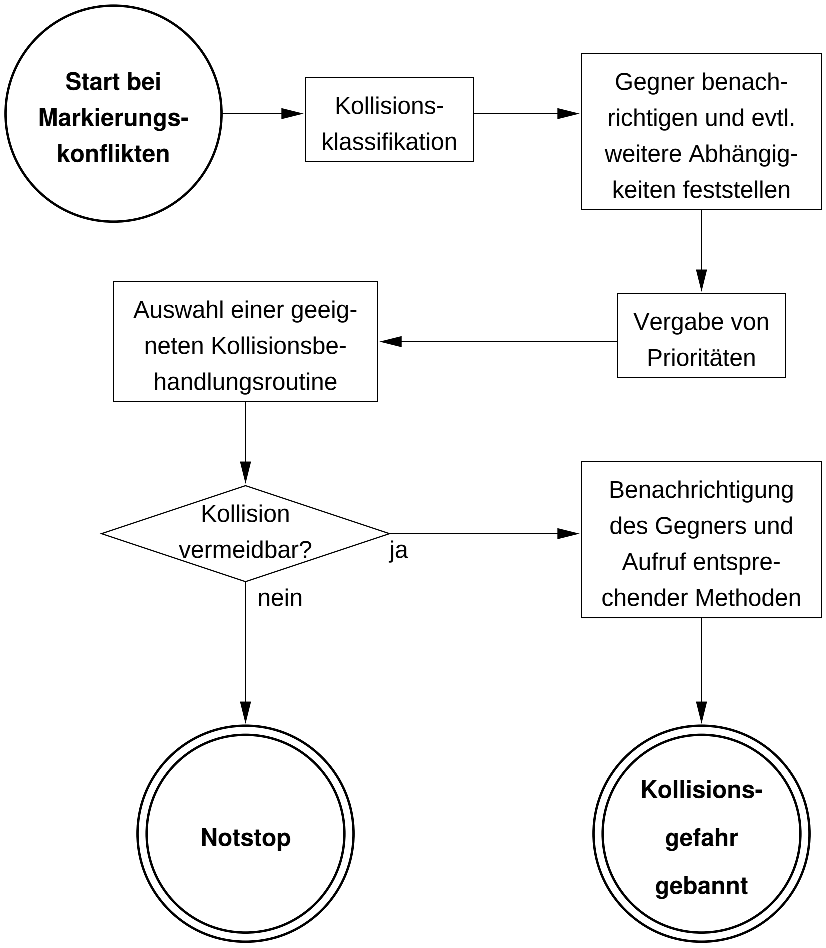

Abbildung 5.1: Ablaufdiagramm Kollisionsbehandlung

2. Die Feststellung der Kollisionsgefahr und des Kollisionstyps erfolgt zunächst von Seiten des markierenden Triebfahrzeuges aus. Der jeweilige Kollisionsgegner hat bis zu diesem Zeitpunkt keine Kenntnis davon und muß entsprechend benachrichtigt werden. Dabei ist es, wie bereits erwähnt, notwendig, eventuelle Abhängigkeiten des Gegners von weiteren Zügen festzustellen. So könnten etwa für ein Triebfahrzeug bestimmte Aktionen weitere Kollisionsbehandlungen zur Folge haben[2]. Diese Abhängigkeiten von anderen Triebfahrzeugen in Sichtweite werden zu jedem Zug als Attribut gespeichert und bei der Kollisionsbehandlung dem jeweiligen Gegner mitgeteilt.

3. Anhand der gefundenen Abhängigkeiten werden Prioritäten für die Züge festgesetzt, so daß eine Rangfolge entsteht, in der die Triebfahrzeuge auf die Kollisionsgefahr reagieren sollen.

[2]z.B. eine Richtungsänderung bei unmittelbar nachfolgende Triebfahrzeugen

4. Anhand des Kollisionstyps wird unter Berücksichtigung der Prioritäten eine geeignete Kollisionsbehandlungsroutine ausgewählt. Das dabei eingesetzte Verfahren wird im folgenden Abschnitt 5.2 detailliert beschrieben.

5. Da, wie in Abschnitt 2.4 erwähnt, einige Aktionen[3] nur beschränkt möglich sind, kann es passieren, daß eine Kollision mit den zu Gebote stehenden Mitteln nicht auf reguläre Weise vermieden werden kann. Dann kann keine geeignete Kollisionsbehandlungsroutine ausgeführt werden. Ist dies der Fall, so wird der gesamte Zugverkehr auf der Anlage aus Sicherheitsgründen umgehend angehalten.

6. Wurde hingegen eine geeignete Routine ausgewählt, so muß der Kollisionsgegner hiervon benachrichtigt werden. Dann werden die entsprechenden Fahrbefehle an die beteiligten Züge abgesetzt bzw. die entsprechenden Weichenstellungen vorgenommen.

7. Durch die Aktionen können sich neue Abhängigkeiten ergeben haben, die gespeichert werden müssen.

Im folgenden soll das Vorgehen bei der Auswahl geeigneter Kollisionsbehandlungsroutinen dargestellt werden. Dabei beschränken sich die Einflußmöglichkeiten auf Änderungen der Fahrtrichtung und -geschwindigkeit von Triebfahrzeugen sowie auf Umstellung von Weichen und Kreuzungsweichen[4].

5.2 Routinen zur Kollisionsvermeidung

Anhand der Klassifikation kann man alle Kollisionen auf wenige Generalfälle zurückführen, für die eine einheitliche Behandlung möglich ist. Für jede mögliche Kollisionsklasse werden Routinen bereitgestellt, die geeignet sind, eine Kollision des entsprechenden Typs zu vermeiden. Zur Zuordnung der in Abschnitt 4.2 eingeführten Kollisionsklassen zu den Behandlungsroutinen gibt die Tabelle in Abb. 5.2 Auskunft.

Klassifikationskriterien			geeignete Behand-lungsroutine
Fahrtrichtung	Ort der Kolli-sionserkennung	Ausweichmög-lichkeit	
gleich	Kante	ja	3
gleich	Kante	nein	1
gleich	Knoten	–	4
gegensätzlich	Kante	ja	5
gegensätzlich	Kante	nein	2
gegensätzlich	Knoten	–	6

Abbildung 5.2: Übersicht über Fälle der Kollisionsbehandlung

Es wird zu beobachten sein, daß es bei den meisten Kollisionsklassen mehrere geeignete (d.h. die Kollision vermeidende) Reaktionsmöglichkeiten gibt. Die Auswahl zwischen diesen Möglichkeiten wird anhand der an die Triebfahrzeuge

[3]z.B. Umstellungen von Weichen oder Richtungsänderungen über Knotenpunkten

[4]Wie bereits erwähnt konnte die Weichenstellung aufgrund von fehlender Hardware derzeit noch nicht implementiert werden.

vergebenen Prioritäten getroffen. Die Reaktionsmöglichkeiten werden im folgenden anhand der bereits bekannten Kollisionsklassen der Reihe nach aufgelistet.

1. Die Züge fahren in gleicher Fahrtrichtung. Der Konflikt wurde auf einer Kante erkannt, zwischen beiden Zügen liegt keine Verzweigung. D.h. beide fahren hintereinander her.

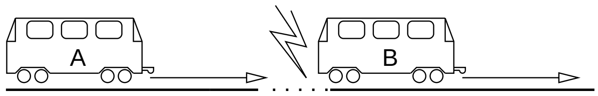

Abbildung 5.3: Kollisionsklasse 1

Die Geschwindigkeit des nachfolgenden Zuges (A) ist größer als die des vorausfahrenden (Zug B), der möglicherweise sogar steht. Die Kollisionsgefahr wird stets von Zug A erkannt.

Möglichkeiten zur Kollisionsvermeidung:

(a) A drosselt seine Geschwindigkeit mindestens bis auf das Niveau von B.

(b) B beschleunigt, bis er nicht mehr langsamer fährt als A.

(c) A verlangsamt in gleichem Maße wie B beschleunigt, bis B mindestens so schnell fährt wie A.

2. Die Fahrtrichtung der Züge ist entgegengesetzt, Ort der Kollisionserkennung ist auf einer Kante und zwischen beiden Zügen liegt keine Ausweichmöglichkeit. Die Züge fahren also unmittelbar aufeinander zu.

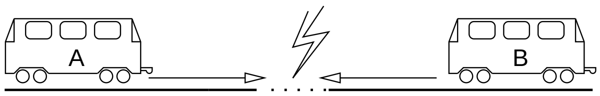

Abbildung 5.4: Kollisionsklasse 2

Über die Geschwindigkeit der beiden Züge kann nichts ausgesagt werden[5]. Die Kollisionsgefahr kann entweder von A oder von B erkannt werden.

Möglichkeiten zur Kollisionsvermeidung:

(a) A kehrt seine Fahrtrichtung um. Ist seine Geschwindigkeit geringer als die von B, so führt dies ggf. zu einer weiteren Kollisionsbehandlung nach Typ 1.

(b) B kehrt seine Fahrtrichtung um. Ist seine Geschwindigkeit geringer als die von A, so führt dies ggf. zu einer weiteren Kollisionsbehandlung nach Typ 1.

(c) A und B kehren beide ihre Fahrtrichtung um. Wenn keine weiteren Züge beteiligt sind, ist damit eine nachfolgende erneute Kollisionsbehandlung nicht erforderlich.

[5] Theoretisch ist es sogar möglich, daß beide Fahrzeuge bereits angehalten haben, aber ein erforderlicher Mindestabstand beim Ausrollen aufgrund der Massenträgheit unterschritten wurde.

3. Die Züge fahren in gleicher Fahrtrichtung hintereinander her, zwischen ihnen liegt (mindestens) eine Verzweigung. Die Kollisionsgefahr wurde auf einer Kante erkannt.

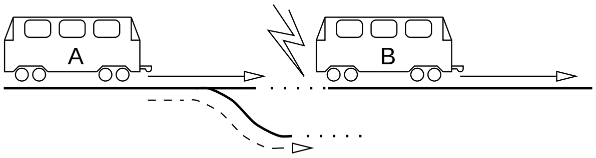

Abbildung 5.5: Kollisionsklasse 3

Die Geschwindigkeit des nachfolgenden Zuges (A) ist größer als die des vorausfahrenden (Zug B), der möglicherweise sogar steht. Die Kollisionsgefahr wird stets von Zug A erkannt.

Möglichkeiten zur Kollisionsvermeidung:

(a) Durch Umstellung der Weiche wird Zug A umgeleitet und die Kollisionsgefahr gebannt[6].

(b) Es erfolgt eine Kollisionsbehandlung wie unter 1.

4. Die Züge fahren in gleicher Fahrtrichtung, zwischen ihnen liegt eine Verzweigung. Die Kollisionsgefahr wurde in einem Knoten erkannt.

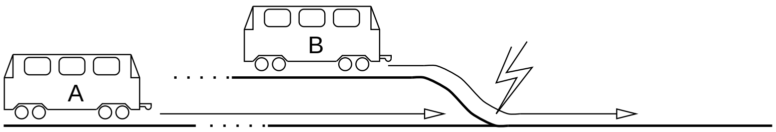

Abbildung 5.6: Kollisionsklasse 4

Tritt ein Konflikt in einem Knoten auf, so muß es sich bei dem Knoten um eine Kreuzung oder Verzweigung handeln, da ein Knoten nie Beginn oder Ende eines markierten Streckenabschnitts sein kann (vgl. Abschnitt 4.1). Über die Geschwindigkeit der beiden Züge kann nichts ausgesagt werden. Die Kollisionsgefahr wird stets von Zug A erkannt.

Möglichkeiten zur Kollisionsvermeidung:

(a) B beschleunigt soweit wie möglich, während A anhält und abwartet, bis B die Kreuzung oder Verzweigung passiert hat. Handelt es sich bei dem Knoten um eine einfache Weiche, so führt dies u.U. zu einer weiteren Kollisionsbehandlung nach Typ 1.

(b) A beschleunigt soweit wie möglich, während B anhält und abwartet, bis A die Kreuzung oder Verzweigung passiert hat. Handelt es sich bei dem Knoten um eine einfache Weiche, so führt dies u.U. zu einer weiteren Kollisionsbehandlung nach Typ 1.

