

Schlussdokumentation PREN2

Gruppe 28

Andreas Rebsamen (Elektrotechnik)

Joel Grepper (Informatik)

Manuel Omlin (Maschinentechnik)

Marco Schöni (Maschinentechnik)

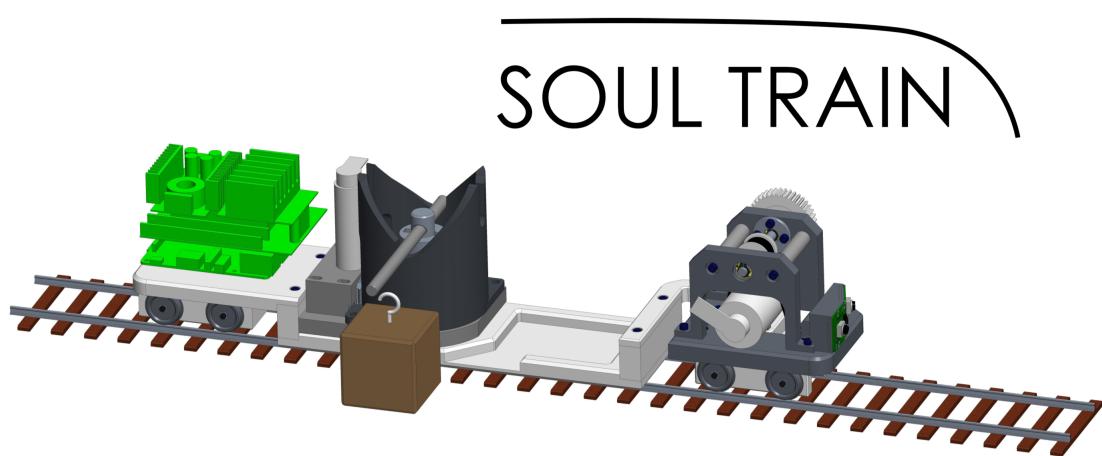
Patrick Marty (Informatik)

Steve Ineichen (Informatik)

Hochgeschwindigkeitsschienenfahrzeug "SOUL TRAIN"

Hochschule Luzern – Technik & Architektur

TA.BA_PREN2.F1901



Horw, Hochschule Luzern – Technik & Architektur, 06.06.2019

Redlichkeitserklärung

Die Verfasser bestätigen mit ihrer Unterschrift, dass die vorliegende Arbeit selbstständig, ohne fremde Hilfe und ohne Benutzung anderer als die angegebenen Hilfsmittel angefertigt worden ist.

Die aus fremden Quellen (einschliesslich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit ist in gleicher oder ähnlicher Form noch nicht vorgelegt worden.

Eingereicht in Horw, am 10. Juni 2019



Andreas Rebsamen



Joel Grepper



Manuel Omlin



Marco Schöni



Patrick Marty



Steve Ineichen

Inhaltsverzeichnis

1 Management Summary	3
2 Einleitung	4
3 Produktbeschrieb	5
3.1 Würfelaufnahme	6
3.2 Motorauslegung	9
3.3 Fahrwerk	10
3.4 Elektrische Verbindungen	16
3.5 Elektronik Komponenten	18
3.6 Software TinyK22	22
3.7 Aufgabentrennung zwischen Pi und Tiny	30
3.8 Interface zwischen Pi und Tiny	31
3.9 Steuerungssoftware	34
3.10 Beschleunigung	39
3.11 Akustik	41
3.12 Webapplikation	43
3.13 Tafel- und Nummererkennung	46
4 Bewertung	53
4.1 Elektronik	53
4.2 Informatik	54
4.3 Maschinentechnik	55
5 Bedienungsanleitung	58
5.1 Inbetriebnahme Mechanik	58
5.2 Bedienungsanleitung Elektronik und WebApp	59
6 Schlussdiskussion	66
6.1 Kosten	66
6.2 Zeitlicher Entwicklungsaufwand	67
6.3 Lessons learned	67
6.4 Offene Punkte	68
6.5 Ausblick	68
7 Verzeichnisse	69

1 Management Summary

Im Rahmen des Moduls PREN (Produktentwicklung) an der Hochschule Luzern wurde dieses Dokument durch das Team 28 erstellt. Dieses interdisziplinäre Team, bestehend aus Maschinenbau-, Elektrotechnik-, und Informatikstudenten, erhielt die Aufgabenstellung einen Schnellzug zu entwickeln. Dieser Zug muss eine definierte Strecke mit Geraden und Kurven in zwei Runden so schnell wie möglich passieren. Dabei muss der Zug eine Fracht, welche in der Startzone mittels eines Krans aufgeladen wird, während der gesamten Strecke transportieren. Während die Strecke zurückgelegt wird, muss der Zug eine Signaltafel mit Nummer erkennen und in der dritten Runde bei dieser Nummer so genau wie möglich anhalten. Das Konzept des Zuges wurde im Vorgängermodul PREN1 entwickelt. Mit Hilfe von Technologierecherchen, Varianten und ersten Machbarkeitsanalyse konnten mehrere Lösungsansätze in PREN1 entwickelt werden. Durch das Zusammenführen der optimalen Lösungsvarianten konnte schlussendlich das finale Lösungskonzept erstellt werden. Im PREN2 wurde nun dieses Lösungskonzept als Prototyp *umgesetzt*. Der Prototyp beinhaltet eine mechanische Grundkonstruktion eines Zuges, Elektronik für die Aktoren und Sensoren, so wie Software für den Ablauf und das Zusammenspiel der einzelnen Komponenten. Nach der Umsetzung wurden umfassende *Tests*, welche sorgfältig geplant worden sind, durchgeführt. Diese stellten sicher, dass der Zug die geforderte Zuverlässigkeit erreicht. Fortlaufend wurden Erkenntnisse aus den Tests als *Optimierungsvorschläge* zurück in die Umsetzung eingespielen. Dies führte zu einer stetigen Verbesserung des Zuges, so dass am Schluss des PREN2 ein funktionierender Prototyp bereitgestellt werden konnte.

2 Einleitung

In dieser Arbeit liegt das Gesamtkonzept für einen autonomen Schnellzug vor. Dieses Konzept wurde im HS18 im Rahmen des PREN1 entwickelt und wird nun FS19 im Modul PREN2 realisiert. Die Aufgabe besteht darin, einen Schnellzug zu entwickeln und zu bauen, welcher eine definierte Strecke mit Geraden und Kurven so schnell wie möglich zurücklegt. Im Startbereich muss mittels einer am Zug befestigten Konstruktion ein Holzwürfel auf den Zug gehoben und transportiert werden. Danach muss der Zug eine Lichtschranke passieren und zwei Runden auf der Strecke so schnell wie möglich absolvieren. Dabei muss er ein seitlich befindendes Signal mit Nummer erkennen, welches die Halteposition signalisiert. Diese Halteposition muss in der dritten Runde angefahren werden, und der Zug soll dort so nahe wie möglich anhalten. Die erkannte Nummer soll auch mittels akustischem Signal ausgegeben werden.

Der Zug wurde mit folgenden Schwerpunkten entwickelt:

- Kompaktheit
- Einfachheit
- niedriges Gewicht
- Robustheit
- niedrige Kosten

Im Hauptteil dieser Arbeit wird das finale Konzept des Zuges beschrieben. Die Beschreibung ist aufgeteilt in die drei Fachgebiete Maschinentechnik, Elektrotechnik und Informatik. Im Maschinentechnik-Bereich wird die mechanische Grundkonstruktion des Zuges und des Krans für den Holzwürfel erläutert. Der Antrieb, die Sensorik und die Stromversorgung des Zuges wird im Elektrotechnikteil beschrieben. Im Abschnitt Informatik wird die Signalerkennung, die akustische Ausgabe und der Softwareaufbau beschrieben. Eine Bedienungsanleitung, Kostenübersicht und Schlussdiskussion sind im hinteren Teil der Arbeit zu finden.

In diesem Konzept wurden bei der Entwicklung die Schwerpunkte Kompaktheit, Einfachheit und niedriges Gewicht berücksichtigt, um eine optimale maximale Geschwindigkeit mit dem Zug zu erreichen. Dabei soll aber auch ein Schwergewicht auf Robustheit und Prozesssicherheit gelegt werden. Das Konzept aus PREN1 wurde annähernd komplett übernommen und mit kleineren Änderungen optimiert.

3 Produktbeschrieb

Diese Kapitel beschreibt das Funktionsprinzip des Zuges. Als erstes werden die Mechanischen Komponenten beschrieben. Kapitel 3.1 zeigt das Funktionsprinzip der Würfelaufnahmen, Kapitel 3.2 und 3.3 beinhaltet Angaben zum Fahrwerk und dem Antrieb. Danach wird in Kapitel 3.4 bis 3.5.2 die Hardware der Elektronik beschrieben. Anschliessend beschreibt das Kapitel 3.6 die Software die auf dem Mikrocontroller läuft und die Hardware ansteuert.

In Kapitel 3.7 und 3.8 wird die Zusammenarbeit und Kommunikation zwischen dem Mikrocontroller und dem Raspberry Pi beschrieben. Zum Schluss wird die Software und die dazugehörigen Subsysteme auf dem Raspberry Pi ab Kapitel 3.9 beschrieben.

Abbildung 1 zeigt eine allgemeine Übersicht der einzelnen Komponenten und deren Verbindungen. Genauere Angaben zu den Komponenten und den Verbindungen werden in den Folgenden Kapitel gemacht.

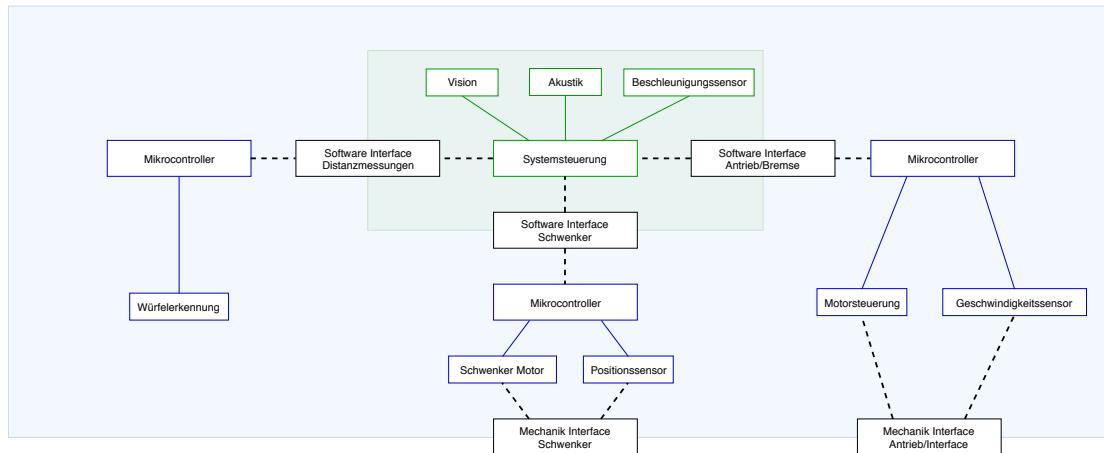


Abbildung 1: Komponentendiagramm allgemein

3.1 Würfelaufnahme

Um den Würfel rechts neben der Gleisstrecke aufzunehmen, wird eine einfache Lösung umgesetzt, steuerungstechnisch sowie mechanisch. Die Würfelaufnahme wird mit einem Draht und einem Stab durchgeführt. Damit nur ein Aktor angesteuert werden muss, wird von dem Prinzip einer Kurvenscheibe Gebrauch gemacht. Die gesamte Vorrichtung besteht grundsätzlich aus drei Elementen: Einem Kran zur Lastaufnahme, einem Antriebsstrang und der Kurvenscheibe. Wesentlichster Unterschied zum Grundkonzept aus PREN1 ist die Einführung bis zum Anschlag damit der Würfel sich nicht drehen kann während der Lastaufnahme. Unten auf der Abbildung ist die Einführung dargestellt. Weiter wurde eine zweite Revision der Kurvenscheibe gedruckt mit einigen Optimierungen. Hauptgrund war aber dass die Drehbewegung des Krans nun über 90 Grad ist. Das verschafft Gewissheit, das der Würfel an den Anschlag gedrückt wird und am für ihn bestimmten Ort zu liegen kommt.

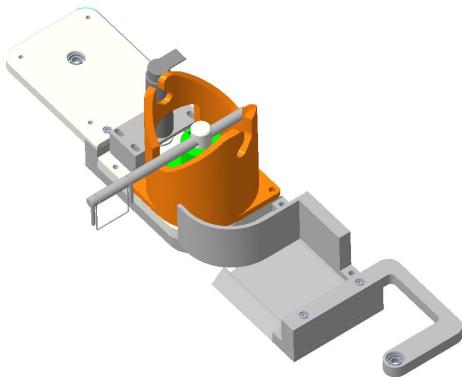


Abbildung 2: Baugruppe

3.1.1 Kran

Der Kran besteht aus drei Drehteilen, welche mit einer Pressverbindung zusammengefügt wurden. Das zentrale Element des Krans wird auf Grund seiner optimalen Gleiteigenschaften und der geringen Dichte aus Teflon gefertigt. Der kleinere Stahlstift ist für die Drehmomentübertragung zuständig. Der grössere der beiden Stahlstifte ist der eigentliche Ausleger. An dessen Ende wird ein Draht aus Federstahl geformt und angehängt. Dieser Draht soll als Haken zur Lastaufnahme dienen. Weiter ist der Ausleger in beide Richtungen von der Drehachse ausgedehnt, da die Kurvenscheibe zwei Laufflächen hat, um für einen stabilen Hub zu sorgen. Der ganze Aufbau wird mittels einer Spielpassung in einer Bohrung mit zwei Längsnuten in einem 3D gedruckten, modifizierten Zahnrad gelagert.

