Upcycling einer Modelleisenbahn mit intelligenter Steuerung

Bachelorarbeit

von
Philipp-Henning Salla

20. Februar 2023

vorgelegt im
Fachgebiet für verteilte Systeme der Informatik
Erstprüfer: Prof. Dr. Torben Weis
Zweitprüfer: Prof. Dr. Gregor Schiele

Fakultät für Ingeneurwissenschaften
Universität Duisburg-Essen

Inhaltsverzeichnis

1 Einführung

Die Entsorgung von veralteten Produkten, wie beispielsweise Modelleisenbahnen, ist ein voreiliger Schritt. Stattdessen lassen sich viele Produkte durch *Upcycling* zu einem neuen Produkt umwandeln, dass dann sowohl im ursprünglichen Anwendungsbereich wieder verwendet werden, als auch neue Anwendungsbereiche erschließen kann.

Die Arbeit beschäftigt sich mit dem Upcycling eines veralteten Modelleisenbahnsystems eines Klemmbausteinherstellers[1], um einen Betrieb mit mehreren Lokomotiven auf dem selben Gleissystem zu ermöglichen und die Steuerung der einzelnen Lokomotive mit intelligenten Funktionen zu erweitern. Es soll evaluiert werden, wie sich ein Betrieb mit mehreren Lokomotiven mit dem vorhandenen Gleissystem umsetzen lässt und dazu eine microcontrollergestützte Steuerung für die Lokomotive entwickelt werden.

Bei der Entwicklung der Software der Lokomotive liegt der Fokus auf der Kommunikation mit einem übergeordneten System und dem Hinzufügen von intelligenten Funktionen zur Steuerung. Dabei gilt es zu untersuchen, ob und welche Auswirkungen Situationen wie das Befahren von Kurven oder das Beschleunigen haben und wie diese in der Steuerung der Lokomotive intelligent integriert werden können.

Der Hintergrund dieser Arbeit ist dabei, sowohl das Aktualisieren des Modelleisenbahnsystems, aber auch die zukünftige Verwendung der Modelleisenbahn als Teil der Forschung auf dem Gebiet der Pysical-Digital Twins und des Transfer-Learnings (Lee u. a. (2020), Maschler u. a. (2021)).

In der Einführung wird zunächst eine Verbindung zu verwandten Arbeiten hergestellt und das Ziel dieser Arbeit aufgezeigt. Daraufhin werden die, für diese Arbeit erforderlichen Grundlagen in Kapitel 2 erläutert. Darunter fällt die Beschreibung des ursprünglichen Systems und Kapitel 3 gilt der Beschreibung der zur Umsetzung dieser Arbeit verwendeten Hardware und die Auswahl der Bauteile. Abgeschlossen wird das Kapitel durch eine Darstellung und Beschreibung des entwickelten Gesamtsystems, bezüglich der Hardware.

Der Kern dieser Arbeit wird in Kapitel 4 behandelt. Hier wird das entwickelte Kommunikationsprotokoll zur Kommunikation zwischen der Lokomotive und einem übergeordneten System mit allen Befehlspaketen erläutert. Außerdem wird die Relevanz einiger Situationen bei der Beschleunigung und dem Befahren von Kurven untersucht und intelligente Funktionen entwickelt, die in den entsprechenden Situationen durch die Steuerung der Lokomotive Anwendung finden. Die entwickelte API zur Steuerung der Lokomotive

[1] LEGO

durch eine externe Software, sowie das zusätzlich entwickelte Konsoleninterface zur Steuerung durch den Benutzer, werden ebenfalls in diesem Kapitel beschrieben. Zuletzt wird die erstmalige Einrichtung des WLAN der Lokomotive erläutert.

Die prägnantesten Problematiken, die während der Entwicklung des Systems aufgetreten sind, werden in Kapitel 5 beschrieben. Dabei wird erläutert, was unternommen wurde, um die entsprechenden Probleme zu lösen. Am Schluss dieser Arbeit, wird diese zusammengefasst und ein Ausblick auf Weiterentwicklungen sowohl der Hardware, als auch der Software gegeben.

1.1 Verwandte Arbeiten

In der Kommunikation mit Modelleisenbahnen wird voranging das Digital Command Control DCC Protokoll als Standard angesehen[2].

In der Arbeit von Zhao (2022) wird das DCC Protokoll für die Kommunikation mit der Modelleisenbahnanlage verwendet. Dabei wird die DCC Zentrale, die die Befehle zu den Lokomotiven sendet durch ein FPGA realisiert. Der Empfang dieser Signale setzt jedoch einen Decoder in der Lokomotive voraus. Eine ähnliche Implementierung, wie von Zhao (2022) ist bei dem System des Klemmbausteinherstellers nicht ohne Weiteres möglich. Dazu müsste zwar auch eine Steuerelektronik in Form eines Decoders der Lokomotive hinzugeügt werden, allerdings auch eine Kontrollstation die zusätzlich zur Versorgungsspannung die Befehle über das Gleis an die Lokomotive senden muss.

Das DCC Protokoll wird ebenfalls in der Arbeit von Ursu u. Condruz (2017) verwendet. Hier liegt der Fokus darauf, die DCC Zentrale zu entlasten, was die Spannungsversorgung der Anlage betrifft. Die Zentrale soll dabei nur noch das Senden der Befehle zu den Lokomotiven übernehmen. Die Spannungsversorgung wird dann von dem entwickelten intelligenten Booster übernommen. Durch die Kommunikation über WLAN, wie sie in dieser Arbeit geplant ist, ist die Verwendung von intelligenten Boostern nicht erforderlich. Analog kann aber ein Booster als zusätzliches oder stärkeres Netztgerät verstanden werden, da die Befehle nicht über die Versogungsspannung übermittelt werden, sondern über das Netzwerk.