(c) A kehrt seine Fahrtrichtung um.

[6] Da Weichenansteuerungen z.Zt. noch nicht möglich sind, ist diese Alternative nur theoretischer Art.

(d) B kehrt seine Fahrtrichtung um.

5. Die Züge fahren in entgegengesetzter Fahrtrichtung aufeinander zu, zwischen ihnen liegt eine Verzweigung. Die Kollisionsgefahr wurde auf einer Kante erkannt.

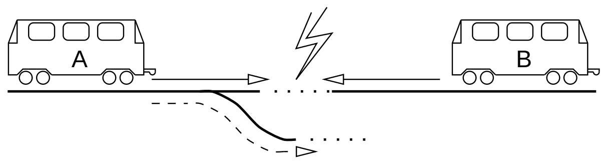

Abbildung 5.7: Kollisionsklasse 5

Über die Geschwindigkeit der beiden Züge kann nichts ausgesagt werden. Die Kollisionsgefahr kann entweder von A oder von B erkannt werden.

Möglichkeiten zur Kollisionsvermeidung:

(a) Durch Umstellung der Weiche wird Zug A umgeleitet und es erfolgt ggf. eine Kollisionsbehandlung nach dem Schema von Typ 6[7].

(b) Es erfolgt eine Kollisionsbehandlung wie unter 2.

6. Die Züge fahren in entgegengesetzter Fahrtrichtung aufeinander zu, die Kollisionsgefahr wurde in einem Knoten erkannt.

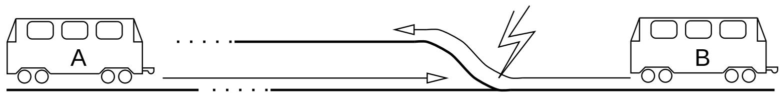

Abbildung 5.8: Kollisionsklasse 6

Tritt ein Konflikt in einem Knoten auf, so muß es sich bei dem Knoten um eine Kreuzung oder Verzweigung handeln, da ein Knoten nie Beginn oder Ende eines markierten Streckenabschnitts sein kann (vgl. Abschnitt 4.1). Über die Geschwindigkeit der beiden Züge kann nichts ausgesagt werden. Die Kollisionsgefahr wird stets von Zug A erkannt.

Möglichkeiten zur Kollisionsvermeidung:

(a) B beschleunigt soweit wie möglich, während A anhält und abwartet, bis B die Kreuzung oder Verzweigung passiert hat. Wenn keine weiteren Züge folgen, wird damit die Strecke für A frei[8].

(b) Es erfolgt eine Kollisionsbehandlung wie unter Punkt 2.

7. In Abschnitt 4.1 wurde der Sonderfall angesprochen, daß ein Zug, der vorübergehend nicht auf Fahrbefehle reagiert, in Kollisionsgefahr gerät.

[7] Da Weichenansteuerungen z.Zt. noch nicht möglich sind, ist diese Alternative nur theoretischer Art.

[8] Im Falle einer Kreuzung könnte u.U. auch A durchfahren und B abwarten. Dieser Fall wird hier jedoch vernachlässigt.

Da das Verhalten eines solchen Zuges nicht vorhersagbar ist, kann eine reguläre Kollisionsbehandlung nicht durchgeführt werden. Stattdessen wird über einen 'Notstop'-Befehl[9] der Zugverkehr auf der gesamten Anlage angehalten. Diese Maßnahme wird ebenfalls ergriffen, wenn aufgrund von technischen Randbedingungen keine Routine zur Kollisionsvermeidung ausführbar ist[10].

5.3 Mehrzugsteuerung

Bisher wurde die Kollisionsvermeidung für einzelne Züge besprochen. Nun soll abschließend das Vorgehen auf einer Anlage im Mehrzugbetrieb übertragen werden. Das Verfahren zur Kollisionserkennung und -vermeidung ändert sich nicht mit der Anzahl der eingesetzten Züge. Es muß lediglich sichergestellt werden, daß für jeden Zug in einem vernünftigen Zeitraum[11] mindestens einmal Gelegenheit dazu besteht, seine aktuelle Position zu übertragen, die Markierung der einsehbaren Fahrstrecke durchzuführen und ggf. eine Kollisionsbehandlung durchzuführen. Da das vorliegende Programm lediglich auf Einzelplatz-PCs unter MS-DOS betrieben wird, sind keine weiteren Vorkehrungen zur Sicherung der Echtzeit-Anforderungen nötig. Es reicht aus, die auf der Anlage befindlichen Züge in einer Endlosschleife zyklisch zu durchlaufen und die genannten Aktionen sukzessive für jeden Zug durchzuführen. Der schematische Ablauf ist in Bild 5.9 dargestellt.

Somit kann jede Kollision prinzipiell vermieden werden. Allerdings ist nicht gewährleistet, daß durch die Änderungen der Fahrtdaten in einer Kollisionsbehandlung weitere Kollisionsbehandlungen für einen Zug erforderlich werden. Prinzipiell kann nicht einmal ein (lokaler) Deadlock ausgeschlossen werden. Diese scheinbar ungünstigen Effekte entsprechen jedoch genau dem Modell 'Fahrt auf Sicht', denn auch dort können solche Auswirkungen auftreten.

Der Benutzer hat stets die Möglichkeit, eigene Steuerbefehle abzusetzen und kann so derartige Situationen unter Umgehung der genannten Beschränkungen 'manuell' auflösen. Selbstverständlich können auch kritische Situationen zu Demonstrationszwecken provoziert werden. Die Darstellung der Benutzungsoberfläche mit ihren Sicherungsmechanismen würde jedoch den Rahmen dieser Arbeit sprengen und soll daher mit Verweis auf die in Anhang A gemachten Bemerkungen unterbleiben.

[9]Intern wird dieser Befehl in einen ZIMO-'Sammelstop'-Befehl umgesetzt; denkbar wäre auch eine vollständige Stromunterbrechung für die gesamte Anlage.

[10]Das kann etwa der Fall sein, wenn eine Umkehr der Fahrtrichtung eines Zuges zwingend erforderlich ist, aber nicht ausgeführt werden kann, weil sich der Zug im kritischen Bereich eines Knotens befindet (z.B. wg. Entgleisungsgefahr beim 'Aufschneiden' von Weichen).

[11]Dieser Zeitraum ist stark von der Anzahl der eingesetzten Züge, deren Fahrtgeschwindigkeit und der Sichtweite abhängig

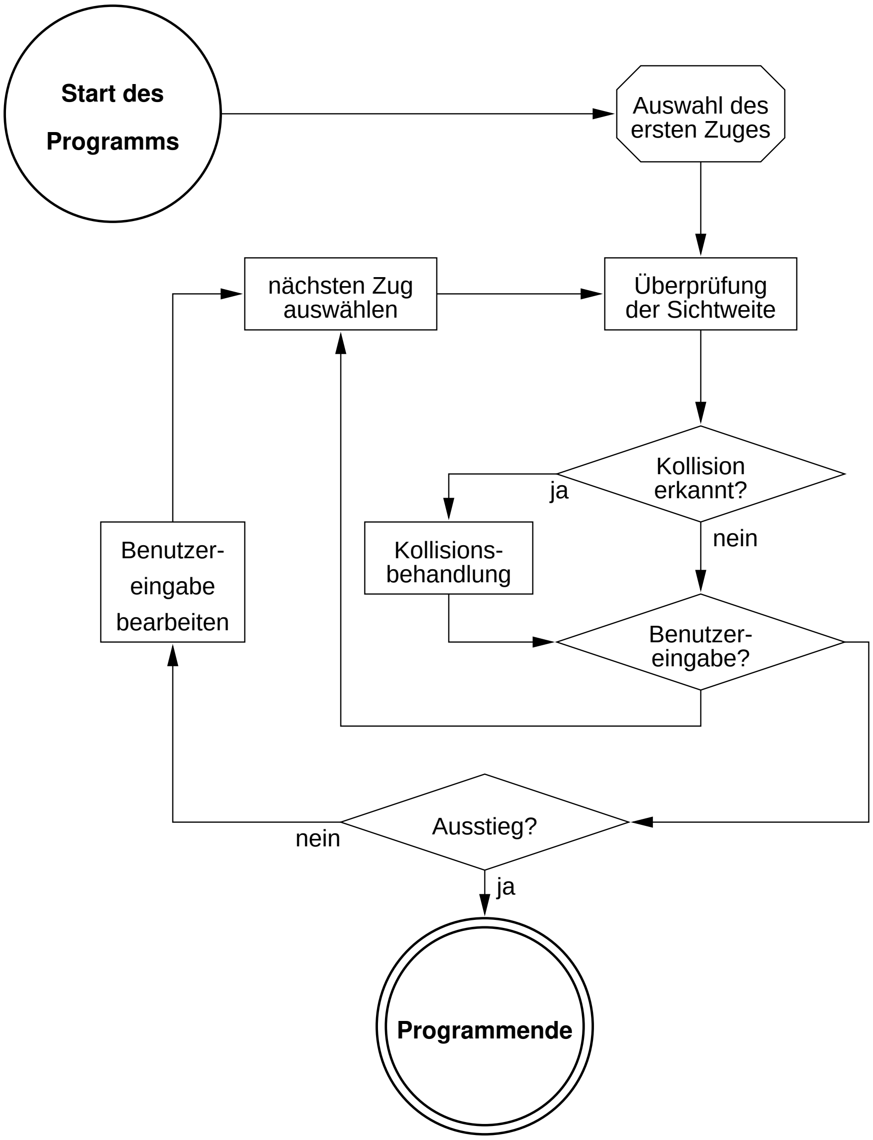

Abbildung 5.9: Ablauf der Mehrzugsteuerung

Kapitel 6

Schlußbetrachtungen

In diesem Kapitel werden die erreichten Ergebnisse mit den Zielvorgaben aus Abschnitt 1.2 verglichen. Gleichzeitig sollen etwaige Unzulänglichkeiten und noch zu implementierende Erweiterungen aufgezeigt werden.

6.1 Auswertung der Ergebnisse

Es soll untersucht werden, inwieweit der Anspruch dieser Arbeit, einen Straßenbahnbetrieb unter Berücksichtigung des Konzeptes 'Fahrt auf Sicht' zu modellieren, umgesetzt werden konnte.

Mit den vorgestellten Methoden ist es ohne weiteres möglich, dem PC die Aufgabe des Zugführers zu übertragen. Die ergriffenen Maßnahmen liegen im Gegensatz zu einem menschlichen Zugführer stets fest; eine Variation des Fahrstils findet nicht statt. Allerdings sind Erweiterungsmöglichkeiten denkbar, so daß sich benutzerseitig unterschiedliche Verhaltensmuster der Zugführer vorgeben lassen, die auch die Auswahl der Kollisionsbehandlungsroutine beeinflussen könnten.

Einschränkungen gegenüber der Realität resultieren im wesentlichen aus hardwaretechnischen Beschränkungen sowie nicht auszuschaltenden Störeinflüssen auf die Datenübertragung[1]. Bedingt durch den beschränkten Platz auf einem Modellbahntriebwagen konnten hier keine Algorithmen zur Fehlererkennung und -korrektur in ausreichendem Maße hardwareseitig unterstützt werden. Die daraus entstehende Notwendigkeit einer softwaremäßigen Plausibilitätskontrolle der empfangenen Positionsdaten durchbricht Schichtgrenzen des in Abschnitt 3.2 vorgestellten 5-Schichten-Modells. Davon abgesehen konnten jedoch die hardwaretechnischen Besonderheiten einzelner Triebfahrzeuge gekapselt werden, so daß eine Ansteuerung über eine einheitliche Schnittstelle und unabhängig von dem Typ des betreffenden Triebfahrzeugs möglich ist.

Die Voraussetzungen für eine benutzerseitig vorzugebende Streckenführung wurden geschaffen. Mit Hilfe des beschriebenen Gleisplanmodells lassen sich beliebige zusammenhängende Gleispläne beschreiben. Die Wahl einer geeigneten Beschreibungssprache bleibt dem Benutzer überlassen, soweit geeignete Konvertierungsroutinen zum Einlesen der Streckenführung bereitgestellt werden.