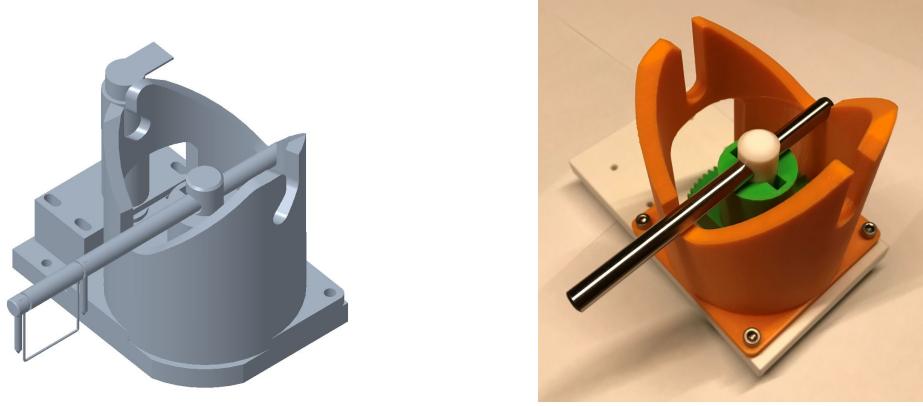


Abbildung 3: Links: Kran Planung / Rechts: Kran Umsetzung

3.1.2 Antriebsstrang

Der Antrieb besteht aus einem Motor, dessen Aufhängung und zwei Zahnrädern. Der Motor ist ein bürstenbehafteter Motor mit Encoder und Getriebe vorne drauf. Mit dieser Variante und der Übersetzung, zusammengesetzt aus Getriebe und Zahnrädern, kann von der Steuerung aus genau definiert werden, wie viele Umdrehungen der Motor benötigt, um mit dem Kranausleger eine Viertelumdrehung zu fahren. Das eine Zahnrad ist Standard und von Mädler eingekauft. Das zweite Zahnrad jedoch wurde nur als STEP von Mädler heruntergeladen und anschliessend im CAD bearbeitet. Die Bohrung und der Flansch in der Mitte wurden verlängert und mit zwei Längsnuten versehen. Die Bohrung gilt als Axialführung und die Nuten dienen als Drehmomentübertragung.

3.1.3 Kurvenscheibe

Die Kurvenscheibe ist ebenfalls ein 3D-Druckteil. Der Grundkörper ist ein Rohr mit dem Aussendurchmesser 80 mm. An diesem wurden zwei Bahnführungen mittels Freiformflächen für den Kranausleger erzeugt. Die Steigung dieser Flächen ist variabel. Zu Beginn ist die Steigung gering und wird dann exponentiell grösser. Dies wurde aus dem einen Grund gewählt, damit das Anfahren für den Motor nicht zu streng ist. Nachdem die Drehbewegung und der vertikale Hub gemacht wurden, stoppt der Motor und der Kranausleger sollte durch die Schwerkraft heruntergezogen werden. Der Würfel wird nun in der für ihn vorgesehenen Aufnahme auf dem Zug platziert.

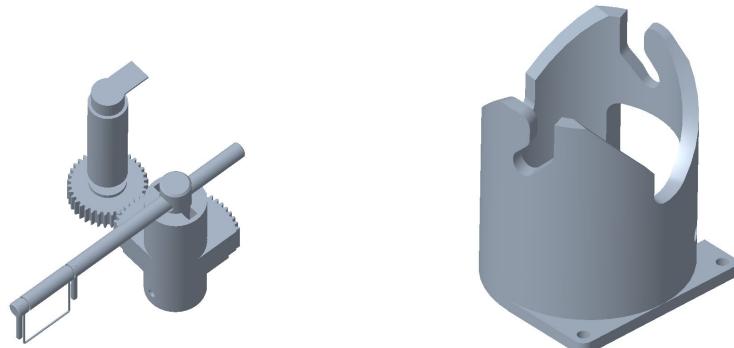


Abbildung 4: Links: Kran Antriebsstrang / Rechts: Kurvenscheibe

3.1.4 Testaufbau

Der Testaufbau besteht hauptsächlich aus 3D-Druckteilen und weichen Kunststoffen. Er dient momentan als Funktionsmuster. Wenn sich dieser weiter bewährt, möchte man mit den gefertigten Teilen weiterarbeiten. Der Test dient zur Probe des ausgewählten Lösungskonzepts der Würfelaufnahme. Erste Tests zeigen, dass die Funktion mit kleinen Anpassungen direkt umgesetzt werden kann.

3.1.5 Finale Umsetzung

Für die endgültige Lösung werden wie bereits erwähnt einige Anpassungen an der Kurvenscheibe und am Anschlag vorgenommen. Auf der Abbildung unten ist die endgültige Version der Würfelaufnahme zu sehen.

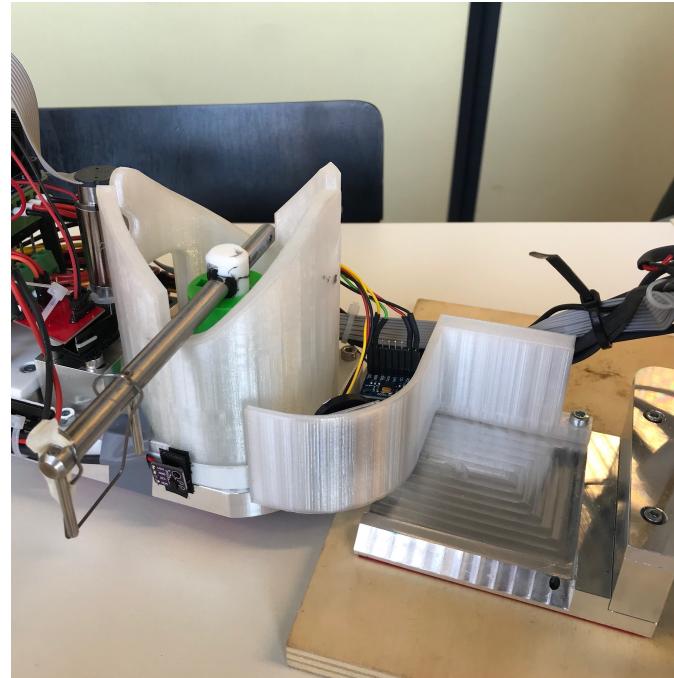


Abbildung 5: Würfelaufnahme

3.2 Motorauslegung

Um die Aufgabenstellung der schnellen Fahrt auf Schienen bestmöglich zu erfüllen, wird ein zuverlässiger starker Motor benötigt. Um einen solchen aus einer Vielzahl von Auswahlmöglichkeiten zu definieren, hat man sich auf den Katalog definiert maxon motor ag beschränkt. Im Anhang findet man ein Dokument mit den Berechnungen zur Motorauswahl. In Absprache der Disziplinen Elektrotechnik und Maschinentechnik wurde ein bürstenbehafteter Gleichstrommotor als geeignetes Modell definiert. Im vorher schon erwähnten Dokument wird für den gesamten Zug eine Masse von 3 Kilogramm gerechnet und einen Raddurchmesser von 22 mm. Für eine Endgeschwindigkeit von 3 m/s mit den definierten Raddurchmessern ergibt sich eine Drehzahl von 2600 1/min. Mit einer Übersetzung von 2, was mit Zahnrädern und den Platzverhältnissen gut realisierbar ist, ergibt sich eine Abgangsdrehzahl für den Motor von 5200 1/min. Das ist im Rahmen der Motoren von maxon motor ag. Was jedoch den Motor an seine Grenzen führt, wird die Grenzbeschleunigung sein. Durch die Beschleunigung bedingte Trägheitskraft, welche durch ein Moment vom Motor überwunden werden muss. Das Moment rechnet sich aus der gewollten Beschleunigung, der Masse und dem Hebel auf den Rädern. Wird nun die Übersetzung von 2 noch eingerechnet, ergibt sich ein Moment von 110 mNm. Durch die Schienen haben wir eine gewisse elektrische Leistung zur Verfügung. Diese ergibt sich aus dem Produkt der Spannung 20 Volt und dem Strom von 3 Ampere. Theoretisch stehen also 60 Watt zur Verfügung. Die Entscheidung fällt auf einen DC Motor der Reihe DCX. Ein DCX 32 wird nun eingebaut und erfüllt nach den ersten Tests seine Anforderungen.



Abbildung 6: DC Motoren

3.3 Fahrwerk

Das Fahrwerk (siehe Abbildung 7) des Zuges besteht aus einem Antriebswagen (Position 1), einem Führungswagen (Position 2) und der Ladefläche (Position 3), auf welcher die Würfelaufnahme angebracht ist. Die drei Positionen werden in den nächsten Abschnitten genauer vorgestellt. Die Position 4 ist das Ladegut, der Holzwürfel.

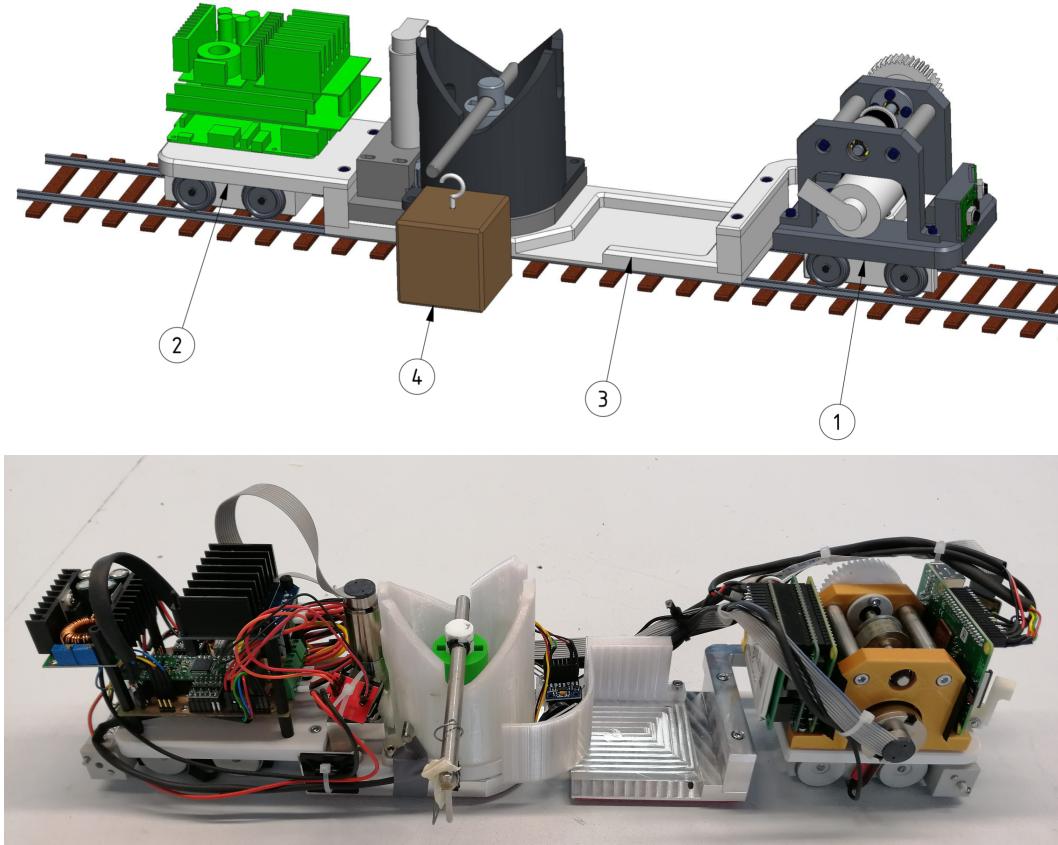
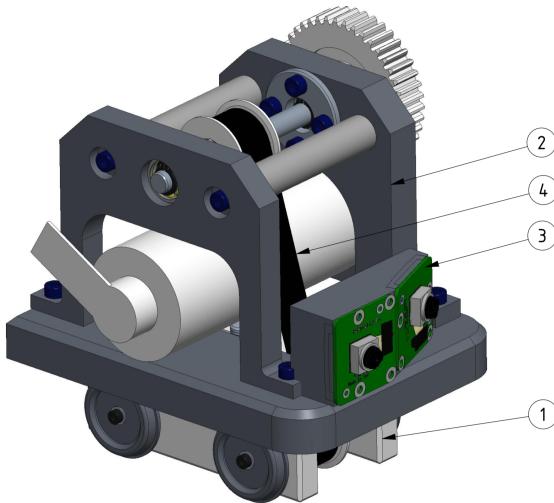


Abbildung 7: Konzept und Ergebnis des Fahrwerkes

Antriebswagen

Der Antriebswagen (siehe Abbildung 8 bzw. 9) besteht aus zwei gelagerten Achsen, welche beide durch einen Riemen angetrieben werden. Der Motor treibt ein kleines Getriebe mit einem Übersetzungsverhältnis von 2:1 den Riemen und somit die beiden Achsen an. Der Motor befindet sich in der Mitte des Riemens, mit dem Ziel den Schwerpunkt möglichst tief zu halten. Alle Räder sind mit Schrauben und Unterlagscheiben an den Achsen befestigt, damit ein schnellerer und einfacher Radwechsel möglich ist. Zwischen den Rädern befindet sich auf beiden Seiten des Wagens je ein Schleifkontakt aus Federblech, welcher mit einer kleinen Vorspannung auf die Gleise drückt. So werden kleine Unebenheiten auf der Strecke kein Problem für die Stromabnahme. An der Spitze des Wagens befinden sich zwei Kamerhalterungen, welche einstellbar und mit einem Gummipuffer gedämpft sind. Durch einen Bügel und einer Radiallagerung ist der Antriebswagen mit dem Ladungsträger verbunden.



Position	Bezeichnung
Position 1	Wagen
Position 2	Antriebseinheit
Position 3	Kameras
Position 4	Zahnriemen

Tabelle 1: Positionsnummern

Abbildung 8: Konzept Antriebswagen



Abbildung 9: Antriebswagen Endprodukt

Direkt unterhalb der Kamerahalterung befindet sich eine Kurvenvorrichtung (siehe Abbildung 10), welche die Gefahr des Entgleisens des Zuges minimiert. Sie funktioniert folgendermassen: Durch zwei Stahlstifte, welche radial die Kontur des Gleises haben und drehbar gelagert sind wird die Lokomotive in der Spur geführt. Die beiden Federn sorgen für den nötige Anpressdruck, damit die Vorrichtung in die Gleise verkeilt wird.