Bei beiden Arbeit wird das DCC Protokoll beibehalten, um die Kompatibilität mit anderen Modellen und Modelleisenbahnen zu erhalten. Im Fall des Loksystems des Klemmbausteinherstellers, kann auf das DCC Protokoll verzichtet und ein eigenes Protokoll zur Kommunikation über WLAN entwickelt werden.

[2] `https://www.nmra.org/dcc-rps-standards` (Abgerufen: 12.02.2023)

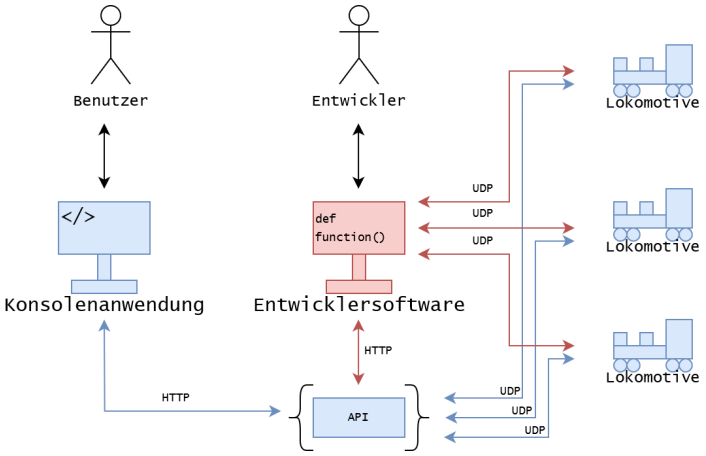

Abbildung 2.3: Geplanter Aufbau des Systems mit - in der Arbeit zu erstellenden Verbindungen - Software (Blau) und optionale externe Verbindungen - Software (Rot)

Damit ein Entwickler jedoch nicht ausschließlich - durch UDP Pakete - direkt mit der Lokomotive kommunizieren muss, soll eine API hinzugefügt werden, die sich an dem RESTful[4] Design orientiert. Aufgrund der gesendeten JSON Befehle zur Steuerung, wird jedoch ausschließlich die *POST* Methode verwendet. Die API soll dann auf einem anderen Computersystem im selben Netzwerk ausgeführt werden,HTTP Befehle von der Software eines Entwicklers entgegen nehmen und in Form von UDP Paketen an die entsprechende Lokomotive weiterleiten.

Für Benutzer, die keine eigene Software zur Steuerung der Lokomotive entwickeln möchten, soll eine Konsolenanwendung bereitgestellt werden, durch die ein Benutzer mit textbasierten Befehlen die Lokomotive steuern und von dieser Daten anfragen kann. In Abbildung 2.3 ist der zuvor beschriebene geplante Aufbau dargestellt.

Damit die Steuerung der Lokomotive externe Veränderungen, wie beispielsweise die Positionsänderung der Lokomotive, erfassen kann, ist ein entsprechender Sensor erforderlich. Für eine mögliche Messung der Geschwindigkeit der Lokomotive wäre ein Geschwindigkeitssensor am Motor verwendbar, wobei für das Erkennen und Ausgleichen von Kurven oder Steigungen, sowie der Ermittlung des Anfahrmoments ein Stromsensor erforderlich ist, der die Stromaufnahme des Motors misst.

[4] `https://de.wikipedia.org/wiki/Representational_State_Transfer` (Abgerufen: 06.02.2023)

Alternativ hätte aber auch ein Schrittmotor verwendet werden können. Dieser ermöglicht eine feinere Positionierung durch eine höhere Anzahl an Polen. Für die Modelllokomotive ist eine derart präzise Positionierung auf dem Gleis allerdings nicht von Bedeutung. Durch die höhere Anzahl an Polen ist der Motor größer - als der bereits vorhandene - und würde nicht in das Gehäuse des Motorblocks passen. Dazu müsste dann ein neues Gehäuse inklusive Radsatz und Getriebe durch 3D Druck angefertigt werden.

3.8 Hardware-Gesamtsystem

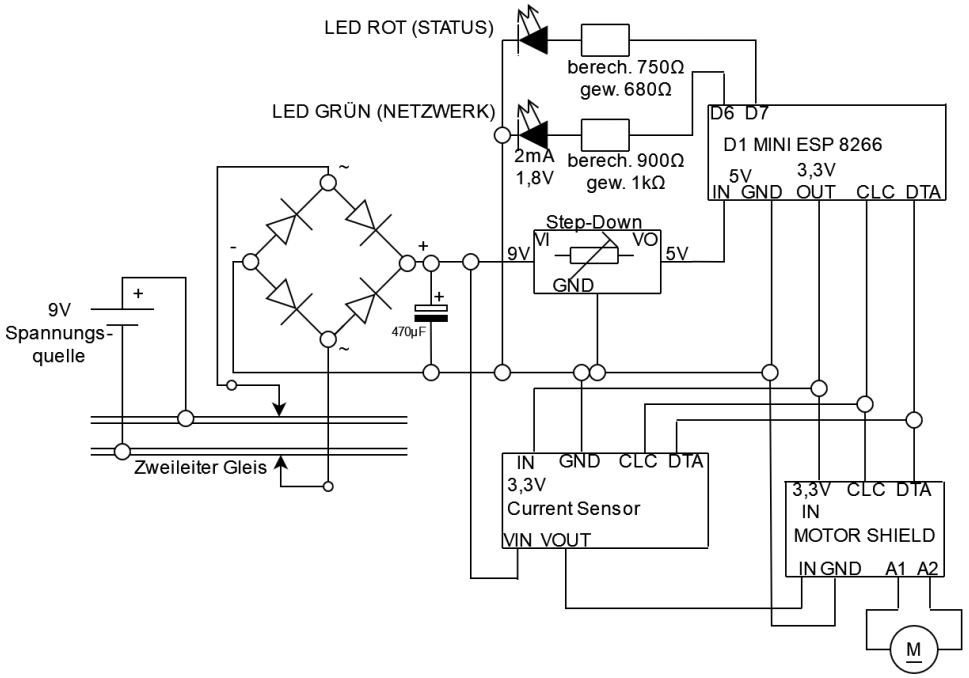

Abbildung 3.5: Schaltplan des Loksystems nach dem Hinzufügen der zusätzlichen Bauteile