[1]Dies betrifft sowohl Fahrbefehle an die Triebfahrzeuge als auch Rückmeldungen von den Zügen.

Durch eine der Steuerungsebene übegeordnete Benutzeroberfläche wurden auch weitgehende Eingriffsmöglichkeiten des Benutzers realisiert. Die Benutzeroberfläche ist kurz in Anhang A erläutert; auf die weitergehenden Möglichkeiten kann in dieser Arbeit aus Platzgründen jedoch leider nicht eingegangen werden.

6.2 Ausblick

Zum gegenwärtigen Stand bietet das Projekt eine Reihe von Verbesserungs- und Erweiterungsmöglichkeiten. So ergibt sich aus der Einbeziehung von Kreuzungen und Verzweigungen in den Streckenverlauf gleichzeitig auch die unmittelbare Forderung nach einer computergesteuerten Weichenstellung. Bisher scheiterte dies an fehlender Hardware. Softwareseitig wurde diese Option jedoch weitgehend unterstützt, so daß eine entsprechende Erweiterung nur geringe Probleme verursachen dürfte. Damit einhergehend bietet sich die Entwicklung einer eigenen Gleiplanbeschreibungssprache an, die dem Benutzer eine komfortable Möglichkeit zur Beschreibung der Streckenführung bietet. Zur Unterstützung dieser Option wurde beim Software-Entwurf die Möglichkeit zur Einbindung spezieller Routinen zum Einlesen der Streckenführung berücksichtigt.

In der vorliegenden Arbeit wurde mehrfach darauf hingewiesen, daß bei der Übertragung der Positionsdaten vom Triebfahrzeug aus Platzgründen auf die hardwaretechnische Unterstützung einer Fehlererkennung und -korrektur verzichtet werden mußte. Neben dem Ausbau der bestehenden softwaretechnischen Sicherungsmechanismen wird sicherlich die Aufhebung dieser Beschränkung, etwa durch Einsatz platzsparender Logikbausteine, eine vordringliche Aufgabe sein.

Eine signifikante Leistungssteigerung ließe sich mit einer Erhöhung der Datenrate des Zimo-Systems erreichen. Durch die Beschränkung auf 40 Steuerbefehle pro Sekunde (vgl. Anhang B) liegt hier der entscheidende Engpaß vor.

Abgesehen von derartigen Verbesserungen bestehender Strukturen bieten sich natürlich auch eine Reihe von Neuerungen an, um den simulierten Betrieb noch realitätsnäher zu gestalten.

So ist beispielsweise eine Kombination von Signal-Blocksteuerung und 'Fahrt auf Sicht' möglich, wie sie auch im praktischen Betrieb häufig anzutreffen ist (vgl. [10]). Der Standort eines Signals wäre dann als neuer Knotentyp mit entsprechenden Attributen für Freigabe, Sperrung oder eingeschränkte Befahrbarkeit des folgenden Streckenabschnitts zu definieren.

Weiterhin liegt eine dynamische Anpassung der Sichtweite nicht nur an die Fahrtgeschwindigkeit, sondern auch an herrschende Streckenverhältnisse[2] nahe. Dies ließe sich über Auswertung entsprechender Kantenattribute bei der Sichtweitenbestimmung im Markierungsprozeß (vgl. Abschnitt 4.1) erreichen. Im Zusammenhang mit der bereits angesprochenen Entwicklung einer eigenen Gleisplanbeschreibungssprache bietet sich die Möglichkeit, diese und weitere Streckenattribute vom Benutzer vorgeben zu lassen. In Erweiterung dieses Konzeptes wäre sogar, dem Vorbild entsprechend, die Erstellung eigener Fahrpläne mit Vorgaben für Fahrtstrecke sowie Ankunfts- und Abfahrtzeiten denkbar.

Aufgrund des objektorientierten Modellansatzes bietet sich noch eine Fülle weiterer, nahezu beliebiger Erweiterungsmöglichkeiten, die in dieser Arbeit nicht alle aufgeführt werden können.

[2] z.B. kurvige oder bergige Strecke

Anhang A

Prototypen der verwendeten Klassen

A.1 Gleisplanbeschreibung

A.1.1 Repräsentation von Streckenabschnitten

Die Klasse TEdge (Datei ROUTE.H)

```
// *******************************************************************
// * Class TEdge                                                     *
// *-----------------------------------------------------------------*
// * Repraesentiert Kanten, d.h. endliche Strecken                   *
// *******************************************************************
   class TEdge {
         unsigned nLength; // Laenge der Kante
         unsigned nNumber; // Nummer der Kante
         TEdgeConnector *pClkConnector, *pAntiClkConnector;
         BI_IListImp<TMark> MarkList;
         typedef BI_IListIteratorImp<TMark> MarkListIteratorType;
//         MarkListIteratorType *pMarkListIterator;
     public:
   // -------------------------------------------------------------------
   // zur Listenbearbeitung noetige Operatoren:
         int operator==(const TEdge& Edge) const {
             return (Edge.GetNumber()==nNumber);
         }
   // -------------------------------------------------------------------
         int operator==(const int Num) const {
             return (Num==nNumber);
         }
   // -------------------------------------------------------------------
         TEdge( unsigned nEdgeLen, unsigned nEdgeNum,
                TNode* pNode1, tDIRECTION SideAtNode1,
                TNode* pNode2, tDIRECTION SideAtNode2 );
                        // Begrenzungsknoten ( Node1 in Zaehlrichtung,
                        //                     Node2 entgegen Zaehlrichtung)
                        // SideAtNode gibt an, auf welcher Seite des Knotens
                        // die Kante eingehaengt wird
         virtual ~TEdge();
         virtual unsigned GetNumber() const ;
         virtual unsigned GetLength() const ;
```

```
        virtual TEdgeConnector* GetConnector(tDIRECTION Side) const ;
        virtual tBool Unmark(TFullTrain* pTrain);
        virtual void Look( TCollClass *pCollision,
                           unsigned nStartPos,
                           tDIRECTION Dir        );
        virtual tBool Mark( TMark *pMark,
                            unsigned nStartPos,
                            tDIRECTION Dir    );
  };
```

Die Klasse TEdge repräsentiert die einzelnen Kantenstücke einer Gleisanlage. Diese haben alle dieselbe Struktur: Sie besitzen keine Verzweigungen, haben eine Länge größer als Null und genau zwei Enden. An jedem Ende sind sie mit genau einem Knoten verbunden[1]. D.h. es befindet sich genau ein Knoten in Zählrichtung und genau einer in entgegengesetzter Zählrichtung[2]. Damit liegt bei Angabe der Fahrtrichtung der zunächst erreichte Knoten fest. Die Verbindung mit einem Knoten wird über einen Zeiger auf ein entsprechendes Verbindungsstück vom Typ TEdgeConnector realisiert.

Jede Kante wird durch eine Kantennummer eineindeutig referenzierbar.

Kanten können (prinzipiell) beliebig viele Markierungen erhalten und beliebig viele Loks aufnehmen, solange die Markierung dies zuläßt.

Die Klasse TNode (Datei ROUTE.H)

```
// ********************************************************************
// * Class TNode                                                      *
// *------------------------------------------------------------------*
// * Repraesentiert Streckenknoten (Knotenpunkte ohne Ausdehnung)      *
// * Basisklasse, abstrakt machen ?                                   *
// ********************************************************************
   class TNode {
        unsigned nNodeNumber;
        TIndex   nNodeIndex;
        TMark    *pLockMark;
        BI_IListImp<TNodeConnector> ClkConnList, AntiClkConnList;
     public:
        int operator==(const TNode& Node) const;
        int operator==(const int Num) const;
        TNode( unsigned nNewNodeNumber, TIndex nNewNodeIndex=-1 );
        virtual ~TNode();
        virtual unsigned GetNumber() const ;
        virtual TIndex GetIndex() const ;
        virtual tBool LinkEdge( TEdgeConnector *pEdgeCon,
                                 const tDIRECTION Side);
        virtual tBool ConnectEdge( const unsigned nFromEdgeNumber,
                                   const unsigned nToEdgeNumber,
                                   const tDIRECTION Dir           );
        virtual tBool Mark(TMark *pMark);
        virtual tBool Unmark(TFullTrain* pTrain);
        virtual TFullTrain* GetMarkingTrain() const ;
   };
```

[1]Ein offenes Ende, z.B. bei einem Prellbock, wird immer durch einen Knoten abgeschlossen

[2]Auch im Falle einer Kehrschleife sind beide Knoten verschieden, da das erforderliche Zimo-Kehrschleifenmodul als Knoten definiert werden muß.

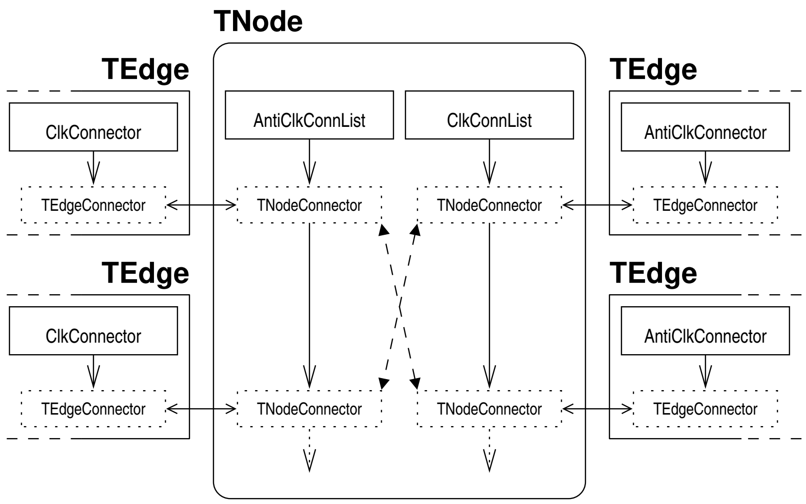

Abbildung A.1: Verbindungsstruktur eines Kreuzungsknotens

Die Klasse TNode repräsentiert die einzelnen Knoten einer Gleisanlage. Diese haben alle dieselbe Struktur: Sie besitzen keine Ausdehnung und beliebig viele, aber mindestens zwei angrenzende Kanten[3]. Sie können markiert (gefärbt) werden, wenn sie in die Sichtweite einer Lok geraten oder die Lok sie überfährt. Dann darf keine andere Lok einen kritischen Mindestabstand vom betreffenden Knoten unterschreiten.

Die Verbindung mit den Kanten ist für beide Zählrichtungen durch je eine Liste von Zeigern auf entsprechende Verbindungsstücke vom Typ TNodeConnector realisiert, wie in Abbildung A.1 deutlich wird.

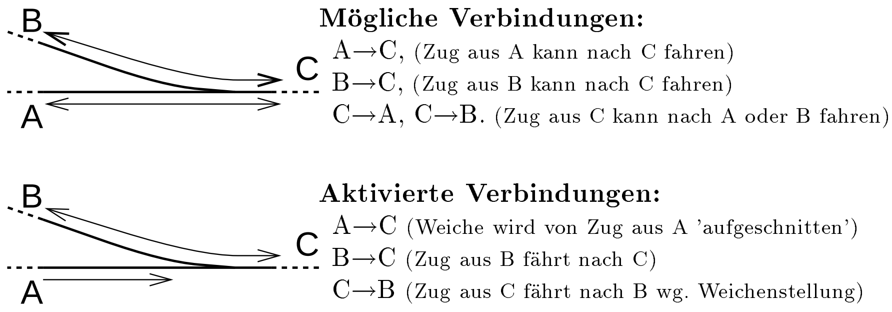

Abbildung A.2: Beispiel für einen Verzweigungsknoten

Entsprechend der Fahrtrichtung und der Streckenführung existieren zu jeder Ausgangskante eine oder mehrere mögliche Verbindungen zu Zielkanten, von denen gemäß der Weichenstellung stets genau eine aktiviert ist. So kann jeder Ausgangskante zu jedem Zeitpunkt genau eine Zielkante zugeordnet werden.