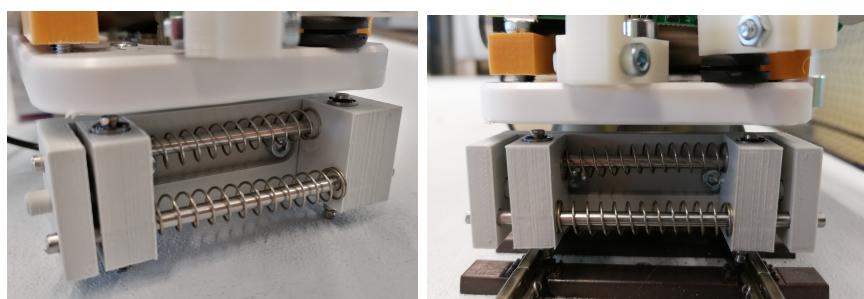


Abbildung 10: Kurvenvorrichtung in unmontierter und montierter Zusatztand

Maximale Beschleunigung

Der Motor kann gemäss Datenblatt ein maximales Moment von 85.6 mNm erreicht werden. Durch das Übersetzungsverhältnis 2:1 des Getriebes wird ein Moment von 171.2 mNm auf die beiden Radachsen übertragen und somit eine theoretische maximale beschleunigung von 44.79 Meter pro Sekunde im Quadrat erreichen (siehe Tabelle 2):

$$F_{Rad} = \frac{F_G}{8} = \frac{3kg \cdot 9.81m/s^2}{8} = 0.375N$$

$$F_{Reibung} = F_{Rad} \cdot k = 0.375N \cdot 0.3 = 0.1125N$$

$$M_{Rad} = M_{Getriebe} \cdot 0.5 = 171.2mNm \cdot 0.5 = 85.6mNm$$

$$M_{Rad} = F_{Reibung} \cdot a_{max,Getriebe} \cdot D_{Rad,Endprodukt} \cdot 0.5$$

$$a_{max,Getriebe} = \frac{M_{Rad}}{\frac{F_{Reibung}}{g} \cdot D_{Rad} \cdot 0.5} = \frac{0.0856Nm}{\frac{0.1125N}{9.81m/s^2} \cdot 0.026m \cdot 0.5} = 58.53m/s^2$$

Grösse	Wert
Durchmesser Rad (Konzept) [D,Konzept]	22 Millimeter
Durchmesser Rad (Endprodukt) [D,Endprodukt]	26 Millimeter
Reibungskoeffizient [k]	0.3

Tabelle 2: Grössen für die Beschleunigungsberechnung

Das theoretische maximale Moment beziehungswiese die maximale Drehzahl kann jedoch nicht erreicht werden da diese durch die vorgegebene Speisung beschränkt ist.

In der Konzeptphase wurde die theoretische Maximalbeschleunigung gemäss CAD-Daten der Lokomotive (siehe Tabelle 2) wie folgt berechnet:

$$M_{Rad,Konzept} = F_{Rad} \cdot 0.5 \cdot D_{Rad,Konzept} = 0.1125N \cdot 0.5 \cdot 22mm = 1.24mNm$$

$$M_{Rad,Prototyp} = F_{Rad} \cdot 0.5 \cdot D_{Rad,Endprodukt} = 0.1125N \cdot 0.5 \cdot 26mm = 1.46mNm$$

$$a_{max} = \frac{F_{Reibung}}{\frac{F_{Rad}}{g}} = \frac{0.1125N}{\frac{0.375N}{9.81m/s^2}} = 2.94m/s^2$$

Maximale Geschwindigkeit

Die Maximale Geschwindigkeit in der Kurve wurde in der Theorie mit den gegebenen Grössen (siehe Tabelle 3) folgendermassen berechnet:

Grösse	Wert
Minimaler Radius [r]	0.8 Meter
Masse [m]	3 Kilogramm
Schwerpunkt in x-Achse (maximaler Wert) [x]	0.0225 Meter
Schwerpunkt in y-Achse (maximaler Wert) [y]	0.05 Meter

Tabelle 3: Grössen für die Geschwindigkeitsberechnung

Die Gewichts- und Zentripetalkraft, welche das Gleichungssystem für die Geschwindigkeitsberechnung bilden, sind wie folgt definiert:

$$F_G = m \cdot g = 3kg \cdot 9.81m/s^2 = 29.4N$$

$$F_{max,z} = \frac{F_G \cdot x}{y} = \frac{29.4N \cdot 0.0225m}{0.05m} = 13.24N$$

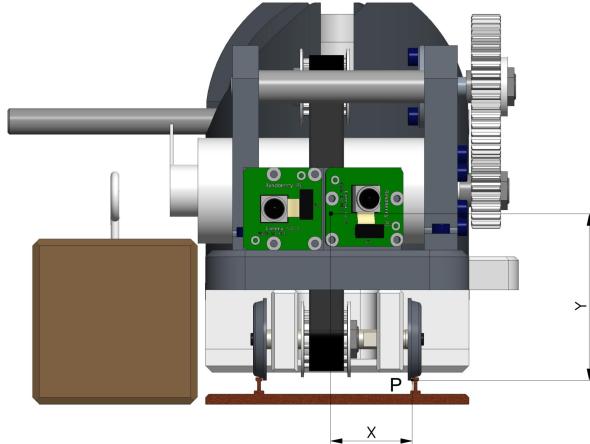


Abbildung 11: Schwerpunkt der Lokomotive

Da das Drehmoment eine vektorielle Grösse ist, müssen die beiden entstehenden Momente am Drehpunkt "P" am Gleis zusammen Null ergeben (siehe Abbildung 11). Oder anders gesagt, müssen die beiden Momente gleich gross sein, damit das System "statisch" bestimmt ist. Die Berechnungen sind auf den kleinsten Kurvenradius ausgelegt, da dort die grösste Zentripetalkraft entsteht. Somit ergibt sich eine maximale Geschwindigkeit von 1.53 Meter pro Sekunde:

$$F_{max,z} = \frac{m \cdot v_{max}^2}{r}; v_{max} = \sqrt{\frac{F_{max,z} \cdot r}{m}} = \sqrt{\frac{13.24N \cdot 0.8m}{3kg}} = \mathbf{1.53m/s}$$

Führwagen

Der Führwagen in Abbildung 12 ist grundsätzlich gleich aufgebaut wie der Antriebswagen. Er hat zwei gelagerte Achsen, an welchen je zwei Räder mit Schrauben und Unterlagscheiben befestigt sind. Zwischen den Rädern hat es jeweils einen Schleifkontakt aus Federstahl, welcher auf die Gleise vorgespannt ist und den Strom somit an einer zweiten Stelle vom Gleis abnimmt. Am hinteren Ende des Führwagens befindet sich wiederum die Vorrichtung für die Kurvenfahrt. Durch einen Stift und ein Radiallager ist der Ladungsträger mit dem Führwagen schwenkbar verbunden.

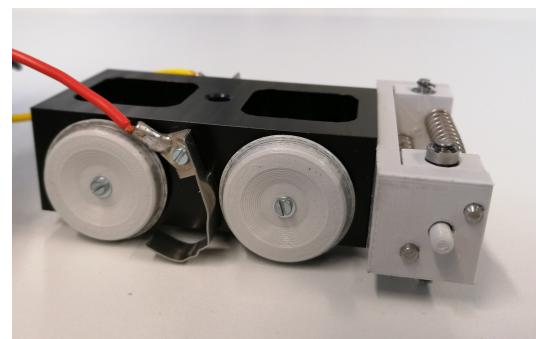


Abbildung 12: Führwagen

Der Aufbau des Führwagens ist wie konzipiert, mit Ausnahme der Räder, finalisiert und hergestellt worden. Der Führwagen ist folgendermassen aufgebaut:

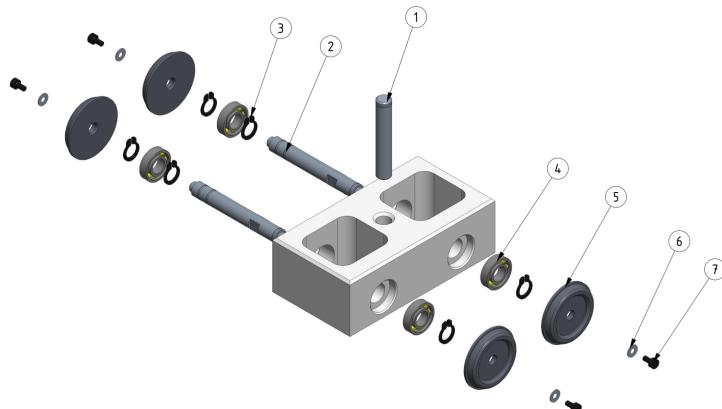


Abbildung 13: Konzept des Führwagens

Position	Bezeichnung
Position 1	Drehachse Wagen-Ladungsträger (eingepresst)
Position 2	Achsen (Gewinde an beiden Seiten, Anfrässtufe für Gabelschlüssel)
Position 3	Sicherungsring für Rillenkugellager
Position 4	Eingepresstes Rillenkugellager (Festlager) bzw. Loslager
Position 5	Rad
Position 6	Unterlagscheibe
Position 7	Zylinderschraube

Tabelle 4: Positionsnummern

Befestigung der Elektronik Komponenten

Der Montageraum für die Elektrokomponenten ist beim Endprodukt nicht nur auf dem Führungswagen, sondern ebenfalls auf dem Antriebwagen. Auf dem Antriebwagen sind Elektronik Komponenten mit Klett an der Antriebseinheit befestigt. Weitere wie Buzzer oder Näherungssensor an den Freien Flächen des Zuges angebracht. Auf der Abbildung 14 sind die montierten Elektronik Komponenten und deren Verkablung ersichtlich. Die Komponenten sind detaillierter in Kapitel 3.4 beschreiben.

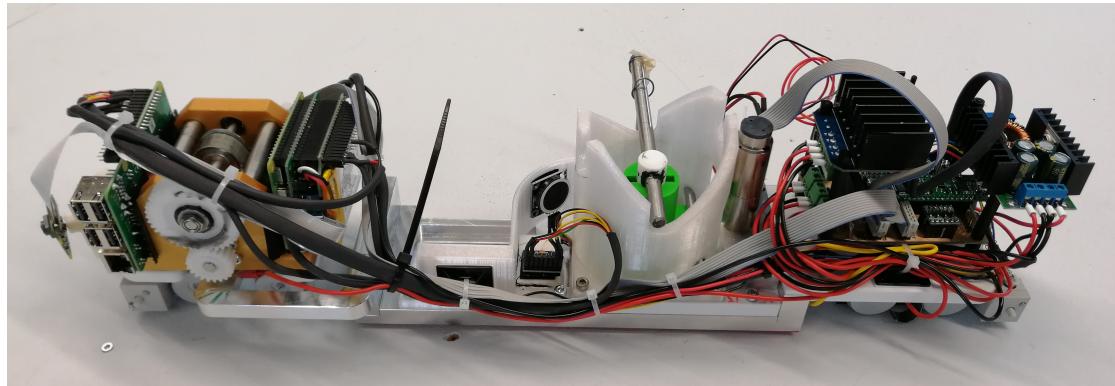


Abbildung 14: Elektrokomponenten

Stromabnahme

Die anfänglich konzipierten Schleifkontakte (siehe Abbildung 15) gebogen aus Federstahl erweisen sich als eine einfache und zuverlässige Art der Stromabnahme. Für die ersten Testfahrten wurde eine Stromabnahme am hinteren Wagen, dem Führungswagen, angebracht. Es wurde jedoch festgestellt, wenn Unebenheiten der Strecke vorhanden sind, die Möglichkeit eines Kontaktverlustes besteht. Somit wurde in einem zweiten Schritt eine zweite Stromabnahme am Antriebwagen befestigt, welche diese Gefahr minimiert.



Abbildung 15: Schleifkontakte

3.4 Elektrische Verbindungen

Dieses Kapitel beschreibt die Verbindungen der Elektronischen Komponenten für die Stromversorgung und Kommunikation. Abbildung 16 veranschaulicht den Aufbau der Elektronik. Darauf sind alle Verbindungen für die Kommunikation der Logik und die Stromversorgung eingezeichnet. Details zu den Logikverbindungen mit den Sensoren und Aktoren werden im Kapitel 3.5 erläutert. Für eine bessere Übersicht wurden die beiden Raspberry Pi's zusammengefasst als nur eine Komponente dargestellt.

Im Zentrum der Elektronik ist der Mikrocontroller Tiny K22. Die Software auf dem Mikrocontroller initialisiert alle Komponenten, überwacht deren Status und sendet die nötigen Informationen an das Pi. Diese Schnittstelle ist detailliert im Kapitel 3.8 beschrieben. Weiteres Angaben zur implementierten Software auf dem Tiny werden in Kapitel 3.6 beschrieben.