In Abbildung 3.5 ist der Schaltplan des Gesamtsystems dargestellt, nachdem alle zuvor beschriebenen Komponenten eingebaut sind. Am linken Rand der Abbildung befindet sich die 9V Spannungsquelle, die im Gegensatz zum ursprünglichen Gleissystem direkt an das Gleis angeschlossen ist. Darauf folgt das Gleis und die Schleifkontakte des Motors durch die Räder. Alles nun im Schaltplan folgende befindet sich in der Lokomotive. Als nächstes folgen der Gleichrichter und der Kondensator, die vor der gesamten Steuerungselektronik platziert sind. Im Zentrum befindet sich der Step-Down, der die 5V Betriebsspannung für den D1 Mini ESP 8266 bereitstellt, der sich mit den Status LEDs im rechten oberen Teil der Abbildung befindet. Zuletzt ist die Mess- und Steuerelektronik - inklusive des Motors selbst - im rechten unteren Teil zu sehen. Abgesehen vom Motor werden diese mit einer Betriebspannung von 3.3V betrieben, die von dem D1 Mini selbst bereitgestellt wird.

4 Software

Dieses Kapitel beschäftigt sich mit der Umsetzung in der Software. Dazu wird zunächst auf den Aufbau der UDP-Pakete eingegangen, die der gesamten Kommunikation zu Grunde liegen, und die darin enthaltenen Header erläutert. Ebenso wird der Kommunikationsablauf des gesamten Systems, von Clientanwendung über API bis zu der Lokomotive, beschrieben. Abschließend werden die zusätzlichen Funktionen der Lokomotive, wie die untere Geschwindigkeitsschwelle und die Simulation realistischen Fahrverhaltens erläutert.

Implementiert wurde die gesamte Software, einschließlich Konsoleninterface und API, mit Hilfe der *Visual Studio Code*[1] IDE, wobei die Software der Lokomotive mithilfe der Erweiterung *PlatformIO* [2], aufgrund der Verfügbarkeit der Arduino kompatiblen Softwaremodule, entwickelt wurde. Das Konsoleninterface und die API wurden in *Python* und die Software der Lokomotive in *C++* programmiert.

4.1 UDP-Paketstruktur

Alle gesendeten Pakete bestehen zunächst aus dem üblichen IPv4-Header und dem darauf folgenden UDP-Header. Im folgenden wird daher nur das Datensegment der entsprechenden Pakete weiter betrachtet. Grundlegend haben die Datensegmente aller Pakete die selbe Struktur und sind wie eigenständige Pakete aufgebaut.

START	TRAIN ID	LENGTH	HEADER	DATA	XOR	END
1 Byte	1 Byte	1 Byte	1 Byte	n Byte(s)	1 Byte	1 Byte

Abbildung 4.1: Allgemeine Struktur des Datensegments in einem UDP Paket

Wie in Abbildung 4.1 zu sehen ist, beginnt jedes Datensegment mit einem Startbyte, welches mit dem Wert *0xAF* den Beginn eines neuen Datensegments anzeigt.

Mit der ID der Lokomotive (TRAIN ID) folgt nun ein Byte, dessen Präsens abhängig von der Senderichtung ist. Wird das Paket von der Lokomotive zu einem Client gesendet, ist dieses Byte immer vorhanden und beinhaltet die ID-Nummer der Lokomotive, die das Paket gesendet hat. Wird hingegen von einem Client zu der Lokomotive gesendet, ist dieses Byte nicht vorhanden. An dessen Stelle folgt dann sofort das Längenbyte. Dieses gibt die Länge des für die Befehlsinterpretation relevanten Bereichs des Datensegments

[1] `https://code.visualstudio.com/` (Abgerufen: 30.01.2023)
[2] `https://platformio.org/` (Abgerufen: 30.01.2023)

4.1.3 Richtung und Geschwindigkeit

Die Richtung und Geschwindigkeit einer Lokomotive werden, unabhängig von der Senderichtung, immer zusammen in einem Paket angegeben. Die gültigen Werte für die Richtung der Lokomotive sind entweder *0x0F* für Fahrtrichtung vorwärts oder *0x0B* für Fahrtrichtung rückwärts.