[3] Im Sonderfall, daß der Knoten einen Streckenendpunkt repräsentiert, besitzt er nur eine angrenzende Kante. Dies erfordert eine Sonderbehandlung.

Das Schema ist in Abbildung A.2 dargestellt.

A.1.2 Verbindung von Streckenabschnitten

Die Klasse TConnector (Datei ROUTE.H)

```
// *******************************************************************
// * Class TConnector (Abstrakte Basisklasse)                        *
// *-----------------------------------------------------------------*
// * Repraesentiert allgemeine Verbindungen zwischen Knoten und Kanten *
// * template <class TLocalConnector, class TRemoteConnector>        *
// *  TLocalConnector bezeichnet den Konnektor im gleichen Objekt auf *
// *  der anderen Seite                                               *
// *  TRemoteConnector bezeichnet den Konnektor im angebundenen Objekt *
// *******************************************************************
   template <class TLocalConnector, class TRemoteConnector>
   class TConnector {
         TLocalConnector *pPartnerConnector; // (Verbindung variabel)
         TRemoteConnector *pLinkedConnector; // (Verbindung fest)
     public:
         TConnector();
         virtual ~TConnector();
         virtual tBool LinkTo(TRemoteConnector *pLink);
                 // feste Verbindung entsprechend dem Gleisaufbau
         virtual tBool ConnectWith(TLocalConnector *pPartner);
                 // dynamische Verbindung entsprechend der Weichenstellung
         virtual TLocalConnector* GetPartner() const;
         virtual TRemoteConnector* GetLink() const;
         virtual void Look( TCollClass *pCollision,
                            unsigned nStartPos    ) = 0;
                                        // feststellen, ob Strecke frei ist
         virtual tBool Mark( TMark *pMark,
                             unsigned nStartPos ) = 0;
         virtual tBool Unmark( TFullTrain *pTrain ) = 0;
                                        // Node:Unlock; Edge: Entfaerben
   };
```

Die Klasse TConnector ist eine abstrakte Basisklasse, mit der elementare Methoden und Datenelemente zur Verbindung von Knoten und Kanten bereitgestellt werden. Wie bereits in Abbildung A.1 deutlich wurde, muß durch einen Konnektor für jede der beiden möglichen Fahrtrichtungen eine eindeutige Verbindung hergestellt werden: Eine zum angrenzenden Konnektor des jeweiligen Nachbarobjekts anderen Typs (repräsentiert durch TRemoteConnector), und eine in innerhalb des betreffenden Objekts zum gegensinnigen Konnektor gleichen Typs (TLocalConnector).

Alle eingesetzten Konnektoren für Knoten und Kanten werden von dieser Basisklasse abgeleitet. Somit können Zeiger auf beliebige Konnektoren vom Typ eines Zeigers auf TConnector definiert werden.

Die Klasse TNodeConnector (Datei ROUTE.H)

```
// *******************************************************************
// * Class TNodeConnector                                            *
// *-----------------------------------------------------------------*
// * Repraesentiert Verbindungspunkte an Knoten                      *
// *******************************************************************
```

```
class TNodeConnector : public TConnector<TNodeConnector,TEdgeConnector> {
      TNode* pWrappingNode;
  public:
// --------------------------------------------------------------------
// zur Listenbearbeitung noetige Operatoren:
      int operator==(const TNodeConnector& NodeConn) const;
      int operator==(const int Num) const;
// --------------------------------------------------------------------
      TNodeConnector(TNode* pWrpNode);
      virtual ~TNodeConnector();
      virtual void Look( TCollClass *pCollision,
                         unsigned nStartPos    );
                                     // feststellen, ob Strecke frei ist
      virtual tBool Mark( TMark *pMark,
                          unsigned nStartPos );
      virtual tBool Unmark( TFullTrain *pTrain );
                                     // Node:Unlock; Edge: Entfaerben
      virtual TNode* GetWrpNode() const;
};
```

Die Klasse TNodeConnector ist eine von TConnector abgeleitete Klasse, mit der von einem Knoten ausgehende Verbindungen realisiert werden. Der zugehörige Knoten ist dem Konnektor bekannt, so daß dort zugängliche Informationen, beispielsweise über die aktuelle Weichenstellung, abgerufen werden können.

Die Klasse TEdgeConnector (Datei ROUTE.H)

```
// ********************************************************************
// * Class TEdgeConnector                                             *
// *------------------------------------------------------------------*
// * Repraesentiert Verbindungspunkte an Kanten                       *
// ********************************************************************
   class TEdgeConnector : public TConnector<TEdgeConnector,TNodeConnector> {
         TEdge* pWrappingEdge;
     public:
         TEdgeConnector(TEdge* pWrpEdge);
         virtual ~TEdgeConnector();
         virtual unsigned GetNumber(); // liefert Nummer der zugehoerigen Kante
         virtual tDIRECTION GetSide(); // auf welcher Seite ist dieser Anschluss?
         virtual void Look( TCollClass *pCollision,
                            unsigned nStartPos    );
                                         // feststellen, ob Strecke frei ist
         virtual tBool Mark( TMark *pMark,
                             unsigned nStartPos );
         virtual tBool Unmark( TFullTrain *pTrain ); // Entfaerben der
                                                     // betreffenden Kante(n)
         virtual TEdge* GetWrpEdge() const;
   };
```

Die Klasse TEdgeConnector ist eine von TConnector abgeleitete Klasse, mit der von einer Kante ausgehende Verbindungen realisiert werden. Die zugehörige Kante ist dem Konnektor bekannt, so daß dort zugängliche Informationen, beispielsweise über die vorübergehend gesperrte Streckenabschnitte auf der Kante[4], abgerufen werden können.

[4] Dieses Feature ist z.Zt. noch nicht implementiert.

A.1.3 Repräsentation der Gesamtstrecke

Die Klasse TRoute (Datei ROUTE.H)

```
// *****************************************************************
// * Class TRoute                                                  *
// *---------------------------------------------------------------*
// * Repraesentiert das gesamte Streckenmodell                     *
// *****************************************************************
   class TRoute {
         BI_IListImp<TNode> NodeList;
         BI_IListImp<TEdge> EdgeList;
     public:
         TRoute();
         ~TRoute();
         TEdge* FindEdge(unsigned nEdgeNum);
   };
```

Die Klasse TRoute repräsentiert die aus Knoten und Kanten zusammengesetzte Gesamtstrecke. Hauptaufgabe dieser Klasse ist es, anhand der Benutzervorgaben den Aufbau des Gleisplangraphen zu bewerkstelligen. Hierzu müssen geeignete Methoden zum Einlesen der Streckenbeschreibung bereitstehen[5].

Des weiteren wird eine Liste aller Knoten und Kanten geführt, so daß ein Knoten oder eine Kante ohne weiteren Aufwand anhand der zugehörigen Nummer lokalisiert werden kann. Dies ist z.B. beim Ersteinsatz neuer Züge im laufenden Betrieb vorteilhaft.

A.2 Positionsermittlung

A.2.1 Repräsentation der Positionsdaten

Die Klasse TPortIO (Datei BASEIO.H)

```
// *****************************************************************
// * Class TPortIO                                                 *
// *---------------------------------------------------------------*
// * regelt die Ein-/Ausgabe mit dem IR-Empfaenger                 *
// *****************************************************************
   class TPortIO {
         tBool bResponse_Error;     // Fehlerbit: Keine Antwort
         tBool bData_Corrupted;     // Fehlerbit: Uebertragungsfehler
         tBool DataComing();        // prueft ob ueberhaupt IR-Daten ankommen
         tBool DataWordReady();     // prueft ob ein vollst. Wort anliegt
      public:
         TPortIO();
         ~TPortIO();
         tBool Data_OK();
         void Reset();            // Hardware-Reset (z.B. nach Fehlempfang)
                                  // Fehler-Flags werden false.
         tBool Read(unsigned InWord[2]); // Liest die IR-Daten,
                    // *keine* Daten:
                    //  => bResponse_Error true && bData_Corrupted true,
                    //     Rueckgabe false
                    // zuwenig Daten:
                    //  => bResponse_Error false  && bData_Corrupted true,
```

[5]Gegenwärtig wird die Strecke im Konstruktor der Klasse aufgebaut.

```
                //     Rueckgabe true
                // ausreichend Daten:
                //  => bResponse_Error false  && bData_Corrupted false,
                //     Rueckgabe true
   };
```

Die Klasse TPortIO regelt die Datenübertragung vom Infrarot-Empfänger zum PC. Als Schnittstelle dient eine zusätzlich im PC installierte Multi-72 I/O-Steckkarte, die über einfache Speicherzugriffe angesprochen werden kann.

Die Klasse TIRPosition (Datei TRAIN.H)

```
// *******************************************************************
// * Class TIRPosition                                               *
// *-----------------------------------------------------------------*
// * zum Empfang der Positionsdaten direkt von der Lok               *
// * (bezogen auf Indexmarken und Motorumdrehungen)                  *
// * 2 Worte werden unterteilt in Datenbloecke:                      *
// *                                                                 *
// *   msb                                lsb                        *
// *   |----------------------------------|-----------|              *
// *   |             Word[1]              |  Word[0]  |              *
// *   | 16 ... 11,   10   , 9 , 8 ... 1  |  16 ... 1 |              *
// *   |----------------------------------|-----------|              *
// *   |  * ... * ,IndexDir,Dir, MarkCode |   CntVal  |              *
// *   |----------------------------------|-----------|              *
// *******************************************************************
   class TIRPosition {
     public:
         TIRPosition();
         // Destruktor wird implizit aufgerufen
         TIndex GetIndex();
         unsigned GetCntVal();
         tDirection Old_Motor_Dir;
         enum tIndexDir { away=-1, towards=0};
     protected:
         union {
           unsigned Word[2];  // vom IR-Empfaenger gelesene Worte
           struct {
             unsigned      CntVal;
                        // relative Position zur zuletzt ueberfahrenen Marke
             // -----------------------------------------------------------
             TIndex     MarkCode     : 8;    // Code der passierten Marke
                                             // (u.U. fehlerbehaftet)
             tDirection Motor_Dir    : 1;    // Drehrichtung des Motors
             tIndexDir  IndexDir     : 1;    // Zaehlrichtung
                          // Lok faehrt von Marke weg => Zaehler zaehlt hoch
                          // Lok faehrt zu Marke hin  => Zaehler zaehlt runter
                          // z.Zt. nicht benutzt!
             unsigned   Speed        : 6;    // z. Zt. nicht benutzt
           } Data;
         }; // end of union
         tBool MotorDirChanged() const;
   };
```

Die Klasse TIRPosition gibt die von der Lok empfangenen Positions- und Fahrtdaten an, wie sie vom IR-Empfängerbaustein geliefert werden. Eine Fehlererkennung oder -korrektur findet auf dieser Ebene nicht statt.

Die Daten liegen in nicht normalisierter Form vor. Positionsangaben erfolgen nach Kennung der letzten überfahrenen Indexmarke und Zählerstand des Abstandszählers auf der Lok. Richtungsangaben beziehen sich auf die tatsächliche Motordrehrichtung, unabhängig von der über das ZIMO-System eingestellten und von der tatsächlich auf der Gleisanlage beobachteten Fahrtrichtung.