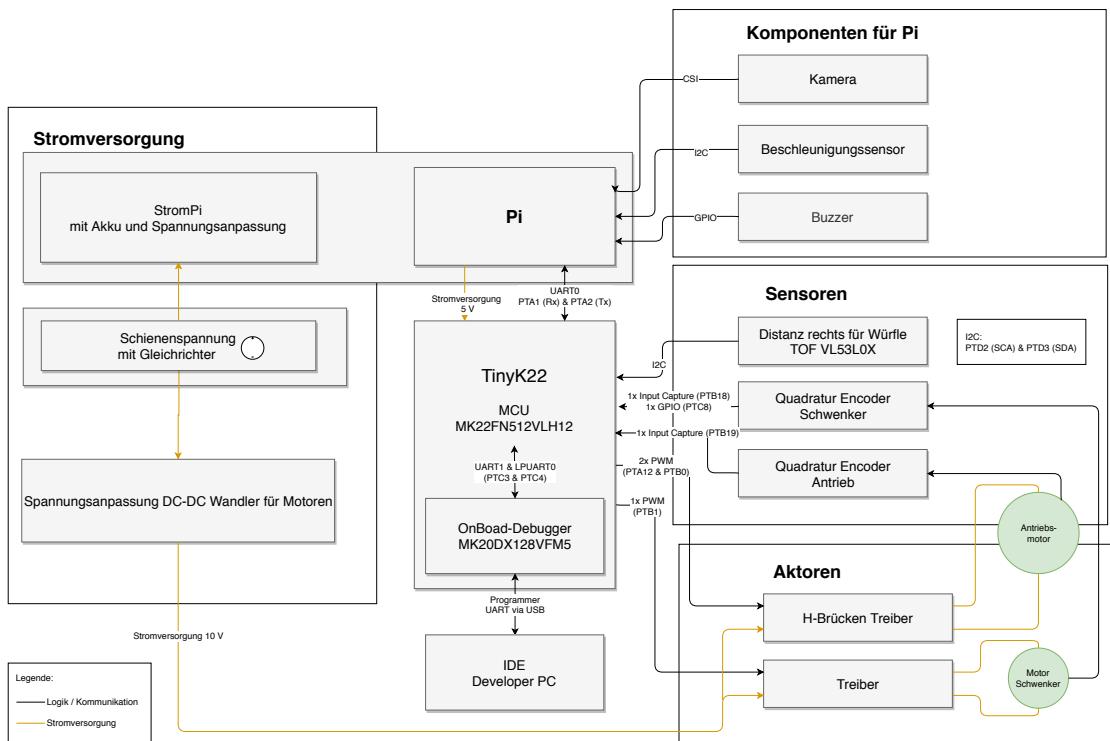


Abbildung 16: Komponentendiagramm Elektronik

Alle Verbindungen mit dem Tiny werden über Stecker auf einem PCB¹ verbunden. (siehe Abbildung 17) Dies ermöglicht einen flexiblen modularen Aufbau. Auch die Gleichrichtung der Versorgungsspannung der Schienen erfolgt direkt auf dem PCB, sowie die nötigen Pegelwandlungen zwischen 3.3 V und 5 V der verschiedenen Komponenten.

¹PCB: Printed Circuit Board - Leiterplatte

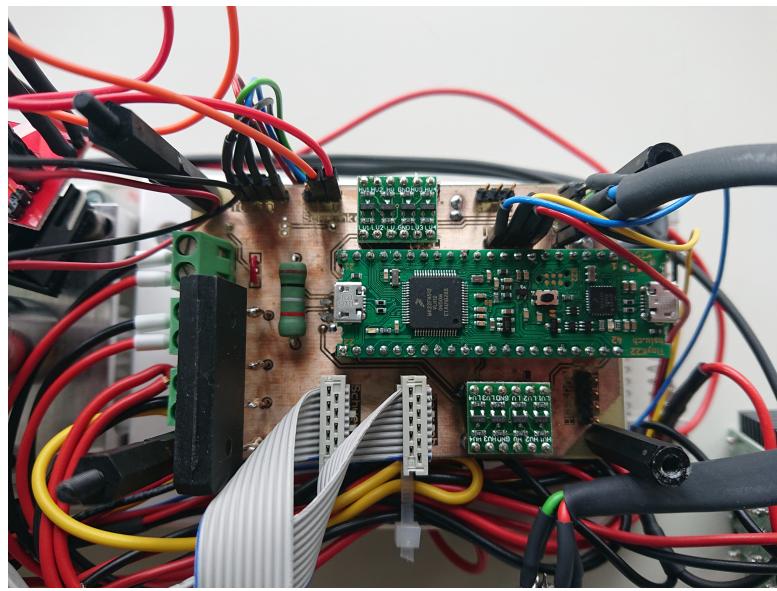


Abbildung 17: PCB mit TinyK22

3.4.1 Anordnung auf dem Zug

In Abbildung 18 sind die Montierten Elektronik Komponenten ersichtlich. Die Komponenten wurden mit Schraub oder Klettverbindungen befestigt.

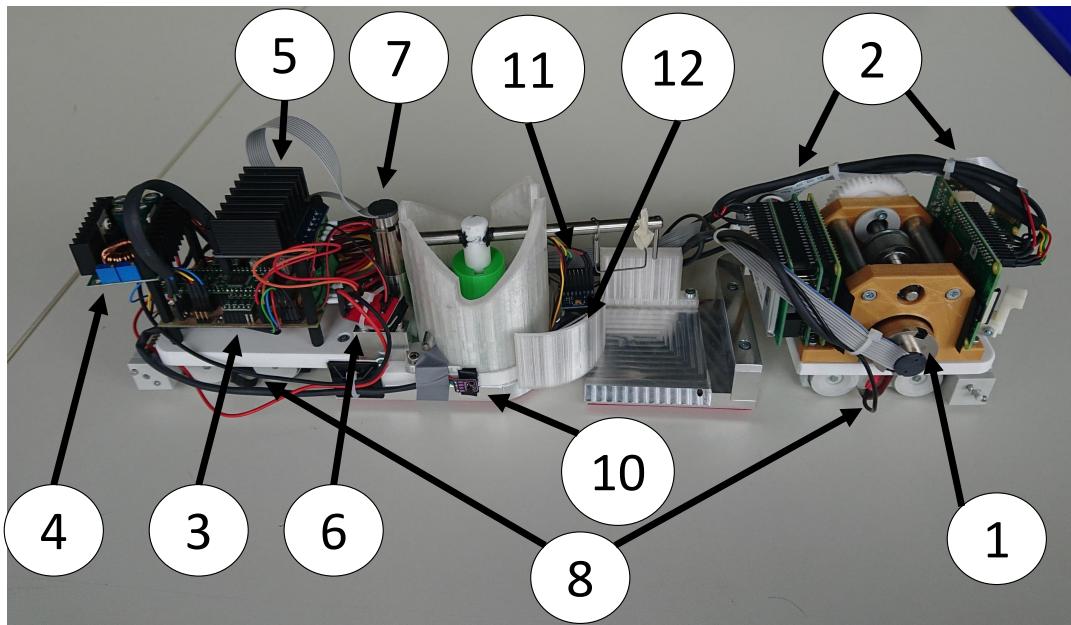


Abbildung 18: Soul Train mit Elektronik

An der Front des Zuges, an der Antriebseinheit mit Antriebsmotor [1], sind die beiden Raspberry Pi's [2] befestigt. Hinten auf dem Zug befindet sich das PCB [3] mit dem DC-DC Wandler [4] und den beiden Motorentreiber (Treiber [5] für den Antriebsmotor [1], Treiber [6] für den Schwenkermotor [7]). Direkt hinter der Schwenkervorrichtung ist der Motor für den Schwenker [7] befestigt. Jeweils zwischen den Räder befindet sich auf

jeder Seite ein Schleifkontakt für die Stromabnahme von den Schienen [8]. Diese werden beide nach hinten geführt und auf das PCB [3] verbunden. Direkt unter dem Schwenkerarm ist der TOF-Sensor [10] für die Würfelerkennung montiert. Vor dem Schwenker befinden sich der Beschleunigungssensor [11] und der Buzzer [12]. Diese sind direkt mit dem Raspberry Pi verbunden und werden auch von dort angesteuert und ausgelesen. Alle Verbindungen sind mit Kabel auf der Rückseite des Zuges durchgeführt und befestigt. (siehe Abbildung 19)

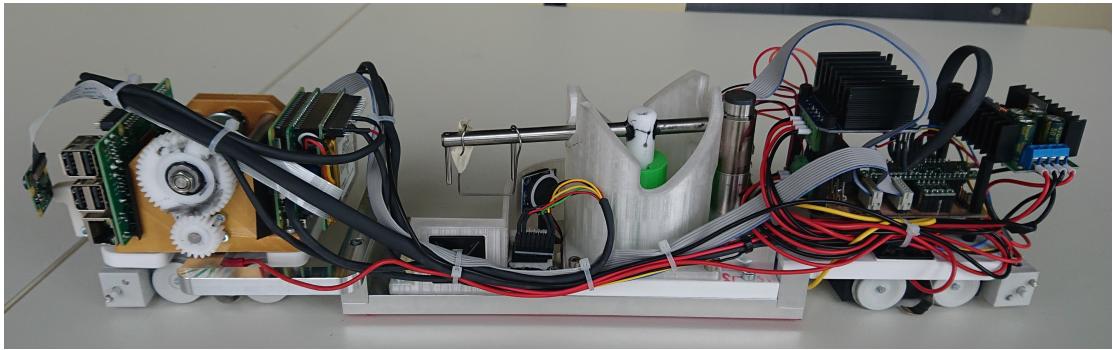


Abbildung 19: Kabelverbindungen Rückseite

3.5 Elektronik Komponenten

In diesem Kapitel werden die einzelnen Komponenten beschreiben, die für die Steuerung des Zuges verwendet werden. Dies beinhaltet alle Sensoren und Aktoren, die mit den Mikrocontroller TinyK22 verbunden sind.

3.5.1 Sensoren

Mit diversen Sensoren sollen folgende Daten aufgenommen werden.

- Geschwindigkeit
- Position
- Distanz rechts (Erkennung Würfel)
- zurückgelegter Winkel des Schwenkers

Geschwindigkeit

Die Geschwindigkeit wird über den Quadratur Encoder am Antriebsmotor aufgenommen.

Der Quadratur Encoder MR, Typ L gibt 1024 Impulse pro Umdrehung. Der Verlauf eines Impulses ist in Abbildung 20 dargestellt. Der Encoder stellt drei Kanäle zur Verfügung. Über die Kanäle *A* und *B* kann einzeln die Geschwindigkeit bestimmt werden. Durch Auswerten der Phasenverschiebung der beiden Kanäle ("*A* eilt *B* vor" oder "*A* eilt *B* nach") kann zusätzlich noch die Drehrichtung bestimmt werden. Der Kanal *I* ist mit Kanal *A* und *B* synchronisiert und kann ebenfalls zur Bestimmung der Geschwindigkeit dienen.

Für die Bestimmung der Geschwindigkeit kann die Dauer zwischen zwei Impulsen gemessen werden, oder es können die Anzahl Impulse in einer bestimmten Zeit gezählt werden. Für dieses Projekt werden die Anzahl Impulse in einer genau definierten Zeit

gemessen. Dazu wird bei jedem Impuls ein Zähler inkrementiert und immer nach 10 ms ausgelesen und wieder zurückgesetzt. So kann immer über die letzten 10ms die Durchschnittsgeschwindigkeit bestimmt werden.

Die Geschwindigkeit des Zuges (v_{Zug}) kann somit aus der Anzahl Impulsen (Anz_I) mittels der Mechanischen Übersetzung und der Grösse der Räder berechnet werden.

$$v_{Zug}(Anz_I) = \frac{2 \cdot \pi \cdot Anz_{Impulse}}{\Delta T \cdot I_{Umdrehung} \cdot U} \quad (1)$$

r_r : Raddurchmesser = 13mm

ΔT : Messperiode = 10ms

$I_{Umdrehung}$: Impulse pro Umdrehung = 1024

U Mechanische Übersetzung = 2

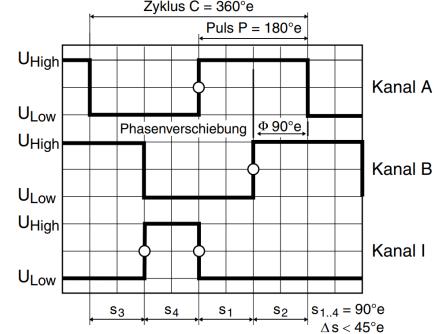


Abbildung 20: Signalverlauf Encoder
(www.maxonmotor.ch)

Position

Durch das Zählen aller Impulse ab dem Start, kann die absolut zurückgelegte Distanz bestimmt werden.

Auch kann die zurückgelegte Distanz durch eine Integration der Geschwindigkeit bestimmt werden. Unter der Annahme, dass zu Beginn der Messung zum Zeitpunkt $t = 0$ die zurückgelegte Strecke 0 ist ($s(t = 0) = 0$), kann die Geschwindigkeit zum Zeitpunkt t bestimmt werden mit

$$s(t) = \int_0^t v(x) dx$$

Da aber auf einem Digitalen System die Daten nur zu diskreten Zeitpunkten ausgewertet werden können, ergibt sich dann eine Summen der Geschwindigkeiten zum Zeitpunkt k

$$s[k] = \sum_{i=0}^k v[i] \Delta t$$

Die Geschwindigkeit wird gemäss der Beschreibung oben bestimmt.

Distanz rechts (Erkennung Würfel)

Gemäss der Aufgabenstellung befindet sich der Würfel in einem Abstand von $8 \pm 1\text{cm}$ von der Gleismitte. Somit muss der Distanzsensor Distanzen zwischen ca. 20mm und 80mm erkennen können. Der exakte Wert der Distanz ist dabei nicht entscheidend, da nur ein bestimmter Schwellwert erkannt werden muss. Die Distanz zum Würfel wird mit einem TOF Sensor ² VL53L0X ermittelt.

Dieser Schwellwert wird Messtechnisch ermittelt. Der Würfel wird an der vorgegebenen Position vor dem Sensor platziert (siehe Abbildung 21) und der Wert ausgelesen. Dieser Wert kann dann in der Software festgelegt werden um zu entscheiden ob der Würfel erkannt wurde oder nicht. Der Wert des Sensors wird nicht in eine exakte Distanz umgerechnet, es wird direkt mit den rohen Sensorwerten der Schwellwert bestimmt. Mit der beschriebenen Messung konnten mit dem Schwellwert 10000 die zuverlässigsten Resultate erzielt werden.

²TOF: Time-Of-Flight

Der genaue Ablauf der Messung und Entscheidung der Software auf dem Tiny ist in Kapitel 3.6.2 beschreiben.

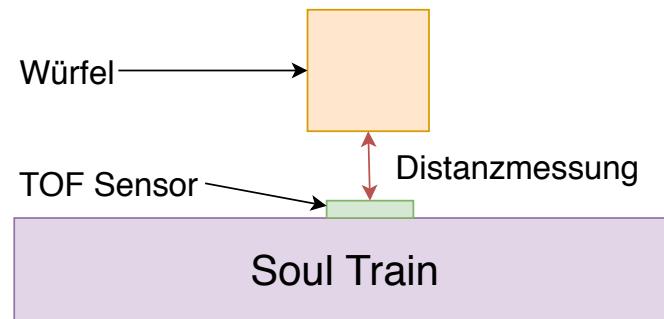


Abbildung 21: Distanzmessung mit TOF Sensor

Schwenkerwinkel

Um der Zurückgelegte Winkel des Schwenkers zu ermitteln ist auch auf dem Motor des Schwenkers ein Quadratur Encoder befestigt. Dieser Verhält sich vom Prinzip identisch wie der Encoder auf dem Antriebsmotor. Um den zurückgelegten Winkel zu ermitteln werden die Anzahl Impulse des Encoders gezählt. Der Motor muss den Schwenker drehen bis ein bestimmter Zielwert der Impulse erreicht ist. Dieser Zielwert wird mit der Mechanik messtechnisch ermittelt indem der Schwenker von Hand von der Start- bis zur Zielposition bewegt wird. (siehe Abbildung 22) Die dabei gemessene Anzahl Impulsen über die ganze Bewegung entspricht dem Zielwert.