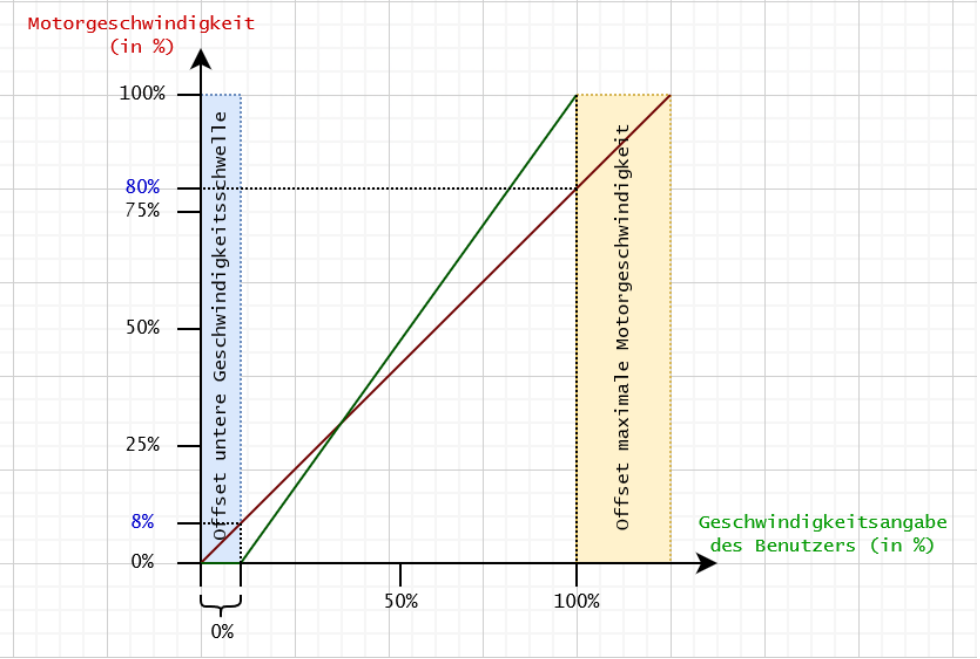

Abbildung 4.4: Geschwindigkeitsmapping und Offsets

Der Motor kann absolute Geschwindigkeiten von 0.00% bis 100.00% (inklusive) fahren und der Benutzer relative Geschwindigkeiten zwischen 0.00% und 100.00% angeben. In welchem Bereich die vom Benutzer wählbare relative Geschwindigkeit liegt, wird im unteren Bereich durch die untere Geschwindigkeisschwelle und im oberen Bereich durch die eingestellte maximale Motorgeschwindigkeit festgelegt. In Kapitel 4.3 wird die untere Geschwindigkeitsschwelle im Detail beschrieben. In Abbildung 4.4 ist das Verhältnis der absoluten Geschwindigkeit des Motors und der vom Benutzer angegebenen relativen Geschwindigkeit in einem Graphen dargestellt. Die relative Geschwindigkeitsangabe von 100.00% durch den Benutzer, entspricht so einer absolut gefahrenen Geschwindigkeit von 80.00%, wenn die Maximalgeschwindigkeit des Motors mit 80.00% eingestellt ist. Gibt der Benutzer die kleinstmögliche relative Geschwindigkeit an, die aber nicht 0 ist, so entspricht dies der absoluten Geschwindigkeit, wie sie von der unteren Geschwindigkeitsschwelle vorgegeben ist. In Abbildung 4.4 liegt diese bei 8.00% absoluter Geschwindigkeit, welche der kleinstmöglichen relativen Geschwindigkeit - aber größer als '0' - entspricht. Damit die Lokomotive bei einer relativen Geschwindigkeitsangabe von 0.00% auch still steht, findet ein Sprung auf absolute 0.00% des Motors statt. Bei der Angabe der relativen Geschwindigkeit ist zu beachten, dass diese immer in zwei Bytes getrennt ist. Im ersten Byte wird der Ganzzahlwert und im zweiten Byte der Kommawert der relativen Geschwindigkeit an-

4.1.7 Lokomotivname und -ID

Jeder Lokomotive kann ein Name und eine ID zugewiesen werden. Der Name muss aus Zeichen im ASCII-Format bestehen und darf maximal 32 Zeichen lang sein. Die ID der Lokomotive muss eine positive Ganzzahl zwischen 0 und 255 (beides inklusive) sein.

Ändern des Namens Der Name der Lokomotive wird durch die Angabe des Headers *0x1E* geändert, wie in 4.10 (A) zu sehen ist. Auf den Header folgen dann die einzelnen Bytes der Zeichen. Die Anzahl der Zeichenbytes darf 32 dabei nicht überschreiten. Die Lokomotive überprüft *nicht*, ob es sich bei jedem einzelnen empfangenen Zeichen tatsächlich um ein gültiges Zeichen im ASCII-Format handelt. Die Werte der Zeichen werden den entsprechenden Bytes entnommen und unverändert im EEPROM gespeichert. Bei einer Anfrage des Namens werden die Werte der Zeichen unverändert übermittelt. Außerdem muss darauf geachtet werden, dass die maximale Anzahl von Zeichen nicht überschritten wird. Ist dennoch mindestens ein Zeichen zu viel in dem Paket, wird die Lokomotive dieses nicht abweisen, sondern den Namen bis zu der maximalen Zeichenanzahl speichern. Alle darüber hinaus gehenden Zeichen werden verworfen. Es wird daher empfohlen nach der Änderung des Namens, den Namen der Lokomotive anzufragen und mit dem gesendeten Namen zu vergleichen.