Die Klasse TPosition (Datei TRAIN.H)

```
// ******************************************************************
// * Class TPosition                                                *
// *----------------------------------------------------------------*
// * zur Speicherung der aufbereiteten Position fuer jede Lok       *
// ******************************************************************
   class TPosition : public TIRPosition {
         enum { strict, sloppy } OldConsCheck, ConsistenceCheck;
         struct TBasePosition {
            TEdge *pEdge;    // Zeiger auf die aktuelle Kante
            unsigned RelPos; // rel. Position auf der Kante (in normalisierten
                             // Laengeneinheiten, bezogen auf Zaehlrichtg.)
            TIndex LastIndex; // zuletzt ueberfahrene Indexmarke
            long IndexDist;  // Abstand von letzter Indexmarke
            tIndexDir IndexDir;
            clock_t ClockTicks;
         } ActPosition,    // aktuelle Position
           Minor_Sync_Pos, // zuletzt fixierte Position (evtl. fehlerhaft)
           Major_Sync_Pos; // letzte gesendete Position
                           // (wird als sicher angenommen)

         unsigned long nEstimatedDistance;
         tDIRECTION Dir;      // Fahrtrichtung im oder gegen Uhrzeigersinn
         tBool LogicDirChanged;
         unsigned ActSpeed;
         double dConvertFactor;

         long AddTo( TBasePosition *pTmpPosition, long Distance );
         long GotoNextEdge( TBasePosition *pTmpPosition );
         long GotoNextIndex( TBasePosition *pTmpPosition );
         tBool DiffOutOfRange(         long lnRealDist,
                              unsigned long luEstimatedDistance);
         tBool Convert();
         tBool ChangeDir();

     public:

         TPosition( double dCF );
         ~TPosition();
         tBool ReceivePos();
         tBool ComputePos();
         tBool ForceAdjustment(int nShure);
         tBool NeedToSend();
         tBool UpdateSpeed(unsigned nNewSpeed);
         tBool UpdateDir(tDIRECTION NewDir);

         TEdge* GetEdge();     // liefert Zeiger auf die aktuelle Kante
         unsigned GetRelPos(); // liefert rel. Position auf der Kante
            // (in normalisierten Laengeneinheiten, bezogen auf Zaehlrichtg.)
         unsigned char GetLastIndex(); // liefert zuletzt ueberfahrene Indexmarke
         long GetIndexDist();  // liefert Abstand von letzter Indexmarke
         clock_t GetLastTime(); // liefert Zeit der letzten Positionsermittlung
```

```
//   private:

        tBool bSendSuccess;
   };
```

Wie in Abschnitt 3.1.1 erläutert wurde, reichen die in TIRPosition gespeicherten Angaben nicht zur eineindeutigen Positionsangabe aus. Daher werden mit der Klasse TPosition Datenelemente und Methoden zur Konvertierung der 'rohen' Positionsdaten in das Graphenmodell bereitgestellt, die den in Abschnitt 3.1.4 beschriebenen Algorithmus nutzen. Dies erfordert eine enge Interaktion mit den Klassen zur Repräsentation von Streckenabschnitten, TNode und TEdge. Die resultierende Position liegt nie in einem Knoten, sondern stets auf einer Kante[6].

Bei der Positionsermittlung wird die Lok als ausdehnungslos betrachtet; ihre Position wird als Punkt angegeben[7]. Da jedoch einerseits die Länge eines Zuges nicht vernachlässigbar klein ist, andererseits eine bereichsmäßige Positionsangabe Probleme bereitet, wird dieser Effekt im weiteren Verlauf beim Markierungsvorgang berücksichtigt.

A.3 Modellierung der Züge

A.3.1 Kommunikationsklassen

Die Klassen TSend (Datei BASEIO.H)

```
// ********************************************************************
// * Class TSend                                                      *
// *------------------------------------------------------------------*
// * Byte wird nach Funktionalitaet unterteilt in Datenbloecke:       *
// *                                                                  *
// *    msb               lsb                                         *
// *    |-------------------|                                         *
// *    |       Byte        |                                         *
// *    | 8,7, 6 ,5,4,3,2,1 |                                         *
// *    |-------------------|                                         *
// *    | *,Z,Dir,L,v,v,v,v |                                         *
// *    |-------------------|                                         *
// ********************************************************************
   class TSend {
     public:
         union {
           unsigned char Byte;      // Fahrbefehl fuer Lok
           struct {
             unsigned     v : 4;    // Geschwindigkeit 0..15
             tBool        L : 1;    // Sonderfunktion "Licht"
                                    // (=>hier: "Position senden")
             tDirection Dir : 1;    // Fahrtrichtung, 1:vorw., 0:rueckwaerts
             tBool        Z : 1;    // Sonderfunktion, z.Zt. nicht benutzt
             int            : 1;    // nicht benutzt, konstant = 0
           } Data;
         }; // end of union
     public:
         TSend(unsigned char Address);
```

[6] Dies ist stets zu erreichen, da Knoten ausdehnungslos sind.

[7] Dieser Punkt gibt in Anlehnung an das verwendete Verfahren zur Positionsermittlung auf der Lok die genaue Position des Indexmarkenlesers des betreffenden Fahrzeugs an.

```
      ~TSend();
      unsigned char TrainAddress; // Adresse der Lok
};
```

In der Klasse TSend werden Zimo-Fahrbefehle gekapselt. Eine Dokumentation der Datenfelder findet sich in Anhang B und in [2].

Die Klasse TCondData (Datei BASEIO.H)

```
// ********************************************************************
// * Class TCondData                                                  *
// *------------------------------------------------------------------*
// * 2 Bytes werden nach Funktionalitaet unterteilt in Datenbloecke:  *
// *                                                                  *
// *    msb                                                lsb        *
// *    |-------------------------|----------------------------|      *
// *    |         Byte[1]         |          Byte[0]           |      *
// *    | 8,7,6,5, 4 , 3 , 2 , 1  |  8 , 7 , 6 , 5, 4, 3, 2, 1 |      *
// *    |-------------------------|----------------------------|      *
// *    | * * * * max max kri kri | ZK3 ZK2 ZK1 bb bb bb ul ul |      *
// *    |-------------------------|----------------------------|      *
// ********************************************************************
   class TCond {
     public:
        union {
          unsigned char Byte[2];
          struct {
            unsigned       ul     : 2;    // weiss nicht, was das ist
            unsigned       bb     : 3;    // weiss auch nicht...
            tBool          ZK1    : 1;    // Sonderfunktion, nicht benutzt
            tBool          ZK2    : 1;    // Sonderfunktion, nicht benutzt
            tBool          ZK3    : 1;    // Sonderfunktion, nicht benutzt
            // -----------------------------------------------------
            unsigned       kri    : 2;    // Kriechpulse (was ist das?)
            unsigned       max    : 2;    // Maximalgeschwindigkeit
            int                   : 4;    // nicht benutzt, konstant = 0
          } Data;
        }; // end of union (no name)
     public:
        TCond(unsigned char Address, unsigned char ul, unsigned char bb,
              unsigned char max, unsigned char kri);
        ~TCond();
        unsigned char TrainAddress;
   };
```

In der Klasse TCondData werden Zimo-Konditionierungsbefehle gekapselt. Eine Dokumentation der Datenfelder findet sich in Anhang B und in [2].

Die Klasse TZimoBaseIO (Datei BASEIO.H)

```
// ********************************************************************
// * Class TZimoBaseIO                                                *
// *------------------------------------------------------------------*
// * regelt das Absetzen von Zimo-Befehlen (ueber ser.Schnittst.)     *
// ********************************************************************
   class TZimoBaseIO {
        void SerOut(unsigned char aByte);
             // gekapselter Zugriff auf ser. Schnittstelle
```

```
    public:
       TZimoBaseIO(); // Initialisierung der ser. Schnittstelle
       ~TZimoBaseIO();
       TZimoBaseIO& operator<< (TSend&); // Zimo-Fahrbefehl absetzen
       TZimoBaseIO& operator<< (TCond&); // Zimo-Konditionierung absetzen
  };
```

Die Klasse TZimoBaseIO stellt ein Low-Level-Interface zur Ansteuerung des Zimo-Systems über die serielle Schnittstelle bereit. Eine genaue Dokumentation der Zimo-Befehle findet sich in Anhang B und in [2]. Die Ansteuerung der seriellen Schnittstelle ist u.a. in [7] und in [9] beschrieben.

A.3.2 Triebwagenansteuerung

Die Klasse TZimoTrain (Datei TRAIN.H)

```
// *********************************************************************
// * Class TZimoTrain                                                  *
// *-------------------------------------------------------------------*
// * repraesentiert eine einzelne Lok, regelt deren Funktionen         *
// *********************************************************************
   class TZimoTrain {
         TSend *pSendData; // Fahrdaten
         TCond *pCondData; // Konditionierungsdaten
      protected:
         unsigned nCommandRepeat;   // Anzahl der Befehls-Wiederholungen
                                    // wenn Lok nicht reagiert
         unsigned nCritCommandRepeat; // Anzahl der Befehls-Wiederholungen
                                      // ohne Rueckmeldung
                                      // ("Licht aus" wird nicht bestaetigt)
         virtual tBool set_v(char speed); // aendert Geschwindgk.
         virtual void change_dir(); // aendert Fahrtrichtg.
         virtual tBool get_L();     // liefert akt. Licht-(Sender-)Status
         virtual void set_L();      // Licht (Sender) an
         virtual void reset_L();    // Licht (Sender) aus
         virtual void toggle_L();   // Lichtbit (Sender) toggeln
         virtual void send(unsigned del);// fuehrt Aenderungen der Fahrdaten aus
         virtual void cond(unsigned del);// fuehrt Neukonditionierung aus
      public:
         TZimoTrain(char TrainSign);
         virtual ~TZimoTrain();
         virtual tDirection get_dir();    // liefert akt. Fahrtrichtung
         virtual char get_v();      // liefert akt. Geschwindigkeit (0..15)
   };
```

Die Klasse TZimoTrain kapselt die spezifischen Besonderheiten einzelner Züge bei der Ansteuerung über das Zimo-System. In dieser Klasse sind alle Fahrtdaten intern gespeichert. Eine Änderung wird erst wirksam, wenn sie auf dieser Klasse durchgeführt und mittels TZimoTrain::send() bzw. TZimoTrain::cond() an das Zimo-Gerät abgeschickt wurde.

Die Klasse TFullTrain (Datei TRAIN.H)

```
// *********************************************************************
// * Class TFullTrain                                                  *
// *-------------------------------------------------------------------*
```

```
// * regelt die Funktionen einer Lok auf hoeherer Ebene                 *
// ***********************************************************************
   class TFullTrain : public TZimoTrain {
         char chTrainSignature;    // Kennung des Zuges
         unsigned nTrainLength;    // Laenge der Lok in mm
         unsigned bwd_fix, fwd_fix;
         unsigned inc_delay, dec_delay;
         unsigned SpeedArray[16];  // Normalisierte Geschwindigkeit
                                   // (Zehntel-mm pro Sekunde)
         char nNominalV;
         TPosition *pPosition;     // Position der Lok
         TCollClass *pFwdCollision, *pBwdCollision;
         TTrainDisplay* pDataWin;  // Fenster zur Anzeige der Fahrdaten
         unsigned nSightWidth;     // Sichtweite
         virtual void InitSpeed(unsigned, unsigned, unsigned, unsigned,
                                unsigned, unsigned, unsigned, unsigned,
                                unsigned, unsigned, unsigned, unsigned,
                                unsigned, unsigned, unsigned, unsigned);
                   // setzt die normalisierte Geschwindigkeit fuer eine Lok
     public:
         tBool NodeLocked; // private machen
   // ----------------------------------------------------------------------
   // zur Listenbearbeitung noetige Operatoren:
         int operator==(const TFullTrain& Train) const;
   // ----------------------------------------------------------------------
         TFullTrain(char TrainSign);
         virtual ~TFullTrain();
         virtual char GetTrainSign() const ;      // liefert Signatur des Zuges
         virtual unsigned GetFixDist(tMarkType NewMarkType);
         virtual unsigned GetSightWidth() const ; // liefert Sichtweite
         virtual unsigned GetCritWidth(); // liefert kritische Distanz
                                          // Geschw.-abhaengig
         virtual unsigned GetCritWidth(unsigned CheckSpeed); // kritische Distanz
                                          // fuer uebergebene Geschwindigkeit
         virtual TPosition* GetPos(); // berechnet oder sendet Position
         virtual TPosition* ReadPos() const ; // liefert pPosition
         virtual tBool Send();       // sendet Zimo-Befehle,
                                     // ermittelt akt. Position
         virtual tBool Activate();   // aktiviert diesen Zug
         virtual tBool InActivate(); // deaktiviert diesen Zug
         virtual unsigned Speed();   // liefert normalisierte Geschwindigkeit
         virtual tBool SetNomV(char v); // setzt Sollgeschwindigkeit
         virtual tBool SetV(char v); // setzt Geschwindigkeitsstufe
         virtual tBool IncSpeed(const unsigned nNewSpeed);
                                     // erhoeht Geschwindigkeit
         virtual tBool DecSpeed(const unsigned nNewSpeed);
                                     // erniedrigt Geschwindigkeit
         virtual tBool Stop();       // haelt Zug an (ohne Positionsabfrage)
         virtual tBool ChangeDir();  // aendert Richtung
         virtual tDIRECTION GetDir();// liefert Richtung
         virtual tBool Toggle_L();   // toggelt L-Signal
         virtual tBool Look( tMarkType LookDir );
         virtual tBool HandleCollision();
         virtual tBool Mark();
         virtual tBool Unmark();
         virtual tBool Execute(unsigned char chCommand);
   };
```

Die Klasse TFullTrain repräsentiert einen Zug auf der Anlage. Es werden Methoden bereitgestellt, mit denen eine Steuerung weitgehend hardwareunabhängig und auf hoher Ebene möglich wird. Aufgrund von Einschränkungen durch hardwaretechnische Randbedingungen ist es ebenfalls erforderlich, einige Methoden zum Zugriff auf hardwarenahe Datenelemente zugänglich zu machen.