Diese Messung ergab 5435 Impulse für den nötigen Winkel.

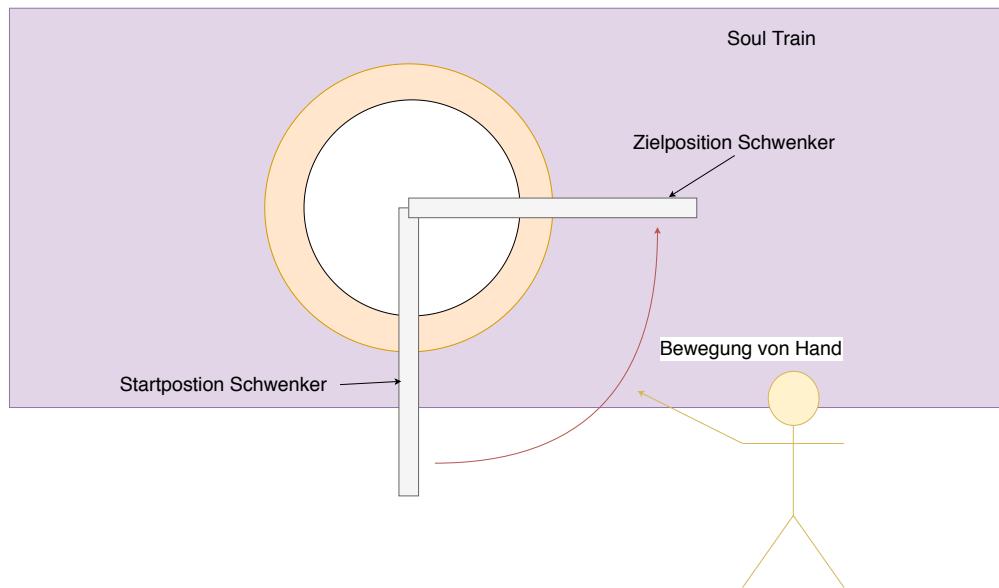


Abbildung 22: Winkelmessung Schwenker

3.5.2 Aktoren

Die Aktoren stellen die Schnittstelle zur Mechanik dar. Diese sollen alle nötigen mechanischen Bewegungen auf Befehl des Mikrocontrollers ausführen.

H-Brücken Treiber für Antriebsmotor

Um den Antriebsmotor anzusteuern, wird ein H-Brückentreiber verwendet. Damit kann der Motor durch ein PWM Signal in der Geschwindigkeit fast beliebig eingestellt werden. Über die wahlweise Ansteuerung einer der beiden Eingänge der H-Brücke wird die Richtung bestimmt.

Es wird ein Arduion IBT_2 DC-Motoren Treiber mit einem BTS7960 eingesetzt. Dieses Bauteil kann Motoren mit einem Strom von bis zu 43A versorgen.

Antriebsmotor

Als Antriebsmotor dient ein Maxon DCX 32 L. Dieser kann eine Leistung von bis zu 70 Watt umsetzen. Maxon [2018a] Dies ist bei einer Versorgungsspannung von 24V spezifiziert. Da aber nicht die maximale Drehzahl benötigt wird, kann auch eine entsprechend tiefere Spannung angelegt werden. Die Drehzahlkonstante des Motors ist $350 \frac{\text{min}^{-1}}{\text{V}}$. Für die angestrebte Drehzahl von 5200min^{-1} ergibt sich eine Spannung von

$$\frac{5200 \text{min}^{-1}}{350 \frac{\text{min}^{-1}}{\text{V}}} = 14.8 \text{V}$$

Somit muss der Antriebsmotor mit einer Spannung von 14.8V versorgt werden.

Motortreiber für Schwenker-Motor

Da die Richtung immer dieselbe ist reicht für den Schwenker ein normaler DC-Motoren Treiber. Um eine sanfte Beschleunigung und Bremsung der Konstruktion zu ermöglichen, wird auch der Schwenker-Motor mit einem PWM angesteuert. Als Treiber wird ein Board mit einem L298N verwendet.

Schwenker-Motor

Für den Schwenker wird ein Maxon DCX 19 S verwendet. Um die Position des Schwenkers zu bestimmen, ist auch an diesem Motor ein Encoder befestigt. Die Verwendung dieses Encoders ist in Kapitel 3.5.1 beschrieben. Maxon [2018b]

3.6 Software TinyK22

Dieses Kapitel beschreibt die Software, welche auf dem Mikrocontroller TinyK22 implementiert ist. Diese Software kommuniziert mit dem Pi und steuert die nötigen Sensoren und aktoren an. Die Schnittstelle zum Pi ist in Kapitel 3.8 beschrieben. Weitere Angaben zu den Sensoren und Aktoren und deren elektrische Verbindungen sind in Kapitel 3.4 und 3.5.

Die Software auf dem Mikrocontroller wird gemäss 5 in verschiedene Module unterteilt. Jedes Modul initialisiert jeweils alle nötigen Komponenten. Die Komponenten initialisieren jeweils die nötigen Hardware Schnittstellen und allfällige Sub-Komponenten.

Modulname	Beschreibung
com	Kommunikation mit dem pi
cube	Würfelerkennung mit Ansteuerung des TOF Sensors über I2C
drive	Ansteuerung des Antriebmotors mit Regelung der Geschwindigkeit mittels dem Feedback vom Quadraturencoder
crane	Ansteuerung des Motors für den Schwenker mit Positionsregelung mittels dem Feedback vom Quadraturencoder

Tabelle 5: Module Software TinyK22

In Abbildung 23 ist die Aufteilung veranschaulicht. Mit grün sind jeweils die einzelnen Module gekennzeichnet. Die Module benutzen jeweils ihre Komponenten abwärts bis zum Zugriff auf die Hardware. In Abbildung 23 gelb gekennzeichnet repräsentiert die Komponenten, welche auf die Hardware des Mikrocontrollers zugreifen. Orange kennzeichnet die Komponenten, welche als Schnittstellen zwischen dem Hardwarezugriff und der Logik dienen. Diese Bereiten Informationen der Hardware auf in ein Format, dass die Logik gut verarbeiten kann. Die Logik ist in den rot gekennzeichneten Komponenten implementiert. Diese verarbeitet die Daten und trifft Entscheidungen. In Abbildung 23 aus Übersichtsgründen nicht ersichtlich ist, dass alle module auf das com modul zugreifen können um mit dem Pi zu kommunizieren. Dabei benützen die Module die Komponenten comPi, comAck und comLog.

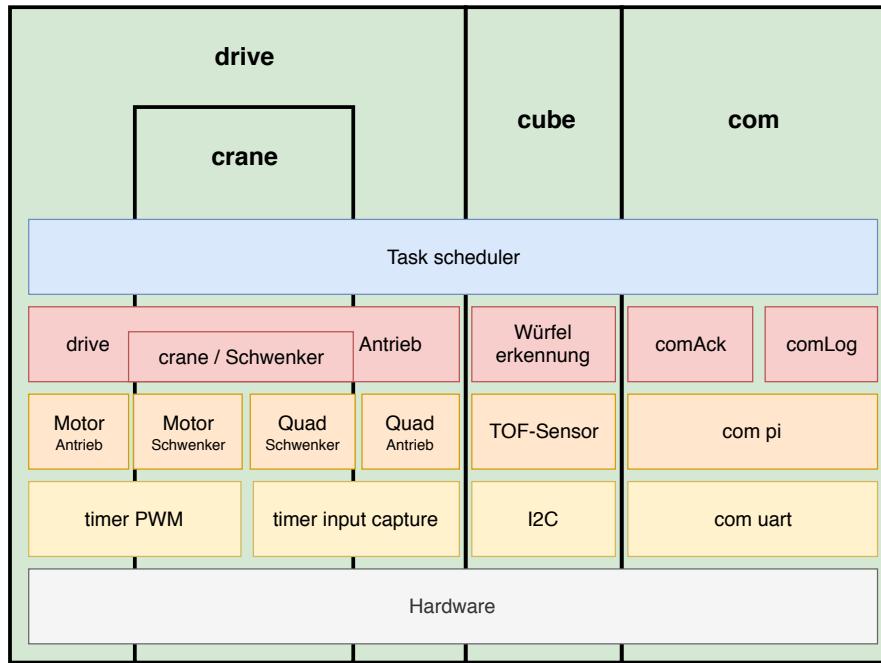


Abbildung 23: Modulaufteilung TinyK22

Task Scheudler

Für jedes Modul gibt es eine Taskfunktion, die während des Betriebs in einem Task Scheduler Loop periodisch aufgerufen wird. Auf die Benutzung eines Betriebssystems wurde in diesem Projekt verzichtet. Das heißt es gibt also keine Prioritäten oder Time-Slicing für die Tasks. Abbildung 24 veranschaulicht das Prinzip der Implementation für einen Task. Wichtig dabei zu beachten ist, dass mit dieser Variante die vorgegebenen Zeitintervalle nicht exakt und je nach Auslastung variieren. Da die Tasks aber ohnehin nicht stark zeitkritisch sind, funktioniert dies trotzdem. Alle Zeitkritischen Aufgaben werden über Interrupts behandelt.

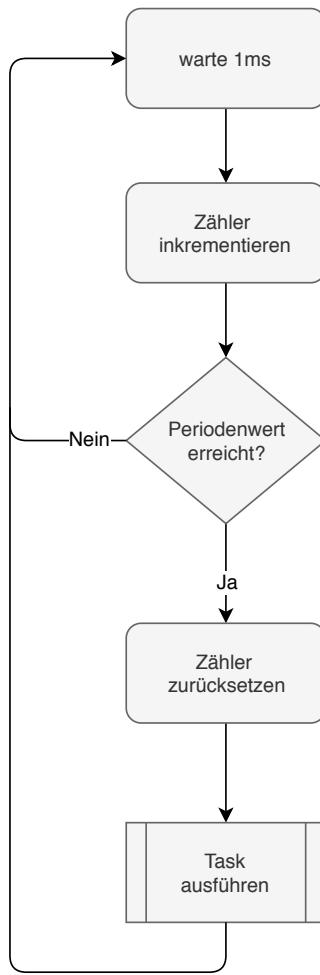


Abbildung 24: Flussdiagramm Task Scheudler

3.6.1 Modul com

Das Modul com ist in verschiedene Komponente unterteilt. In Tabelle 6 sind die verschiedenen Komponenten und deren Funktion aufgelistet. Die Komponente uart regelt den Zugriff auf die Hardware der UART-Schnittstelle und die Komponente pi verarbeitet die Frames allgemein. Die übrigen Komponenten benutzen für ihre Aufgabe jeweils diese beiden Komponenten als Basis.

Komponente	Beschreibung
uart	regelt den Zugriff auf die Hardware Schnittstelle und Interrupts hat einen Ringbuffer implementiert, in dem die Empfangenen Daten zwischengespeichert werden.
pi	hat einen Task implementiert in dem periodisch der Ringbuffer der Komponente uart überprüft und wenn nötig ausgelesen wird bei einem vollständig Empfangenen Frame wird das entsprechende Modul informiert und der Empfangene Wert übergeben
comAck	speichert und verwaltet alle ausstehenden acknowledge Meldungen
comLog	Alle Module können mit den Funktionen und Makros dieser Komponente geordnete Log Meldungen an das Pi Schicken mit einem durch eine Präprozessor-Anweisung einstellbaren logLevel kann die Menge an nicht wichtigem Informationsaustausch reduziert werden

Tabelle 6: Komponente Modul pi

uart

Diese Komponente ermöglicht anderen Komponenten einen einfachen Zugriff auf die UART Schnittstelle. Es wird ein Sende- und Empfangs-Ringbuffer verwaltet. Diese Komponente Stellt funktionen zur verfüzung um einzelne Zeichen oder ganze Zeilen in den Sende-Ringbuffer zu schreiben oder vom Empfangs-Ringbuffer zu lesen.

Für das Empfangen und Senden einzelnder Zeichen ist in dieser Komponente ein Interrupt Handler für die UART Schnittstelle implementiert.

pi

Diese Komponente Stellt funktionen zur verfüzung um Strings, signed Int oder unsigned Int für ein bestimmtes frame Topic zu versenden. Auch kann beim Senden einen Acknowledge-Handler angegeben werden (genauere Angaben dazu im Abschnitt comAck). Desweiteren können Module hier einen FrameLineHandler registrieren. In diesem wird ein bestimmtes Topic und eine Funktionszeiger angegeben. Wenn ein Frame mit dem angegeben Topic empfangen wird, ruft die Komponente pi diese angegeben Funktion auf und übergibt den empfangen Wert. Damit kann das Modul auf gewünschte Topics informiert werden.

Die Funktion piDoWork() wird als Task periodisch aufgerufen und überprüft den Empfangs-Ringbuffer auf neue Frames. Ist ein solches angekommen wird das entsprechende Topic identifiziert und das entsprechende Modul über den Handler informiert. Abbildung 25 veranschaulicht den Ablauf der Kommunikation zwischen einem Modul und der Komponente pi.