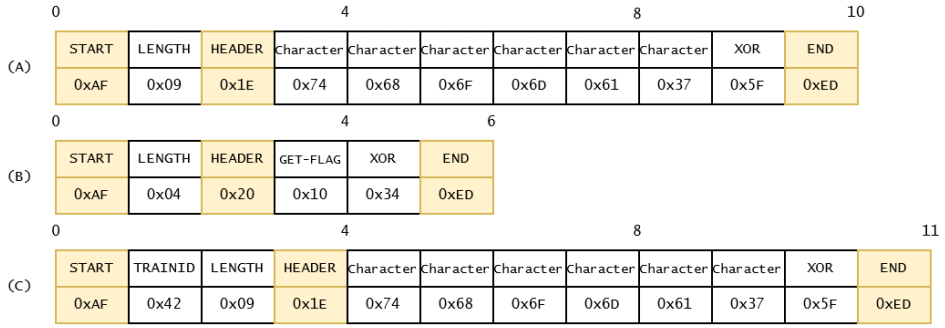

Abbildung 4.10: Beispielpaket für das Ändern des Lokomotivenamens (A), Anfragen des aktuellen Lokomotivnamens (B) und die Antwort der Lokomotive mit ihrem Namen

Anfragen des aktuellen Namens Der Name der Lokomotive kann durch den *GetData*-Header *0x20* angefragt werden, mit der entsprechenden *GET-FLAG* von *0x10*, wie in Abbildung 4.10 (B) dargestellt ist. Die Lokomotive wird dieses Paket dann nach Empfang durch ein ACK oder NACK quittieren. Wenn mit einem ACK quittiert wurde, folgt nun ein weiteres Paket mit dem Namen der Lokomotive. Dieses enthält den Lokomotivnamen-Header 0x1E und darauf folgend maximal 32 Zeichenbytes. Jedes Byte beinhaltet genau ein Zeichen im ASCII-Format, wobei von der Lokomotive nicht sichergestellt wurde, dass es sich um ein korrektes Zeichen im ASCII-Format handelt. Die Interpretation der Bytewerte zu ASCII-Zeichen obliegt dem Client. Ein Beispielpaket mit dem Namen *thomas* ist in Abbildung 4.10 (C) abgebildet.

Darauf folgen von Index 23 bis Index 30 jeweils vier Bytes für die Client IPv4 Adresse und den Client Port. Hier werden die Verbindungsangaben des zuletzt verbundenen Clients gespeichert, um sich nach einem Neustart sofort wieder bei diesem anzumelden. Da weder API noch das Konsoleninterface diese Funktion unterstützen ist lediglich der Speicherplatz im EEPROM dafür reserviert. Gespeichert oder geladen werden die Daten jedoch nicht, was aber bei einer Weiterentwicklung der Software der Lokomotive für zukünftige Funktionen leicht aufgegriffen werden kann. Bei Index 31 befindet sich die vom Benutzer vergebene ID der Lokomotive und anschließend von Index 32 bis 63 der Name der Lokomotive mit 32 Bytes. Darauf folgen - ebenfalls mit jeweils 32 Bytes - die WiFi-ssid von Index 64 bis 95 und von Index 96 bis 127 das WiFi-Passwort. Das Ende einer Zeichenfolge wird im EEPROM durch den Wert 0xFF angezeigt, da der Wert 0xFF kein gültiges ASCII-Format Zeichen ist. Dies gilt nur, wenn nicht die volle Länge von 32 Bytes verwendet wird.

4.3 Untere Geschwindigkeitsschwelle

Bei den ersten Testfahrten der Lokomotive hat sich gezeigt, dass sich der Motor bei geringer Energieversorgung noch nicht bewegt, obwohl die Energieversorgung kontinuierlich erhöht wurde. Erst ab einem bestimmten Punkt während der Erhöhung der Energieversorgung, konnte der Motor die Lokomotive in Bewegung versetzen.

Je nach Testaufbau mit größerer oder kleinerer angehängter Last an der Lokomotive, verschiebt sich das Anfahrmoment. Wird der Motor von den Gleisen abgehoben, wodurch kein Gleiskontakt mehr besteht, lag dieser Punkt bei 5% der absoluten Motorgeschwindigkeit und etwa 7mA Stromaufnahme. Die Lokomotive wurde in diesem Fall über einen Kabelanschluss direkt mit Spannung versorgt. Bei einem anderen Testaufbau, wo die Lokomotive Gleiskontakt hat und sich selbst über das Gleissystem bewegen kann, befand sich der Punkt, ab dem sich der Motor zu bewegen begann, bei 8% der absoluten Motorgeschwindigkeit und 11mA. Danach wurde dieser Aufbau wiederholt, wobei ein zusätzlicher Waggon an die Lokomotive angehängt wurde. Die Erwartung in diesem Fall war, dass sich der Motor erst ab einem noch späteren Zeitpunkt, als bei dem zweiten Testaufbau, zu bewegen beginnt und ebenfalls bei noch höherer Stromaufnahme. Diese Erwartung hat sich nach mindestens fünf Testfahrten pro Testaufbau bestätigt. Der ermittelte Startwert lag im Durchschnitt bei 15% der absoluten Motorgeschwindigkeit bei einer Stromaufnahme von 22mA.

Es hat sich gezeigt, dass in dem unteren Bereich der PWM Steuerung des Motors eine Schwelle liegt, ab der sich der Motor beginnt zu bewegen. Dieser Punkt wird als untere Geschwindigkeitsschwelle UGS bezeichnet. Anhand der zuvor durchgeführten Testfahrten mit unterschiedlichen Testaufbauten, konnte die Annahme bestätigt werden, dass mit steigender Last (in Waggons) an der Lokomotive und dem Hinzufügen von Reibung an den Rädern durch Kontakt mit dem Gleis, eine höhere Energieversorgung erforderlich ist, um den Motor und somit auch die Lokomotive in Bewegung zu versetzen.

4.4 Simulation realistischen Fahrverhaltens

Zu Beginn dieser Arbeit wurde auch das Ziel der Entwicklung eines realistischen Fahrverhaltens gesetzt, dessen Umsetzung im folgenden Abschnitt erläutert wird.