Diese Klasse greift auf Methoden der Klasse TZimoTrain zurück und bildet die Schnittstelle zur Ansteuerung der Züge sowohl für die Benutzungsoberfläche als auch für die Kollisionsbehandlungsroutinen.

Die Klasse TAllTrains (Datei TRAIN.H)

```
// ********************************************************************
// * Class TAllTrains                                                 *
// *------------------------------------------------------------------*
// * Regelt Zusammenspiel aller Zuege und Kollisionsvermeidung        *
// ********************************************************************
   class TAllTrains {
         BI_IListImp<TFullTrain> TrainList;
         typedef BI_IListIteratorImp<TFullTrain> TrainListIteratorType;
         TrainListIteratorType *pTrainListIterator;
         char chActiveTrain;
     public:
         TAllTrains(char *achTrainsToMake = "");
         ~TAllTrains();
         tBool IsTrainSign(char chKey);  // true wenn chKey einen Zug bezeichnet
         TFullTrain* GetTrain(char chTrainSig);
         tBool ActivateTrain(char chTrainSig);
         tBool AllStop(); // haelt alle Zuege an
         tBool HandleTrains(unsigned char chCommand);
                            // Standard-Handler zur Kollisionserkennung
   };
```

Die Klasse TAllTrains stellt Methoden zur Überwachung aller auf der Anlage befindlichen Züge bereit. Im Zusammenhang damit werden Benutzereingaben abgefragt und ausgewertet. Für jeden Zug wird eine Positionsaktualisierung, eine Kollisionserkennung und im Konfliktfall eine Kollisionsvermeidung durchgeführt.

Für den Fall einer nicht auf reguläre Weise vermeidbaren Kollision stellt diese Klasse eine Methode zur Verfügung, alle Züge auf der Anlage unmittelbar anzuhalten.

A.4 Kollisionen

A.4.1 Färbung von Streckenabschnitten

Die Klasse TMark (Datei ROUTE.H)

```
// ********************************************************************
// * Class TMark                                                      *
// *------------------------------------------------------------------*
// * Repraesentiert eine Markierung auf einer Kante                   *
// ********************************************************************
   class TMark {
         unsigned nRelPos;       // Relative Position der Marke auf der Kante,
                                 // bezogen auf Uhrzeigersinn
         tMarkType MarkType;     // oeffnende Markierung
                                 //     (liegt *hinter* dem Zug --> behind)
                                 // schliessende Markierung
```

```
                             //      (liegt *vor* dem Zug --> infront)
        TEdge* pEdge;          // Zeiger auf die betreffende Kante
        TFullTrain* pMarkingTrain; // markierender Zug
//        unsigned nDistance;    // Abstand zum Zug
        unsigned nMarkDistance, // insges. zu markierender Abstand
                                // beginnend bei Spitze des Zuges
                 nRestDistance; // noch zu markierender Restabstand
                                // es kann nRestDistance>nMarkDistance
                                // wg. Beruecksichtigg. der Zuglaenge
    public:
        int operator==(const TMark& Mark) const;
        TMark( TMark *pMark );
        TMark( TFullTrain* pTrain, tMarkType NewMarkType );
        ~TMark();
        tBool SetEdge(TEdge* pEdgeToSet);
        tBool SetPos(unsigned nPosToSet);
        tBool UpdDist(unsigned nDiffDist);

        unsigned GetPos() const;
        unsigned GetDistToTrain() const;
        unsigned GetRestDist() const;
        tMarkType GetType() const;
        TEdge* GetEdge() const;
        TFullTrain* GetMarkingTrain() const;
  };
```

Diese Klasse repräsentiert eine Markierung auf einer Kante, mit der ein Streckenabschnitt als belegt markiert wird[8]. In Abhängigkeit von der Kantenlänge und der Sichtweite können sich die markierten Streckenabschnitte prinzipiell über beliebig viele Kanten erstrecken. Geeignete Markierungsprozeduren sorgen dafür, daß alle betroffenen Kantenteilstücke und Knoten markiert werden.

Die Klasse TMark realisiert ein Teilstück einer solchen Markierung, jeweils bezogen auf eine Kante. Dazu reicht es aus, die Anfangs- und Endposition des betreffenden Abschnitts in Längeneinheiten gemäß der festgelegten Zählweise zu speichern. Eine Überprüfung, ob eine Position in einer Markierung liegt, erfolgt dann durch einfachen Größenvergleich der relativen Längenwerte. Die Objekte vom Typ TMark werden listenartig in der jeweils betreffenden Kante verwaltet.

A.4.2 Kollisionsbehandlung

Die Klasse TCollClass (Datei TRAIN.H)

```
// *********************************************************************
// * Class TCollClass                                                  *
// *-------------------------------------------------------------------*
// * zur Klassifikation der Kollisionen                                *
// *********************************************************************
  class TCollClass {
        unsigned int wCollType; // Klassifikation der Kollisionsarten:
            // * msb                                           lsb       *
            // * |-----------------------------------------------|       *
            // * | 15,14,13,12,11,10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 |     *
            // * |-----+-----------------------------------------|       *
            // * | Bit | Bedeutung                                 |     *
            // * |  15 |                                           |     *
```

[8] Ein Knoten kann einfacher durch Setzen eines entsprechenden Attributs markiert werden, da er ausdehnungslos ist.

```
        // * |  14 |                                        |        *
        // * |  13 |                                        |        *
        // * |  12 |                                        |        *
        // * |  11 |                                        |        *
        // * |  10 |                                        |        *
        // * |   9 |                                        |        *
        // * |   8 |                                        |        *
        // * |   7 |                                        |        *
        // * |   6 |                                        |        *
        // * |   5 |                                        |        *
        // * |   4 |                                        |        *
        // * |   3 |                                        |        *
        // * |   2 |                                        |        *
        // * |   1 |                                        |        *
        // * |   0 |                                        |        *
        // * |------------------------------------------------|        *
                              // keine
                              // Frontal
                              // Seitlich
                              // Auffahren
                              // Ausweichen moeglich
                              // Richtung wechseln hilfreich
                              // Beschleunigen/Verlangsamen
      TFullTrain *pThisTrain; // Zeiger auf aufrufenden Zug
      TFullTrain *pRefTrain;  // Zeiger auf aufrufenden Zug,
                              // bei Konflikt: auf Konfliktzug
      unsigned nMarkDistance, // insges. zu ueberprufender Abstand
                              // beginnend bei Spitze des Zuges
               nRestDistance, // noch zu ueberpruefender Restabstand
                              // es kann nRestDistance>nMarkDistance
                              // wg. Beruecksichtigg. der Zuglaenge
               nDistToTrain;  // Abstand zwischen Konfliktmarke
                              // und Konfliktzug
      unsigned long nTotalDist,
                    nLastTotal;
  public:
      TCollClass( TFullTrain *pCallingTrain, tMarkType MarkType );
      ~TCollClass();
      TFullTrain* GetRefTrain(); // liefert Konfliktzug
      unsigned long GetDistToTrain(); // liefert Abstand zum Konfliktzug
      unsigned GetRestDist(); // liefert zu ueberpruefende Restdistanz
      unsigned GetPriority(); // liefert Prioritaet des aufrufenden Zuges
                              // 0:
                              // 1:
                              // 2:
                              // 3:
      tBool isOnEdge();       // true bei Kollision auf Kante,
                              // false bei Kollision im Knoten
      tBool isFrontal();      // true wenn Zuege aufeinander zufahren
                              // sonst false (gleiche Richtung oder Knoten)
      tBool CollDetect();     // true bei Kollision
      tBool isLookForward();  // true, wenn in Fahrtrichtung
                              // des aufrufenden Zuges, sonst false
      tBool isRaising();      // Abstand nimmt zu
      void reset( unsigned nPriority = 0 );
      void Check( TFullTrain *pTrain ); // Kollisionspruefung,
                                        // Aufruf aus Knoten heraus
      void Check( TMark *pMark, unsigned nCheckDist ); // Kollisionspruefung,
                                        // Aufruf von Kante aus
};
```

Mit der Klasse TCollClass wird die Klassifikation von Kollisionen realisiert.

Die genannten Kriterien sowie ggf. hilfreiche Zusatzinformationen, die sich im Verlauf der Markierung von Streckenabschnitten ergeben, sind hier in Datenelementen gespeichert. Beim Markierungsvorgang wird ein Objekt vom Typ TCollClass mitgeführt, um dort relevante Informationen abzulegen. Wird im weiteren Verlauf eine Kollision erkannt, so stehen die Kollisionsklasse und ggf. weitere Zusatzinformationen, z.B. über die Distanz zwischen den Kollisionsgegnern o.ä., ohne zusätzlichen Aufwand zur Verfügung.

A.5 Benutzungsschnittstelle

A.5.1 Die Benutzungsoberfläche

Die Klasse TUserDisplay (Datei BASEIO.H)

```
// ********************************************************************
// * Class TUserDisplay                                               *
// *------------------------------------------------------------------*
// * zeigt die Aktivitaeten auf dem Bildschirm an                     *
// ********************************************************************
   class TUserDisplay {
         unsigned TrainsOnTrail;      // Anzahl der Zuegen auf der Anlage
         constream IRDetectWin, ZimoWatchWin,            // veraenderliche...
                   WholeWin, StatusWin, FrameMessageWin; //...statische Fenster
     public:
         constream MessageWin;  // Ausgabestream ins Message-Fenster
         ofstream Temp1Win,
                  Temp2Win,
                  Temp3Win;     // temporaere Ausgabestreams zu Debug-Zwecken
         TUserDisplay();
         ~TUserDisplay();
         TTrainDisplay* NewTrainWin(TFullTrain* pCallingTrain);
                                      // liefert neues Zug-Display-Fenster
         constream& GetIRDetectWin();
         constream& GetZimoWatchWin();
   };
```

Die Klasse TUserDisplay stellt dem Benutzer eine intuitiv verständliche Benutzungsoberfläche zur Verfügung. Informationen über Fahrtrichtung, Geschwindigkeit und Position der einzelnen Züge werden unter Rückgriff auf Methoden der Klasse TTrainDisplay präsentiert; darüberhinaus werden in drei weiteren Fenstern weitere Informationen ausgegeben:

1. Das Fenster '**Message**' beinhaltet allgemeine Informationen über den Systemstatus und die Plausibilität der berechneten Daten. Hier wird beispielsweise eine Mitteilung über eine gescheiterte Plausibilitätskontrolle nach der Umwandlung empfangener Positionsdaten ausgegeben.

2. Im Fenster '**IR-Detect**' werden die vom IR-Empfänger gelesenen Bits unverändert in Hexadezimaldarstellung ausgegeben. Im Falle einer Empfangsstörung erscheint eine entsprechende Meldung.

3. Im Fenster '**ZIMO-Cmd**' werden die über die serielle Schnittstelle an das Zimo-System abgesetzten Befehle byteweise protokolliert. Die erste Zahl gibt die Zugadresse an, die darauffolgende Hexadezimalzahl die anschließenden Datenbytes.