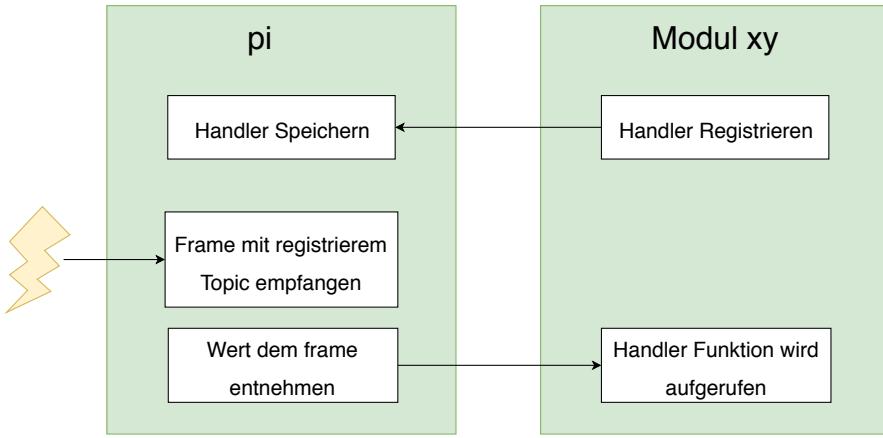


Abbildung 25: Ablauf mit Frame Handler

comAck

Damit ein Modul sicherstellen kann, dass seine verschickten Nachrichten beim Pi angekommen sind, kann dafür ein ackHandler registriert werden. Beim Senden einer Nachricht kann dieser dann als Ausstehend markiert werden.

Die Funktion `ackCheckQueue()` wird dann als Task periodisch aufgerufen. Diese überprüft welche Acknowledge noch ausstehend sind und informiert das Modul über den ackHandler wenn ein Acknowledge zu lange nicht empfangen wird.

Um festzustellen welche Acknowledge empfangen werden, registriert die Komponente `comAck` einen FrameLineHandler bei der Komponente `pi` gemäss vorherigem Abschnitt.

3.6.2 Modul cube

Das Modul `cube` benutzt den TOF-Sensor über die I2C Schnittstelle um zu erkennen ob der Würfel neben dem Zug erkannt wurde. Sobald der Würfel erkannt wurde wird das Pi informiert.

Die Komponenten beinhalten die Implementation für das I2C Protokoll und die nötige Kommunikation mit dem TOF-Sensor. Diese Komponenten werden dann von der Komponente `cubeDetection` benutzt und die Daten ausgewertet und davon eine Entscheidung getroffen.

Komponente	Beschreibung
I2C	Implementation des I2C Protokolls und den Zugriff auf die Hardware der I2C Schnittstelle
VL53L0X	Initialisierung und Kommunikation für den TOF-Sensor benutzt die Komponente I2C
cubeDetection	Wertet die Daten des TOF-Sensor aus und entscheidet wann der Würfel erkannt wurde und meldet dies dem pi

Tabelle 7: Komponente Modul cube

I2C

Für die Implementation des I2C Protokolls wird eine Komponente aus dem Unterricht an der HSLU verwendet. Diese initialisiert die Hardware Schnittstelle und stellt sicher

dass das I2C Protokoll korrekt umgesetzt wird. Sie stellt Funktionen zur Verfügung um in ein Register auf dem Slave zu schreiben und von einem Register von dem Slave zu lesen.

VL53L0X

Dies ist eine Bibliothek für den TOF-Sensor VL53L0X. Sie wird vom Hersteller Pololu zur Verfügung gestellt, ist aber nur für Arduino Systeme verfügbar. Für diese Projekt wurde die Bibliothek angepasst, damit der Zugriff auf die I2C Schnittstelle über die Funktionen der im vorhergehenden Abschnitt beschriebenen Komponente erfolgt.

Diese Komponente stellt Funktionen zur Verfügung um den Sensor zu initialisieren und danach bei Bedarf den aktuellen Messwert auszulesen.

cubeDetection

Diese Komponente wertet die Daten des TOF-Sensors aus und entscheidet, wann der Würfel erkannt wurde. Die Funktion cubeDoWork() wird periodisch als Task aufgerufen. Dort wird der TOF-Sensor ausgelesen und der gemessene Wert mit einem Schwellwert verglichen. Die Ermittlung dieses Schwellwertes ist in Kapitel 3.5.1 beschrieben.

Das Modul kann drei Zustände einnehmen. Diese sind "notFound", "finding" und "found". Der Initialzustand ist "notFound".

Sobald der Schwellwert bei einer Messung unterschritten wurde, wird der Zustand in "finding" geändert und es werden noch 20 weitere Messungen durchgeführt und mit dem Schwellwert verglichen. Von diesen 20 Messungen müssen mindestens 90% den Schwellwert ebenfalls unterschreiten um in den Zustand "found" zu wechseln. Durch diese Verifizierung können einzelne Fehlmessungen gefiltert werden. Werden die 90% bei der Verifizierung nicht erreicht wird noch überprüft ob mindestens 50% den Schwellwert unterschritten haben. Ist dies der Fall wird sofort eine neue Verifizierung gestartet und der Zustand bleibt auf "finding". Ist dies nicht der Fall geht das Modul wieder in den Zustand "notFound".

Abbildung 26 veranschaulicht den Ablauf der Würfelerkennung.

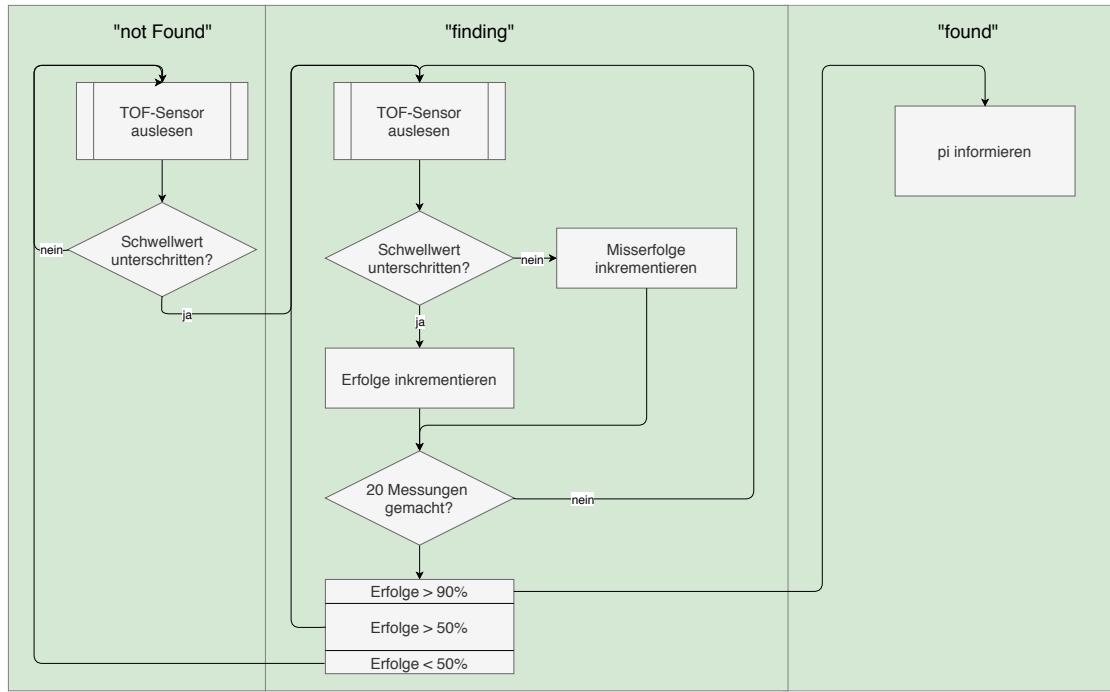


Abbildung 26: Ablauf für die Würfelerkennung

3.6.3 Modul drive

Für den Antrieb ist das Modul drive zuständig. Hier wird der Antriebsmotor angesteuert, die wahre Geschwindigkeit über den Quadraturencoder bestimmt und mit diesem Feedback dann die Geschwindigkeit durch eine Regelung stabilisiert.

Komponente	Beschreibung
motor	Ansteuerung des PWM für den Motor über ein Hardware Timer-Modul
quad	Zählt die Impulse des Quadraturencoders und bestimmt die wahre Geschwindigkeit des Zuges
drive	wertet die Geschwindigkeit der Komponente quad aus und setzt anhand dieses Feedbacks die Ansteuerung des Motors mit der Komponente motor

Tabelle 8: Komponente Modul drive

motor

Die Komponente motor setzt den Hardware Timer des Mikrocontrollers gemäss den Vorgaben. Dabei kann mit der Funktion „motor_A_SetPwm()“ der Duty-Cycle des PWMs eingestellt werden. Dabei kann ein Wert zwischen $-100'000$ und $100'000$ übergeben werden. Das Vorzeichen bestimmt dabei die Richtung. (- entspricht rückwärts fahren)

quad

Mit der Komponente quad wird die wahre Geschwindigkeit des Zuges bestimmt. Dazu werden die Impulse des Quadraturencoders über eine bestimmte Zeit gezählt und daraus die Geschwindigkeit bestimmt. Um die Zeit des Messintervalls möglichst genau zu haben wird dafür ein Timer eingesetzt. Dieser überläuft jeweils nach $10ms$. Beim Überlauf

dieses Timers werden die Anzahl Impulse gespeichert und der Zähler wieder auf null zurückgesetzt.

Die Geschwindigkeit des Zuges kann dann mit der Formel 1 auf Seite 19 in der Software bestimmt werden.

drive

In dieser Komponente ist die Taskfunktion `driveToWork()`. Diese wird periodisch aufgerufen. Dabei wird die wahre Geschwindigkeit bestimmt und mit der Soll-Geschwindigkeit verglichen. Damit die wahre Geschwindigkeit möglichst gut der Soll-Geschwindigkeit entspricht, ist in dieser Taskfunktion ein PID-Regler implementiert. Die Stellgrösse dieses Reglers steuert mittels der Komponente `motor` den Antriebmotors an. Abbildung 27 zeigt das allgemeine Funktionsprinzip eines PID-Reglers.

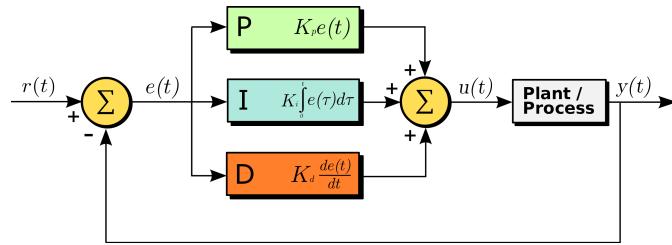


Abbildung 27: Allgemeines Prinzip eines PID-Reglers Wikipedia

Auch begrenzt das Modul `drive` die Beschleunigung des Zuges indem die Soll-Geschwindigkeit nicht direkt als von der Vorgabe übernommen wird, sondern mit einer Konstanten Steigung nachgeführt wird bis die Vorgabe-Geschwindigkeit erreicht ist. Damit beschleunigt der Zug mit höchstens einem Millimeter pro Sekunde pro Mikrosekunde.

3.6.4 Modul crane

Das Modul `crane` steuert den Motor für den Schwenker an. Es sorgt dafür, dass der Würfel auf Befehl vom Pi aufgenommen wird. Es besteht aus nur einer Komponente `crane`. In dieser Komponente wird der Schwenker Motor angetrieben bis am Quadraturencoder eine Vorgegebene Anzahl Impulse erfasst wurden. Diese Anzahl Impulse wird gemäss Kapitel 3.5.1 und Abbildung 22 auf Seite 20 ermittelt.

3.7 Aufgabentrennung zwischen Pi und Tiny

Dieses Kapitel beschreibt die Aufgabentrennung zwischen der Systemsteuerung auf dem Raspberry Pi 3+ (im folgenden Pi genannt) und dem Mikrocontroller MK22FN512VLH12 auf dem Entwicklerboard TinyK22 (im folgenden Tiny genannt). Das Pi dient im System als Master. Damit fällt das Pi alle Entscheidungen. Das Tiny dient als Slave, es führt die Entscheidungen vom Pi aus und gibt Rückmeldung zum aktuellen Status und zu den Sensordaten.

Im System sind folgende Aufgaben für die jeweiligen Steuerungen vorgesehen:

Pi:

- Entscheidung Start gemäss Befehl vom Webinterface
- Auswertung der Kameraaufnahmen
 - Entscheidungen gemäss erkannter Schilder
- Entscheidung der Fahrgeschwindigkeit
- Auswertung des Beschleunigungssensors
- Ansteuerung des Buzzer
- Versenden der Sensordaten an das Webinterface

Tiny:

- Ansteuerung / Regelung Antriebsmotor
- Ansteuerung Schwenkermotor
 - inkl. Auswertung Position des Schwenkers bis vollständig eingefahren
- Auslesen von Sensordaten
 - Objekterkennung Würfel
 - aktuelle Ist-Geschwindigkeit
 - bestimmung Schwenkerposition beim einfahren

Somit ist jedes System auf Informationen des anderen angewiesen. Deshalb ist eine klare Definition der Schnittstelle der beiden Komponenten nötig. Im folgenden Kapitel 3.8 wird diese Interface genauer beschrieben.

3.8 Interface zwischen Pi und Tiny

Dieses Kapitel beschreibt die Kommunikation zwischen der Systemsteuerung auf dem Raspberry Pi und dem Mikrocontroller MK22FN512VLH12 auf dem Entwicklerboard TinyK22. Dabei sollen Informationen zur aktuellen Situation, sowie auch Informationen zum aktuellen Status des jeweiligen Systems ausgetauscht werden.

Diese Kommunikation soll es den Systemen ermöglichen, die zugewiesenen Aufgaben gemäss Kapitel 3.7 zu erfüllen.

3.8.1 Hardware Schnittstelle

Als Hardware Schnittstelle wird UART (auch RS-232 genannt) verwendet. Dies ist eine asynchrone serielle Schnittstelle. Die Kommunikation kann mit zwei Leitungen realisiert werden. Eine dient als Empfangsverbindung (Rx) und eine als Sendeverbindung (Tx). Diese Trennung erlaubt eine voll-duplexe Kommunikation.

3.8.2 Übertragungsprotokoll

Sobald ein System Information für das andere System hat werden Datenpakete (im folgenden Frame genannt) in einem fest vorgelegten Format ausgetauscht. Dabei gibt es ein Grundformat für die Informationen und Definitionen für Schlüsselwörter für eine Identifikation der Daten.