4.4.1 Beschleunigungs- und Bremssimulation

Um das Fahrverhalten realistischer zu gestalten, wurde die Geschwindigkeitssteuerung so implementiert, dass ein langsames Beschleunigen und Bremsen ermöglicht wird. Zusätzlich wird der abrupte Wechsel der Fahrtrichtung während der Fahrt verhindert. Dabei wird zunächst bis zum Stillstand der Lokomotive gebremst, dann erst die Richtung gewechselt und darauf folgend bis zu der Zielgeschwindigkeit beschleunigt, die zuvor angegeben wurde.

Umsetzung Bei der Beschleunigungs- und Bremssimulation werden zunächst zwei Variablen betrachtet: Die aktuelle Motorgeschwindigkeit und die angestrebte Motorgeschwindigkeit. Nur wenn diese voneinander abweichen, muss die Geschwindigkeit angeglichen werden. Dazu ist die Geschwindigkeitskontrolle in die drei Bereiche *Verlangsamen*, *Beschleunigen* und *Richtungswechsel* unterteilt. Zunächst wird auf eine Verlangsamung überprüft, da dies eine Voraussetung für den Richtungswechsel ist. Danach wird überprüft ob beschleunigt werden muss und zuletzt, ob die Richtung gewechselt werden muss und auch kann.

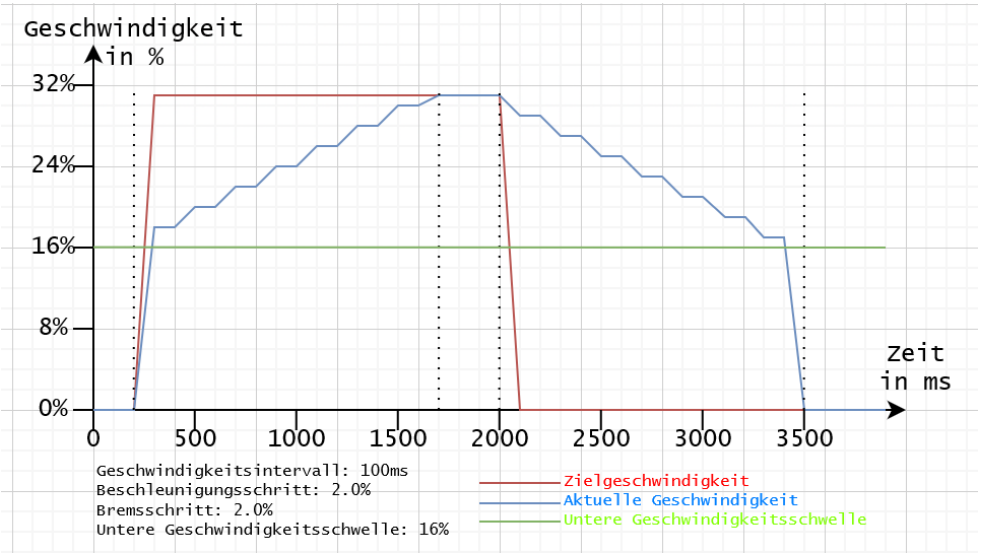

Abbildung 4.16: Beschleunigungs- und Bremsverhalten

Wenn während der schrittweisen Änderung der Geschwindigkeit die Zielgeschwindigkeit erneut verändert wird, wird auch sofort zu der neuen Zielgeschwindigkeit gebremst oder beschleunigt. Der Richtungswechsel kann nur durchgeführt werden, wenn zum einen die aktuelle Richtung von der Zielrichtung abweicht (ansonsten würde kein Richtungswechsel vorliegen) und zum anderen die Lokomotive still steht, wie in Codeauszug 4.1 Zeile 2 zu sehen ist. Würde sie dies bei dem Wechsel der Richtung nicht, dann würde sie während der Fahrt ruckartig in die andere Richtung wechseln und das Ziel eines realistischen Fahrbetriebs wäre verfehlt. Damit nach dem Richtungswechsel auch sofort die neue Zielgeschwindigkeit gesetzt werden kann, wird diese - vor dem Bremsbeginn zum Richtungswechsel - in einer zusätzlichen Variablen (*newDirSpeed*) zwischengespeichert.

4.4.2 Geschwindigkeitsausgleich in Kurven

Eine weitere Funktion, die das realistische Fahrverhalten der Lokomotive unterstützen soll, ist das Halten der Geschwindigkeit auf einem konstanten Niveau, besonders in kurvigen Gleisabschnitten. Um dies zu realisieren wurde in dieser Arbeit ein Verfahren entwickelt, diese Gleisabschnitte zu erkennen und entsprechend darauf zu reagieren, damit die Geschwindigkeit konstant bleibt.

Hypothese Zu Beginn dieser Arbeit wurde die Hypothese aufgestellt, dass die Geschwindigkeit der Lokomotive reduziert wird, wenn diese in einen kurvigen Streckenabschnitt einfährt, weil sich die Reibung von Rad und Schiene erhöht hat. Um diese Hypothese zu untersuchen, wurde der Motorstrom bei einer konstanten Geschwindigkeit gemessen.