Darüberhinaus erhält der Benutzer die Möglichkeit, einzelne Züge selbst anzusteuern. Tastatureingaben werden, soweit es sich um Oberflächenkommandos handelt, ausgewertet oder, falls es sich um Steuerungskommandos für Züge handelt, entsprechend weitergereicht. Die folgenden Kommandos werden erkannt:

+	Beschleunigung des aktiven Zuges um eine Fahrstufe
-	Abbremsen des aktiven Zuges um eine Fahrstufe
R	Richtungsumkehr des aktiven Zuges
P	Aufforderung an den aktiven Zug, seine Position zu senden
L	Ein-/Ausschalten('Toggle') der Lichtfunktion des aktiven Zuges[9]
W	Erneute Aussendung des aktuellen Fahrbefehls für den aktiven Zug
0..9	Direkte Anwahl der Fahrstufen 0 bis 9 für den aktiven Zug
[Shift] 0..5	Direkte Anwahl der Fahrstufen 10 bis 15 für den aktiven Zug
A..C	Auswahl des aktiven Zuges[10]
S	Ein 'Notstop'-Befehl wird an alle Züge auf der Anlage ausgesendet
X	Alle Züge werden angehalten, das Programm wird beendet

Die Klasse TTrainDisplay (Datei BASEIO.H)

```
// ********************************************************************
// * Class TTrainDisplay                                              *
// *------------------------------------------------------------------*
// * Zur Bildschirmausgabe der Fahrdaten eines zugeordneten Zuges     *
// * Es existiert genau eine Instanz pro Zug                          *
// ********************************************************************
   class TTrainDisplay {
         TFullTrain* pThisTrain;   // zugeordneter Zug
         constream OutWin;         // Ausgabezeile
     public:
         TTrainDisplay(int nLinesInUse, TFullTrain* pCallingTrain);
         ~TTrainDisplay();
         void ActivateDisplay();   // aktiviert Display fuer zugeordneten Zug
         void InActivateDisplay(); // deaktiviert Display fuer zugeordneten Zug
         // Anzeige der Fahrdaten des zugeordneten Zuges:
         void DisplayDir();        // Fahrtrichtung
         void DisplaySpeed();      // Geschwindigkeit
         void DisplayPos();        // Position anzeigen
   };
```

Diese Klasse dient zum Aufbau eines Informationsfensters auf dem Bildschirm, in dem die Fahrt- und Positionsdaten eines zugeordneten Zuges ausgegeben werden. Da hier die Aussagekraft der gewählten Darstellungsform sehr vom Anwendungsfall abhängt, wurde die Anzeige zur leichteren Wartbarkeit in einer

[9] Aus Effizienzgründen wurde die Lichtfunktion auf der Lok zum Senden der Positionsdaten abgeändert. Um die Dokumentation konsistent zu halten, wird jedoch auch weiterhin von der 'Licht'- oder L-Funktion gesprochen.

[10] Da zum Test drei Züge zur Verfügung standen, wurde die Kombination A..C gewählt. Bei Bedarf sind die Kennungen erweiterbar, solange keine Verwechslung mit einem sonstigen Befehl möglich ist

eigenen Klasse gekapselt. Von der Klasse TTrainDisplay existiert genau eine Instanz zu jedem auf der Gleisanlage befindlichen Zug.

Im entsprechenden Fenster werden zu jedem auf der Gleisanlage befindlichen Zug die folgenden Daten angegeben:

- Kennung des Zuges mit Markierung des aktiven Zuges[11]
- Fahrtrichtung des Zuges
- Fahrstufe des Zuges nach Einstellung des Zimo-Systems
- Normalisierte Fahrgeschwindigkeit des Zuges in $\frac{mm}{10}$
- Auf der Lok gespeicherte Indexmarke (gemäß der letzten empfangenen Positionsangabe)
- Stand des Abstandszählers auf der Lok (gemäß der letzten empfangenen Positionsangabe)
- Verstrichene Zeit in ClockTicks
- Aktuelle Position des Zuges nach Kantennummer und relativer Position auf der Kante in mm, gemessen in Zählrichtung

[11]Aktiv heißt der Zug, der auf einen Steuerungsbefehl des Benutzers – z.B. 'Beschleunigung um eine Fahrstufe' – reagiert

Anhang B

Technische Daten des Zimo-Systems

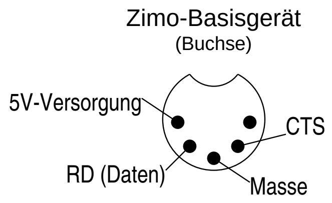

Abbildung B.1: 5-polige DIN-Buchse für Computeranschluß

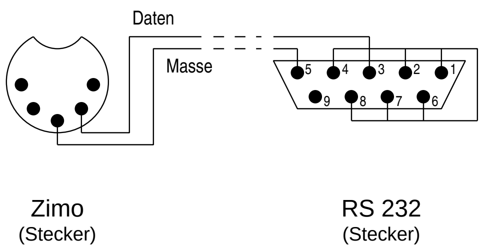

Abbildung B.2: RS232 Anbindung des Zimo-Basisgeräts ohne CTS-Leitung

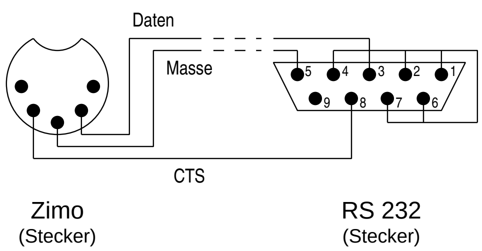

Abbildung B.3: RS232 Anbindung des Zimo-Basisgeräts mit CTS-Leitung

Abbildung B.4: Sammelbefehle

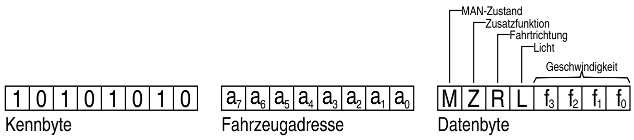

Abbildung B.5: Aufbau der Fahrbefehle

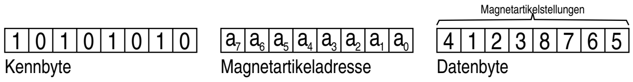

Abbildung B.6: Aufbau der Magnetartikel-Steuerbefehle

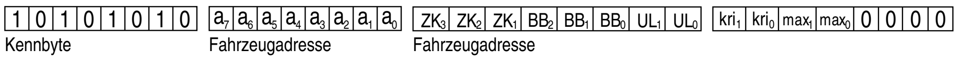

Abbildung B.7: Aufbau der Konditionierungsbefehle

Transferrate (Baud rate)	2400 bit / s
Datenformat	8 bit
Paritätsprüfung im M1000	keine
Anzahl der Stopbits	1 oder 2

Abbildung B.8: Schnittstellenspezifikation für RS232

Datenübertragungskapazität Basisgerät–Schiene bei Computersteuerung	
ohne Einsatz von Fahrpulten	35–40 Steuerbefehle / s
bei zusätzlichem Einsatz von Fahrpulten	15–20 Steuerbefehle / s
Maximale Anzahl an Fahr-/Konditionierungsbefehlen	
ohne Einsatz von Fahrpulten	10 Befehle / s
bei zusätzlichem Einsatz von Fahrpulten	5 Befehle / s

Abbildung B.9: Timing der Steuerbefehle bei Computereinsatz

ohne Booster	
zulässige Versorgungsspannung(Eingang 'TRAFO')	24 bis 28 V
max. Fahrstrom (Ausgang 'SCHIENE Fahrstromkreis')	5 bis 6 A
max. Gesamtfahrstrom (Ausgänge H & L)	2,5 A
max. Versorgungsstrom f. Fahrpulte (Stromkreis 'SYS')	3 A
max. Gesamtkabellänge für alle Fahrpulte	40 m
Abmessungen (B*H*T)	380 * 80 * 200 mm
Gewicht	3,5 kg
mit Booster M1000BT	
zulässige Versorgungsspannung(3 Eingänge 'TRAFO')	je 24 bis 30 V
max. Fahrstrom (3 Ausgänge 'SCHIENE Fahrstromkreis')	je 5 bis 6 A
max. Gesamtfahrstrom (Ausgänge H & L)	2,5 A
max. Versorgungsstrom f. Fahrpulte (Stromkreis 'SYS' 5V)	3 A
max. Versorgungsstrom (15 V)	3 A
max. Gesamtkabellänge für alle Fahrpulte	40 m
Abmessungen (B*H*T)	380 * 80 * 200 mm
Gewicht	4,5 kg

Abbildung B.10: Technische Daten des Basisgeräts M1000

Fahrspannung über Schiene	12 bis 22 V
max. Motorstrom	
bei Dauerlast	0,8 A
Spitzenlast für max. 5 sec.	2 A
max. Stirnlampenstrom	
bei direktem Anschluß am Empfänger	0,1 A
bei Anschluß über ext. Verstärkermodul M4000-V	0,8 A
bei Anschluß über ext. Stirnlampenmodul M4000-M	0,2 A
max. Strom Z-Funktion (nur über Verstärkermodul M4000-Z)	0,5 A
max. Strom ZK-Funktion (nur über Verstärkermodul M4000-Z)	0,5 A
Betriebstemperatur	0 – 80° C
Abmessungen	
M4000G, M4000W	24 * 12 * 7 mm
M4000F	29 * 15 * 8 mm
M4000-V, M4000-Z (Verstärkermodul)	12 * 10 * 7 mm
M4000-M (Stirnlampenmodul)	12 * 8 * 8 mm

Abbildung B.11: Technische Daten der Fahrzeug-Empfänger M4000G/M4000W

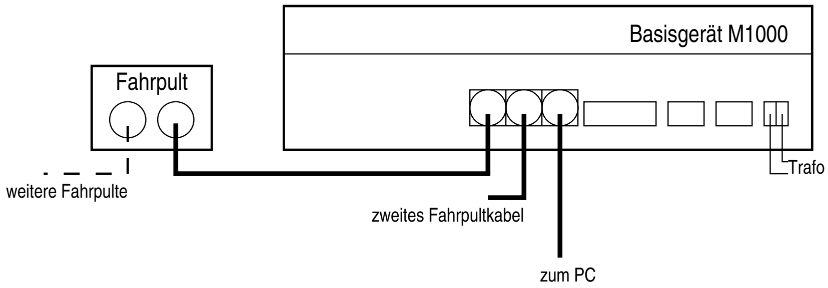

Abbildung B.12: Anschlüsse des Basisgeräts

Anhang C

IR-Datenübertragung

Abbildung C.1: Übertragene Bits

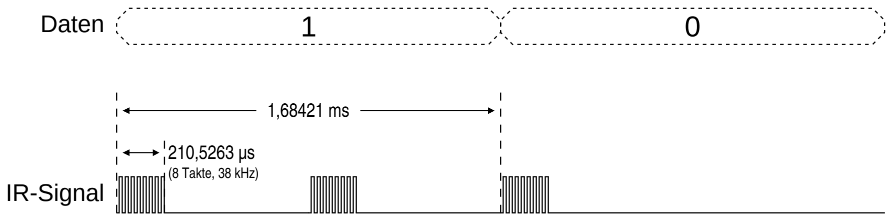

Abbildung C.2: Timingdiagramm der Senderseite

Taktfrequenz	2.457600 MHz
Versorgungsspannung	5 V / 12 V
Typ des eingesetzten FPGAs	Altera EPM 7128 LC 84
Reichweite	ca. 6 m
Verwendetes Modulationsverfahren	Pulse Code Modulation
Modulationsfrequenz	38 kHz
Paketlänge	8 Bit
Datenübertragung	seriell
Transferrate (Baud rate)	600 Bit / s
Datenformat	8 bit
Parität	keine
Anzahl der Start-/Stopbits	1 Start-, 1 Stopbit

Abbildung C.3: Technische Daten des IR-Senders

Taktfrequenz	2.457600 MHz
Versorgungsspannung	5 V
Typ des eingesetzten FPGAs	Altera EPM 7128 LC 84
Typ der eingesetzten IR-Empfängerbausteins	Sharp IS 1460
Reichweite	ca. 6 m
Breite des parallen Datenausgangs	16 Bit
Serieller Ausgang	
Transferrate (Baud rate)	600 Bit / s
Datenformat	8 bit
Parität	keine
Anzahl der Start-/Stopbits	1 Start-, 1 Stopbit