Die Schlüsselwörter und die Bedeutung der zugehörigen Daten unterscheiden sich dabei für Frames vom Tiny zum Pi und für Frame vom Pi zum Tiny. Die Inhalte sind so ausgelegt, dass jedes System seine Aufgaben gemäss dem Kapitel 3.7 erfüllen kann.

Grundformat

Die Frames bestehen jeweils aus einem Bezeichner (Key) und einem Wert (Value). Diese werden mit einem Komma getrennt. Als Abschluss dient das New Line Zeichen '\n'. Somit sieht ein Informationsstück folgendermassen aus:

$\{Key\}, \{Value\}\n$

Zum Beispiel würde eine Information zur Geschwindigkeit folgendermassen aussehen:

speed, 200

Die gesamten Informationen werden als Zeichenkette (String) im Ascii Format verschickt. Zahlenwerte werden jeweils vor dem Senden in einen String umgewandelt und nach dem Empfangen wieder in einen Zahlenwert zurückkonvertiert. Dies verbessert die Leserlichkeit, was zum Testen und Simulieren der Kommunikation sehr hilfreich sein kann.

Frames Inhalte

In Tabelle 9 sind die Schlüsselwörter (Key) und die zugehörigen möglichen Werte aufgeführt.

Key (String)	Values	Von	Nach	Interval
speed	int32 [mm/s]	Pi	Tiny	-
crane	Each int32: 0 = nothing 1 = retract crane	Pi	Tiny	-
is_crane	Each int32: 0 = nothing 1 = done retract crane	Tiny	Pi	-
phase	Jeweils int32: 0 = startup 1 = find_cube 2 = grab_cube 3 = round_one 4 = round_two 5 = find_stop 6 = stopping 7 = finished	Pi	Tiny	-
is_speed	int32 [mm/s]	Tiny	Pi	50ms
cube	Each int32: 0 = nothing 1 = detected	Tiny	Pi	-
log	String to log on Raspi	Tiny	Pi	-
ack	String of command. One of: - speed - crane - is_crane - phase - is_speed - cube - log	Tiny/Pi	Pi/Tiny	-

Tabelle 9: Schlüsselwörter und mögliche Werte Kommunikation Pi \leftrightarrow Tiny

Acknowledge (ack)

Um eine zuverlässige Informationsübertragung zu garantieren bestätigen sich die Systeme jeweils den Empfangen eines Frames mit einer Acknowledge Nachricht. Der Inhalt dieser Nachricht ist der Schlüssel der Empfangen Nachricht. Bleibt also ein Acknowledge für zu lange Zeit aus, kann das System davon ausgehe, dass das Frame gar nicht oder nicht korrekt angekommen ist.

Periodische Frames

Die Information über die akutelle Geschwindigkeit (is_speed) wird alle 50ms vom Tiny zum Pi geschickt. Neben dem Austausch der Information dient dies auch als Lebenszeichen vom Tiny. Wenn das Tiny die aktuelle Geschwindigkeit nicht mehr schickt kann das Pi davon ausgehen, dass es ein Problem gibt und kann darauf reagieren indem z.B. der Benutzer über das Web-Interface informiert wird. Mit den Acknowledge Nachrichten auf die is_speed Frames weiss auch das Tiny jeweils, dass das System auf dem Pi noch korrekt läuft. Bleibt ein Acknowledge auf ein is_speed Frame zu lange aus kann das Tiny darauf reagieren indem es den Zug sicher zum Stillstand bringt bis das Pi wieder reagiert und neue Befehle schickt.

Alle anderen Frames werden nur nach bedarf verschickt, sobald neue Information verfügbar sind.

3.9 Steuerungssoftware

In diesem Kapitel wird auf die Implementierung der technischen Komponente 'Steuerungssoftware' eingegangen. Auch wird ein Augenmerk auf unsere Middleware 'ZeroMQ' und die Datenserialisierung mittels 'Protobufs' gelegt. Die Steuerungssoftware befindet sich im Mittelpunkt der Applikation und kann somit auch als Herzstück bezeichnet werden. Sie orchestriert die verschiedenen Module, damit ein funktionierendes Zusammenspiel ermöglicht wird und der Applikationsablauf geschmeidig durchlaufen wird.

3.9.1 Anforderung

- Korrekte Ausführung des PREN-Ablaufs
- Robust
- Ausführung einzelner Module

3.9.2 Lösung

Die Steuerungssoftware wird wie in PREN beschrieben mittels der Middleware Technologie 'ZeroMQ' und 'Protobuffers' unterstützt. Die Steuerungssoftware selbst erbt wie alle anderen Module auch von der Basisklasse 'App', dazu später mehr. Solange die Steuerungssoftware läuft wartet sie auf ein Startsignal, welches den Applikationsablauf in Gang setzt.

3.9.3 Technologien / Aufbau

Einblick in verwendete Technologien, sowie ein Überblick über Programmstruktur und Aufbau der Middleware.

ZeroMQ

ZeroMQ

Protocol buffers

Protocol buffers, auch Protobuf, ist ein von Google entwickeltes Protokoll zum Senden und Empfangen von serialisierten und deserialisierten Daten über verschiedene Dienste. Google's Design Ziel für Protobuf liegt darin kleiner, einfacher und schneller als XML zu sein. Unterstützt werden alle häufig verwendet Sprachen wie: Python, Java, Objective-C, C# und andere.

Beispiel Protobuf:

```
syntax = "proto3";  
  
message Direction {  
    string direction = 1; // straight , left , right  
}
```

Man sieht nun, dass eine 'Direction' Mitteilung definiert wird. Diese hat ein String Attribut, welches direction heisst. Mit dieser Definition wird mithilfe eines 'protobuffer compiler' Code generiert. Dieser kann die definierte Message Serialisieren und Deserialisieren.

Beispiel Generierung für Python:

```
protoc -I=pb --python_out=pb pb/direction.proto
```

Mithilfe von diesem Beispiel wird die Protobuf Definition (direction.proto) zu Python Code generiert.

Beispiel Generierter Python Code:

```
# Generated by the protocol buffer compiler. DO NOT EDIT!
# source: direction.proto
```

```
import sys
_b=sys.version_info[0]<3 and (lambda x:x) or (lambda x:x.encode('latin1'))
from google.protobuf import descriptor as _descriptor
from google.protobuf import message as _message
from google.protobuf import reflection as _reflection
from google.protobuf import symbol_database as _symbol_database
# @@protoc_insertion_point(imports)

_sym_db = _symbol_database.Default()

DESCRIPTOR = _descriptor.FileDescriptor(
    name='direction.proto',
    package='',
    syntax='proto3',
    serialized_options=None,
    serialized_pb=_b('\n\x0f\x64irection.proto\"\\x1e\\n\\tDirection\\x12\\x11\\n\\tdirection'
))

_DIRECTION = _descriptor.Descriptor(
    name='Direction',
    full_name='Direction',
    filename=None,
    file=DESCRIPTOR,
    containing_type=None,
    fields=[
        _descriptor.FieldDescriptor(
            name='direction', full_name='Direction.direction', index=0,
            number=1, type=9, cpp_type=9, label=1,
            has_default_value=False, default_value=_b("").decode('utf-8'),
            message_type=None, enum_type=None, containing_type=None,
            is_extension=False, extension_scope=None,
            serialized_options=None, file=DESCRIPTOR),
    ],
    extensions=[],
    nested_types=[],
    enum_types=[],
    serialized_options=None,
    is_extendable=False,
```

```

syntax='proto3',
extension_ranges=[],
oneofs=[
],
serialized_start=19,
serialized_end=49,
)

DESCRIPTOR.message_types_by_name[ 'Direction' ] = _DIRECTION
_sym_db.RegisterFileDescriptor(DESCRIPTOR)

Direction = _reflection.GeneratedProtocolMessageType( 'Direction' , ( _message.Message , )
DESCRIPTOR = _DIRECTION,
__module__ = 'direction_pb2',
# @@protoc_insertion_point(class_scope:Direction)
))
_sym_db.RegisterMessage(Direction)

# @@protoc_insertion_point(module_scope)

```

Das generierte Python File, welches mithilfe des 'protobuf compiler' erzeugt wurde.

Protobuf Nachrichten

Eine Auflistung aller verwendeter Protobuf Nachrichten, welche über die Middleware gesendet werden.

Nachricht	Bezeichner	Datentyp
Acceleration	x	Integer32bit
	y	Integer32bit
	z	Integer32bit
CraneCommand	command	Integer32bit
Cube	state	Integer32bit
Current	current	Integer32bit
Direction	direction	String
Distance	distance	Float
Heartbeat	component	String
	status	String
MoveCommand	speed	Integer32bit
NumberDetection	number	String
Speed	speed	Integer32bit
SystemCommand	command	String
	phases	Map<String, Boolean>
SystemStatus	phase	String
	message	String

Kommunikationsmodule

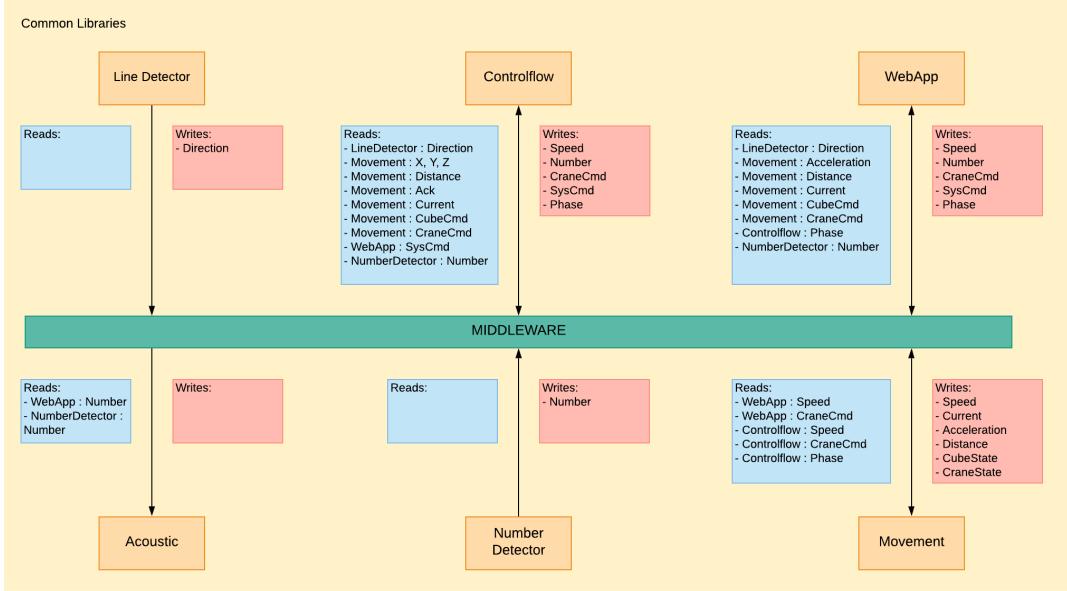


Abbildung 28: Kommunikation

Das Senden / Empfangen der Hearbeats wurde in obigem Diagramm bewusst weggelassen, die Hearbeats haben keinen direkten Einfluss auf das Verhalten der Applikation und sind somit für die Steuerung irrelevant.

Heartbeat

Der Heartbeat spiegelt den Zustand des Moduls wieder, welches ihn sendet. Jedes Modul besitzt einen Heartbeat, welches in regelmässigen Abständen gesendet wird. Ein Modul kann folgende Zustände annehmen:

Starting	Während der Initialisierungsphase des Moduls
Running	Solange das Modul läuft
Error	Sobald ein Fehler gemeldet wird oder das Modul keinen Heartbeat mehr sendet
Finished	Nach Abschluss seiner eigentlichen Tätigkeit

Folgende Module werden überwacht:

Line Detector	Liest die Richtung der aufkommenden Gleise während der Fahrt
Number Detector	Liest die Nummern während der Fahrt
Movement	Liest Geschwindigkeit und regelt die momentane Geschwindigkeit anhand dessen
Acoustic	Gibt die eingelesene Nummer akustisch wieder
Control Flow	Regelt den Ablauf der verschiedenen Phasen

Codestruktur (Basisklasse)

Alle Module erben von der Basisklasse.

```
class Buzzer(base_app.App):
    def __init__(self, *args, **kwargs):
        super().__init__("Acoustic", self.acoustic_loop, *args, **kwargs)
```

Module teilen sich die Eigenschaft einer nebenläufigen Methode, welche Nachrichten aus der Middleware liest. Somit wurde diese Funktionalität in eine Basisklasse ausgelagert. Die Basisklasse führt die Methode aus bis die Applikation beendet wird.

```
def run(self):
    try:
        while not self.stopped:
            self.loop(self.args, self.kwargs)
    except ProgramKilled:
        self.stopped = True
```

Das Auslesen, Senden und Verarbeiten der Daten erfolgt in jedem Modul separat.

3.9.4 Entwicklungsablauf

Bevor mit dem Ablauf begonnen wurde, stellten wir sicher, dass die einzelnen Module einwandfrei funktionieren, um Probleme während der Fahrt auf ein Minimum zu begrenzen. Die Steuerungssoftware wurde gegen Ende von PREN2 entwickelt und getestet.

Die Architektur war schon zu Beginn von PREN2 verfügbar und lauffähig. Während der Entwicklung von PREN2 wurde die Middleware kontinuierlich erweitert. Gegen Ende gab es ein Refactoring der Middleware Schnittstelle, wobei das Abonnieren von Nachrichten um ein Vielfaches einfacher wurde.

3.9.5 Testing

Während dem Testen kamen verschiedene Probleme auf. Ein grosser Blocker war das Fahren selbst. Der Zug blockierte (bei niedrigen Geschwindigkeiten) und entgleiste oft. Dies erschwerte das Testen des kompletten Ablaufs erheblich. Das Problem wurde von unseren Mechanikern rasch beseitigt.

Bei der Ausgabe eines akustischen Signals wurde festgestellt, dass die Kabel falsch verbunden wurden. Auch war die Ausgabe des Signals nicht wie erwartet, was ebenfalls behoben wurde.

Die Kamera wackelte während der Fahrt stark, wobei die Bilder verwackelt wurden. Mithilfe eines Gummiuntersatzes an der Kamerahalterung wurde dem Wackeln entgegengewirkt. Das Ergebnis waren einiges schärfere Bilder.