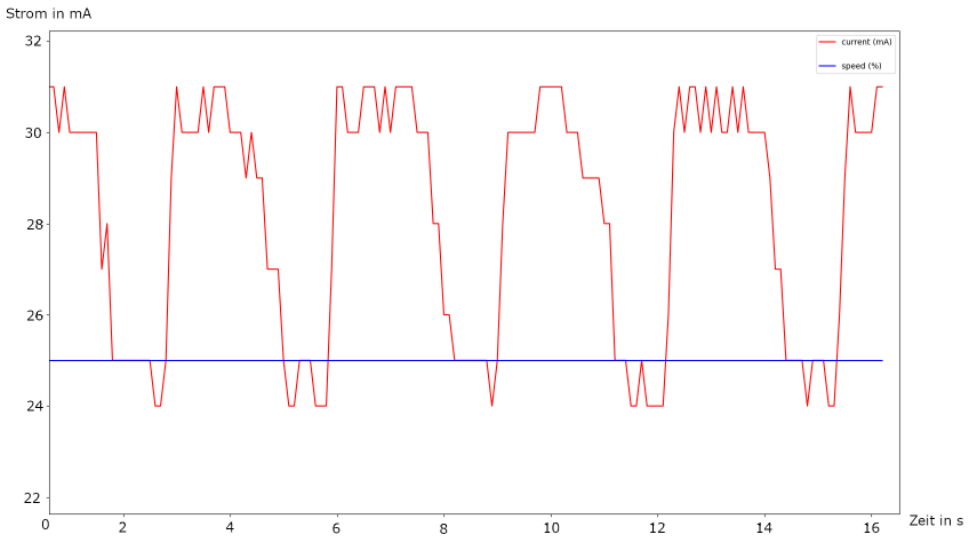

Abbildung 4.17: Strommessungen bei dem Befahren von kurvigen und geraden Gleisabschnitten

4.5 API und Konsolensteuerung (TCS)

In diesem Unterkapitel wird zunächst der Aufbau der Befehle, die zu der API gesendet werden, erläutert. Daraufhin wird die Struktur der möglichen Antworten beschrieben und abschließend die in der Arbeit entwickelte Konsolensteuerung dargestellt. Bei der in der Arbeit entwickelten API handelt es sich um eine RESTful-API basierend auf einem *FLASK*[3] Webserver, geschrieben in *Python*. Nachdem die API gestartet ist, nimmt diese bei dem Endpunkt */api* bei Port 10200 Befehle im JSON-Format entgegen.

4.5.1 Befehle zur API

Alle Befehle, die an die API gesendet werden, sind in ihrer Struktur ähnlich aufgebaut. So enthält jeder Befehl ein Attribut, dass den Befehl selbst beschreibt. Wenn sich ein Befehl an eine Lokomotive richtet, ist dazu noch die IPv4 der Lokomotive als Attribut angegeben, da die ID oder der Name der Lokomotive nicht eindeutig sein müssen. Zuletzt enthält ein Befehl ein Daten-Attribute, dass weitere Informationen zu dem Befehl beinhaltet. Dazu gehört beispielsweise bei einer Änderung einer Variablen, der Name dieser Variablen und der neue Wert. Ein Befehl mit allen zusätzlichen Attributen ist in Codeauszug 4.3 dargestellt. Weitergehend können die Befehle in die vier Kategorien *Fahren, Anfragen, Ändern* und *Diverse* unterteilt werden. Auf jeden Befehl, unabhängig der Kategorie, folgt eine Antwort, die den Erfolgsstatus mitteilt. Befehle der Kategorie *Anfragen* enthalten in der Antwort zusätzliche Daten, die durch den Befehl bei der Lokomotive angefragt wurden.

```
{
    "command": "setData",
    "trainAddress": "192.168.0.236",
    "data": {
        "type": "trainId",
        "trainId": 42
    }
},
```

Listing 4.3: Zu der API gesendeter Befehl im JSON-Format zur Änderung der LokomotivID

Fahren Es gibt zwei Befehle, die die Fahrt der Lokomotive direkt beeinflussen. Mit dem Befehl *stop* wird der Nothalt der Lokomotive ausgelöst. Dies hält die Lokomotive sofort an, blockiert jedoch nicht das Weiterfahren. Außerdem muss zusätzlich die Adresse der Lokomotive angegeben werden, bei der der Nothalt ausgelöst werden soll.

[3] `https://pypi.org/project/Flask/` (Abgerufen: 01.02.2023)

Diverses In dieser Kategorie sind die Befehle enthalten, die keine Adresse für eine Lokomotive enthalten und / oder kein Datenattribut. Zu Letzterem gehören die Befehle *restoreDefault* und *threshold*, die jeweils nur aus dem Befehl selbst und der Adresse der Lokomotive bestehen. Der Befehl *restoreDefault* stell die Standardwerte der Konfigurationsvariablen wieder her und *threshold* startet die Ermittlung der UGS, weshalb diese kein zusätzliches Daten-Attribut benötigen.

Die Befehle *helloThere* und *discovery* bestehen ausschließlich aus dem Befehl selbst und haben keine weiteren Attribute. Der Befehl *helloThere* wird verwendet, um zu überprüfen, ob sich an der Adresse eine API befindet. Es wird dabei lediglich das Gegenzeichen *generalKenobi* zurück gesendet und es findet keine Kommunikation mit einer Lokomotive statt. Bei dem Befehl *discovery* werden allen Adressen im näheren Netzwerk das *Discovery*-Paket gesendet. Wenn eine Adresse zu einer Lokomotive gehört, wird diese darauf antworten und daraufhin zu einer Liste hinzugefügt. Wenn der Prozess abgeschlossen ist, sendet die API eine Antwort mit der Liste aller gefundenen Lokomotiven zurück.