Abbildung C.4: Technische Daten des Empfängermoduls

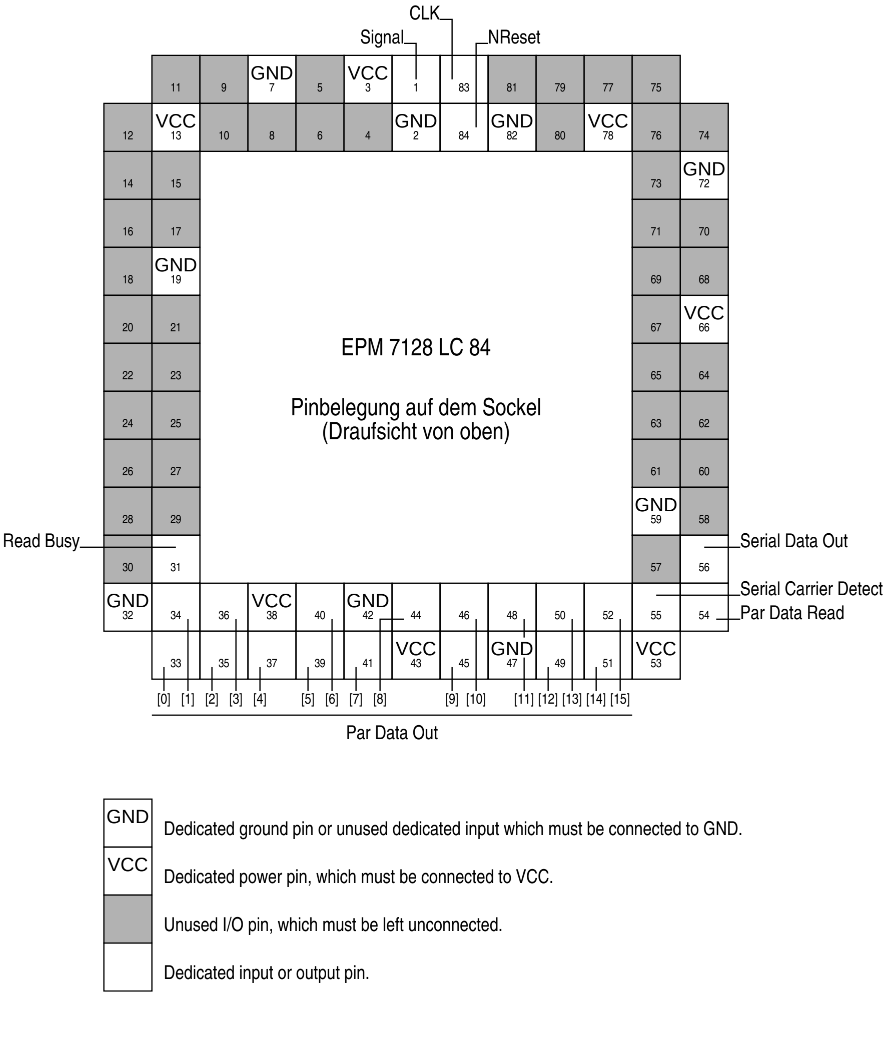

Abbildung C.5: Pinbelegung des FPGA im Empfängermodul

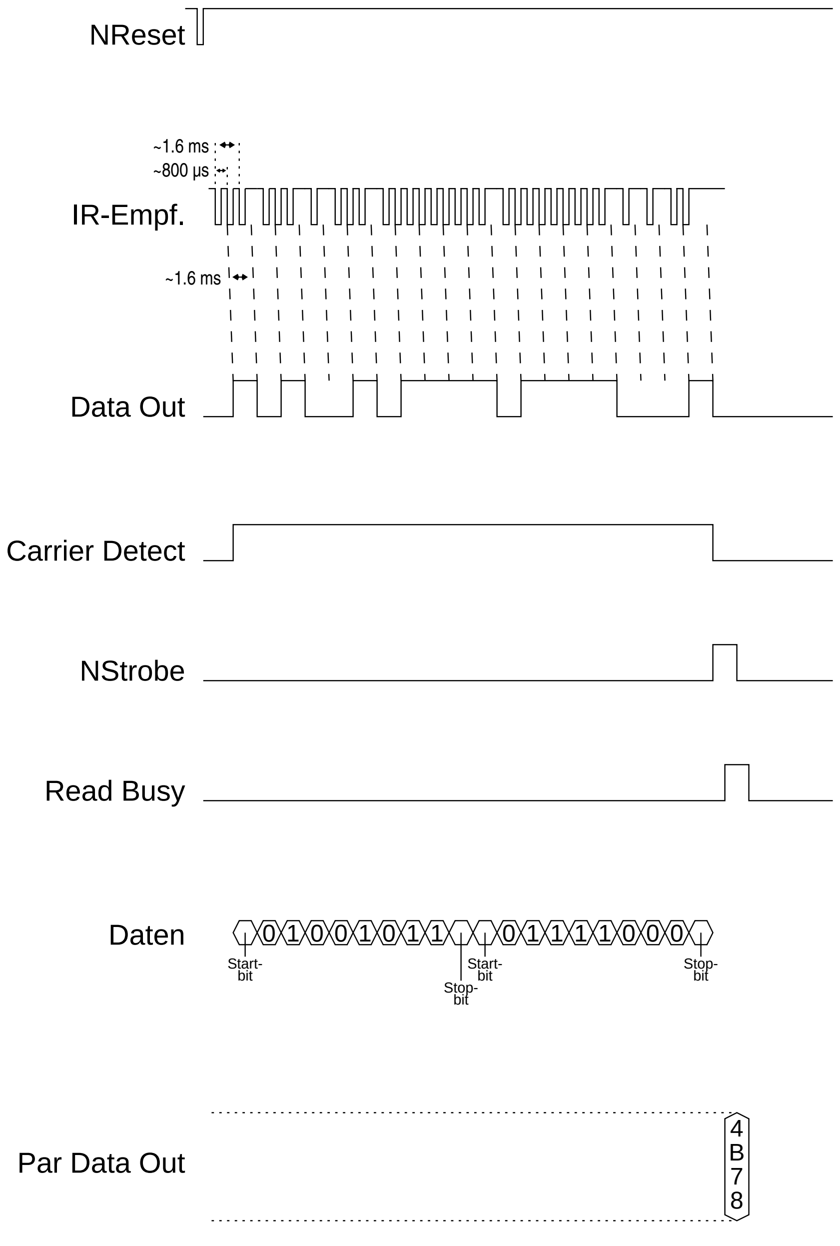

Abbildung C.6: Timingdiagramm des Empfängermoduls

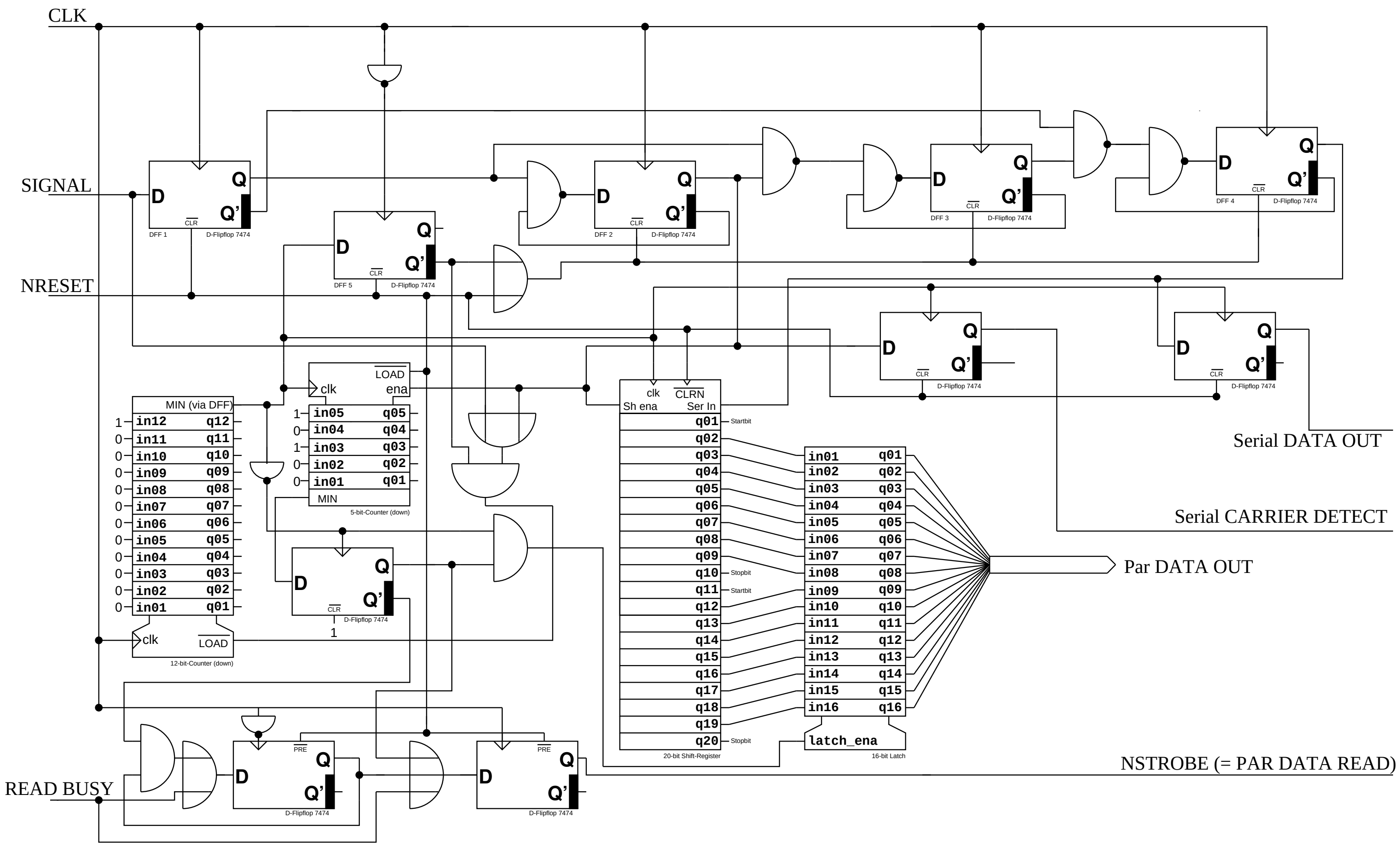

Abbildung C.7: Schaltplan Decodiererbaustein

Abbildungsverzeichnis

Literaturverzeichnis

[1] Rudolf Rusterholz. Zimo-Steuerungssystem. In: Die Modell-Eisenbahn, 9. Jg., Nr. 4., S. 36ff. Fachpresse Goldach Hudson & Co., Goldach(CH), 1988

[2] Zimo-Betriebshandbuch Oktober 1991. ZIMO Elektronik, Wien, 1991

[3] Lars Larsson, Bernd Schütz et al. Teaching System Integration Using PLDs. Universität Hamburg – FB Informatik, AB Technische Grundlagen, Hamburg 1994

[4] Duden Informatik. Bibliographisches Institut & F.A. Brockhaus AG, Mannheim, 1993

[5] Robert Sedgewick. Algorithmen in C. Addison–Wesley Deutschland GmbH, Bonn, 1992

[6] Bjarne Stroustrup. Die C++ – Programmiersprache. Addison–Wesley Deutschland GmbH, Bonn, 1992

[7] Borland C++ Referenzhandbuch. Borland GmbH, Starnberg, 1992

[8] Borland C++ Programmierhandbuch. Borland GmbH, Starnberg, 1992

[9] Arne Schäpers. DOS 5.0 für Programmierer. Addison–Wesley Deutschland GmbH, Bonn, 1991

[10] Jürgen Duensing. Signalbuch für die Modellbahn. Alba Buchverlag, Düsseldorf, 1976

[11] Die Anwender-Programmierung des Betriebssteuergerätes M5000. In: Zimo-Jahreskatalog 1992/93, S 22f. ZIMO Elektronik, Wien, 1992