4.5.2 Antworten der API

Die Antworten der API sind grundlegend ähnlich und unterscheiden sich ausschließlich in dem Objekt des Daten-Attributes. Wie in Codeauszug 4.7 zu sehen ist, hat jede Antwort einen *status*. Dieser gibt an ob der zuvor gesendete Befehl durch ein ACK angenommen wurde '1' oder durch ein NACK von der Lokomotive abgelehnt wurde '0'.

```
{
    "status": 0,
    "data": {
        "type": "nack",
        "message": "Error message"
    }
},
{
    "status": 1,
    "data": {
        "type": "ack",
        "trainAddress": "192.168.0.236"
    }
},
{
    "status": 1,
    "data": {
        "type": "speedValue",
        "trainAddress": "192.168.0.236",
        "direction": 1,
        "speedValue": 16
    }
},
```

Listing 4.7: Beispielantworten der API

6 Zusammenfassung

Diese Arbeit sollte ein bestehendes, veraltetes Lok- und Gleissystem eines Klemmbausteinherstellers mit intelligenten Funktionen erweitern und einen Betrieb mit mehr als einer Lokomotive auf dem selben Gleissystem ermöglichen. Des weiteren sollte die Steuerung der Lokomotive so vorbereitet werden, dass die Steuerung einer anderen bestehenden Modelleisenbahnanlage mittels Transfer Learning auf dieses System übertragen werden kann.

Um dies umzusetzen, wurde die Lokomotive mit einem eigenen Prozessor ausgestattet der nun die Steuerung übernimmt und Sensortechnik hinzugefügt, damit die Steuerung der Lokomotive auf Veränderungen reagieren kann. Außerdem wurde die Steuerungsform des Motors von einer Spannungsregelung zu einer PWM geändert, um die Integration eines Steuermoduls zu ermöglichen.

Die Lokomotive verfügt nun über Funktionen, die die Anfahrschwelle erkennen und anderen Funktionen bereitstellen, sowie einen realistischen Beschleunigungs- und Bremsvorgang simulieren können. Außerdem wird das Einfahren in kurvige Gleisabschnitte und Steigungen erkannt und bei dem Durchfahren die Geschwindigkeit annähernd konstant gehalten. Außerdem wurden der Lokomotive einige Variablen hinzugefügt, die während des Betriebs von außen konfiguriert werden können und die Lokomotive je nach Anwendung sehr flexibel machen.

Zur Steuerung der Lokomotive von einer anderen Software aus, wurde neben der Möglichkeit die Lokomotive direkt durch UDP Pakete zu steuern, auch eine API entwickelt, die über das Netzwerk entsprechende JSON formatierte Zeichenketten annimmt und diese als UDP Pakete an die entsprechende Lokomotive sendet. Rückmeldungen von einer Lokomotive werden auf diesem Weg ebenfalls ermöglicht. Durch diese API kann eine Software Befehle an das gesamte System senden und auf Rückmeldungen reagieren.

Für die Steuerung der Lokomotive ohne eine eigene Software, wurde ein Konsoleninterface entwickelt, das durch die Systemkonsole Textbefehle des Benutzers entgegen nimmt und diese mittels der API an die entsprechende Lokomotive sendet. Die Konfiguration der Lokomotive und das Anfragen von Informationen wird ebenfalls ermöglicht.

Somit wurde mit dieser Arbeit ein veraltetes Lokomotiv- und Gleissystem aufgewertet und wiederverwendbar gemacht, sowohl für den ursprünglichen Verwendungszweck als Spielzeug, als auch für eine Anwendung im Bereich der Forschung mit anderen Technologien.

Literaturverzeichnis

[Fielding 2000] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*, UNIVERSITY OF CALIFORNIA, IRVINE, Diss., 2000. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. – 76–106 S.

[K. G . Bush 1998] K. G . Bush: *Regelbare Elektroantriebe: Antriebsmethoden, Betriebssicherheit, Instandhaltung.* München : Richard Pflaum Verlag, 1998. – 80–81 S. – ISBN 3–7905–0754–7

[Lee u. a. 2020] In: Lee, J. ; Azamfar, M. ; Singh, J. ; Siahpour, S: *Integration of digital twin and deep learning in cyber-physical systems: towards smart manufacturing.* The Institution of Engineering and Technology, 2020

[Maschler u. a. 2021] Maschler, Benjamin ; Braun, Dominik ; Jazdi, Nasser ; Weyrich, Michael: Transfer learning as an enabler of the intelligent digital twin. In: *Procedia CIRP* 100 (2021), 127-132. http://dx.doi.org/https://doi.org/10.1016/j.procir.2021.05.020. – DOI https://doi.org/10.1016/j.procir.2021.05.020. – ISSN 2212–8271. – 31st CIRP Design Conference 2021 (CIRP Design 2021)

[Postel 1980] Postel, J.: User Datagram Protocol, RFC Editor, August 1980 (768)

[R. Fielding, et. al. 1999] R. Fielding, et. al.: Hypertext Transfer Protocol – HTTP/1.1 / RFC Editor. Version: June 1999. https://www.rfc-editor.org/rfc/rfc6690. RFC Editor, June 1999 (2616). – RFC. – 53–71 S.

[Stoll 1988] In: Stoll, Dieter: *Gleichrichter.* Wiesbaden : Vieweg+Teubner Verlag, 1988. – ISBN 978–3–663–00130–0, 77–84

[Ursu u. Condruz 2017] Ursu, M P. ; Condruz, D A.: Digital intelligent booster for DCC miniature train networks. In: *IOP Conference Series: Materials Science and Engineering* 227 (2017), aug, Nr. 1, 012133. http://dx.doi.org/10.1088/1757-899X/227/1/012133. – DOI 10.1088/1757–899X/227/1/012133

[Wegener 2016] In: Wegener, Charlotte: *Upcycling.* London : Palgrave Macmillan UK, 2016. – ISBN 978–1–137–51180–5, 181–188

[Zhao 2022] Zhao, Yisong: A digital command control protocol designed on an FPGA to control a standard railway. In: Subramaniyam, Kannimuthu (Hrsg.) ; International Society for Optics and Photonics (Veranst.): *Second International Conference on Advanced Algorithms and Signal Image Processing (AASIP 2022)* Bd. 12475 International Society for Optics and Photonics, SPIE, 2022