3.10 Beschleunigung

In diesem Kapitel wird auf die Implementierung der technischen Komponente 'Beschleunigung' eingegangen.

3.10.1 Anforderung

Anforderungen

- Momentane Beschleunigung auslesen
 - Fahrtrichtung
 - Querbeschleunigung
- Beschleunigung in Geschwindigkeit und Distanz umrechnen
- Approximierte Position berechnen

3.10.2 Lösung

Die Lösung wird wie in PREN1 beschrieben, umgesetzt.

Der Beschleunigungssensor wird direkt an die 'Movement'-Klasse angebunden. Diese verarbeitet die erhaltenen Daten weiter zu Geschwindigkeit und Beschleunigung.

Technologien / Komponente Softwaretechnisch ist die Komponente in Python3 realisiert. Für die Umsetzung wird auf die Bibliotheken 'smbus2', 'time' und 'sys' zurückgegriffen. 'smbus2' enthält alle nötigen Informationen über den GPIO (General Purpose Input / Output) Bus. 'time' und 'sys' enthalten für die Entwicklung unterstützende Funktionen.

Aufbau Der Sensor wird über die I^2C Schnittstelle angesprochen und verwendet somit eine Datenleitung (SDA) und eine Clockleitung (SCL). Für die Stromversorgung werden die Anschlüsse für 3.3V des GPIO Headers benutzt.

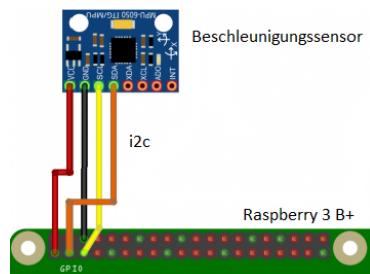


Abbildung 29: Verkabelung Beschleunigungssensor (<http://fritzing.org>)

Bezeichnung	GPIO Port	MPU 6050
Stromversorgung	3V3	VCC
Ground	GND	GND
Daten	SDA	SDA
Clock	SCL	SCL

Auf dem Zug findet der Sensor Platz zwischen unserem Kran und dem Abstellplatz für den Würfel. Bei der Montage wurde darauf geachtet, dass die x-Achse Richtung Zugspitze zeigt und die y-Achse die Kräfte nach Links oder Rechts erfasst. Die z-Achse findet keine direkte Anwendung.

Prozessablauf Der Sensor wird für jede Achse separat angesprochen, die erhaltenen Daten werden direkt das erste Mal umgewandelt. Sobald die 'Movement' Komponente gestartet wird, liest diese im 50ms Takt Daten über den Beschleunigungssensor ein. Die erhaltenen Daten werden ausgewertet und in Geschwindigkeit und Beschleunigung gerechnet.

3.10.3 Entwicklungsablauf

Es gibt es Vielzahl ähnlicher Beschleunigungssensoren auf dem Markt und somit auch mehr als genügend Bibliotheken auf dem Internet. Das Finden einer kompatiblen Bibliotheken erwies sich somit als eher trivial. Als nächstes wurde der Sensor an den GPIO Bus angeschlossen und mithilfe einer Testapplikation getestet.

Der Sensor reagierte jedoch nicht und so ging es weiter mit der Fehleranalyse. Nach dem Ausführen verschiedener anderer Applikationen und dem Austausch des Sensors wurde ich bald fündig. Wie sich herausstellte war das ganze nicht ganz so trivial wie gedacht und die Applikation muss auf Sensor, sowie GPIO angepasst werden. Ein Informatiker am Werk :)

Mit einem funktionierenden Sensor, konnte ich fortfahren und schrieb eine Klasse, welche den Sensor im 50ms Takt ausliest und weitergibt. Die WebApp und die Middleware wurde noch entsprechend angepasst.

3.10.4 Testing

Anfangs wurde die Komponente über eine einfache Klasse mit Ausgabe auf ein Terminal Fenster getestet. Nach der Anbindung an das WebApp konnte die Beschleunigung jederzeit auf dem Web-Client angesehen werden. Eine Plausibilisierung der erhaltenen Daten erfolgte nicht. Wir haben volles Vertrauen in den Sensor und die in der Applikation erfolgende Umrechnung.

3.10.5 Reflexion

Der Beschleunigungssensor ist eine interessante und im korrekten Einsatzgebiet sehr nützliche Komponente. Jedoch erhalten wir die Geschwindigkeit bereits vom Motor, somit haben wir redundante Daten bezüglich der Geschwindigkeit. Weil der Motor zuverlässigere Daten sendet, entschieden wir uns für den Weitergebrauch dieser Daten.

3.11 Akustik

In diesem Kapitel wird auf die Implementierung der technischen Komponente 'Akustik' eingegangen. Die akustische Ausgabe einer Zahl eine Teilanforderung für den korrekten Ablauf von PREN. Während der Fahrt wird ein Signal mit einer Nummer gelesen, diese Nummer wird am Ende der Fahrt akustisch wiedergegeben.

3.11.1 Anforderung

Anforderungen

- Zahl akustisch wiedergeben (Speaker oder Buzzer)
- Korrekte Zahl wird wiedergegeben
- Kompakt
- Günstig
- Keine eigene Stromquelle
- Verständliche Ausgabe

3.11.2 Lösung

Die Lösung entspricht dem Lösungskonzept aus PREN1. Sobald ein Signal, in Form einer Zahl von 1-10, erhalten wird, piepst der Buzzer die entsprechende Anzahl.

Technologien / Komponente Über die GPIO Header des Raspberry Pi kann der Buzzer direkt angesprochen und versorgt werden. Ein einzelnes Signalkabel reicht für die Kommunikation aus.

Softwaretechnisch wird das Ganze mit Python ausgeführt. Python besitzt mächtige Bibliotheken in diesen Bereichen.

Buzzer Es gibt zwei Arten von Buzzer, den Aktiv-Buzzer und den Passiv-Buzzer. Wir verwenden einen Passiv-Buzzer, der exakte Unterschied der beiden Modelle ist mir nicht bekannt, jedoch ist dies für die Implementierung nicht relevant. Für den Passiv-Buzzer gibt es viele Beispiele auf dem Internet.

Auf dem Zug befindet sich der Buzzer zwischen Kran und dem Abstellplatz für den Würfel. Grösse und Gewicht erlauben eine einfache Montur auf einem Klettstreifen.

GPIO Anbindung des Buzzers an den Raspberry Pi 3 B+ via GPIO Bus.

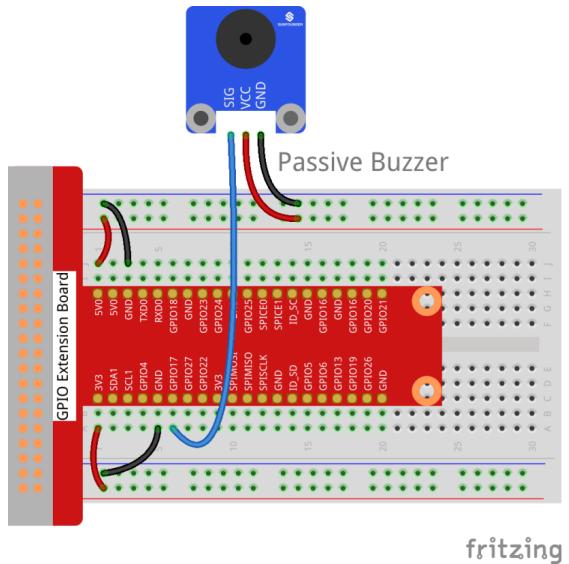


Abbildung 30: Verkabelung Buzzer

Bezeichnung	GPIO Header	Buzzer
Stromversorgung	3V3	VCC
Ground	GND	GND
Signal	GPIO17	SIG

Prozessablauf

3.11.3 Entwicklungsablauf

Logik und Aufbau für die Kommunikation mit dem Buzzer war bereits aus PREN1 vorhanden. Die Herausforderung lag lediglich in der Kommunikation mit der Applikation. Der Buzzer wurde mithilfe einer Akustik-Klasse an die Middleware gebunden. Zuletzt wird die WebApp mit einer Akustik-Schnittstelle erweitert, welche das Senden von Akustik-Signalen erlaubt.

3.11.4 Testing

Über die Akustik-Schnittstelle auf der WebApp kann der Buzzer jederzeit getestet werden.

3.11.5 Reflexion

Es gab keine bösen Überraschungen und der Buzzer läuft zuverlässig.

3.12 Webapplikation

In diesem Kapitel wird auf die Implementierung der technischen Komponente 'Web Applikation' eingegangen. Die Webapplikation wurde am Anfang des Entwicklungsprozess eingeführt und ist somit nicht in PREN1 dokumentiert.

3.12.1 Anforderung

Ziel und Zweck der Applikation ist es eine zentrale Übersicht der einzelnen Module / Komponente zu erhalten und diese zu testen. Somit fungiert die WebApp als Testing/-Monitoring Komponente und ist für den Ablauf des Controlflows irrelevant. Der Zug kann mittels Startknopf über das Webinterface gestartet werden.

Anforderungen

- Übersicht
 - Heartbeat
 - Informationen zu Zug & Fahrt
- Module testen
- Fahrt starten

3.12.2 Lösung

Der Client ist eine klassische Kombination aus html & javascript. Dieser spricht über eine Rest API mit unserem Server. Auf dem Server verwenden wir Sanic, eine auf Python basierte Servertechnologie.

Technologien / Aufbau

Struktur Über Sanic wird ein Webserver erstellt, welcher index.html und den Ordner Static bereitstellt. Die Static files beinhalten Dateien, welche vom Client direkt verwendet werden können, wie z.B. Javascript oder CSS.

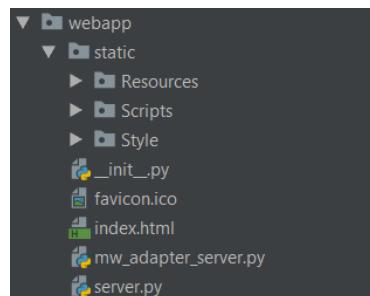


Abbildung 31: Directory Tree

Sanic Sanic ist ein Python 3.6+ Webserver und Webframework, welches den Fokus auf Performanz legt. Durch den Einsatz der im Python 3.5 hinzugefügten 'async/await' Syntax wird der Code nicht-blockierend und schneller.

Das Projekt ist für kleine Projekte ideal, auch ist Python ein grosser Pluspunkt, weshalb wir uns für diese Technologie entschieden haben.

REST API Kommunikation zwischen Client und Server wird mittels einer REST (Representational State Transfer) API (Application-Programming-Interface) gewährleistet. Wir beschränken uns auf die GET und POST Methoden, weil unsere Applikation eine überschaubare Komplexität aufweist.

REST ist eine weit verbreitete API Architektur, welche vor allem im Web Anwendung findet.

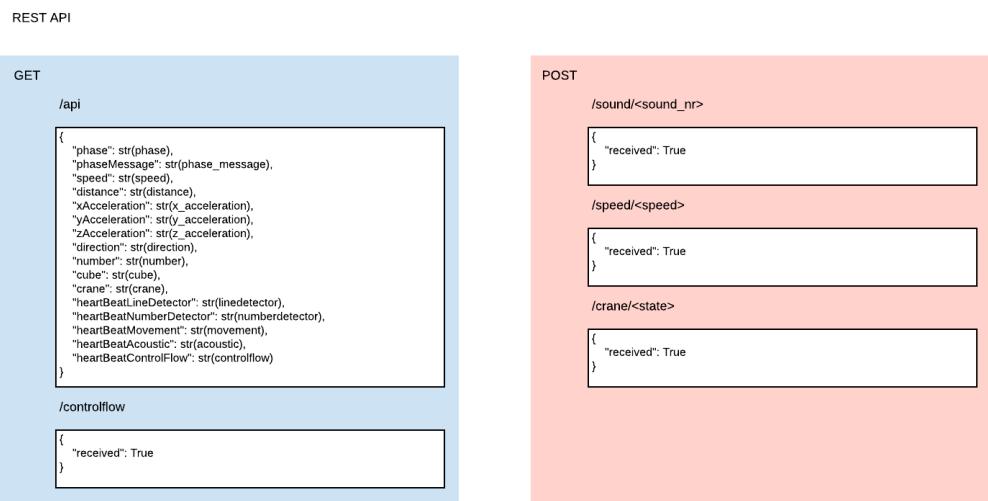


Abbildung 32: REST API

3.12.3 Entwicklungsablauf

Der Entwurf, sowie erster Prototyp der WebApp, wurden von Steve Ineichen anfangs von PREN1 erstellt. Das Hinzufügen immer weiterer Komponenten führte dazu, dass die WebApp im Laufe der Entwicklung kontinuierlich erweitert wurde.

3.12.4 Testing

Eine spezifische Testkomponente für Client oder Server existiert nicht. Die WebApp funktioniert als Testkomponente und ist somit Teststrategie der meisten Komponenten auf dem RasperryPi 3+.

Das Testing erfolgt während des Entwicklungsablaufs automatisch. Wird eine neu eingebundene Komponente auf dem Client simuliert, so wird die Funktionalität der WebApp mitgetestet.

3.12.5 Reflexion

Eine Testkomponente einzuführen war eine tolle Idee und hat sich während der Entwicklung sehr bewährt. Die WebApp liefert eine attraktive Übersicht betreffend den verschiedenen Komponenten.

3.13 Tafel- und Nummererkennung

In diesem Kapitel wird die Schild- und Nummererkennung des Soul- Trains erläutert. Die Schild- und Nummererkennung muss drei Aufgaben erfüllen:

- Erkennung der Schilder (grosse- / kleine- Tafel)
- Erkennung der Nummer auf dem Schild
- Erkennung des Startsignals

Wie im Abbildung 33 ersichtlich, wurde im Rahmen des PREN 1 die Erfüllung der Funktionalität bei einer mindestgeschwindigkeit von 0.5 m/s als Anforderung vorausgesetzt.

1.4	Fahrwerk		M, E, I
1.4.1	W Beschleunigung	0.5m/s2	M, E, I
1.4.2	W Höchstgeschwindigkeiten	0.5m/s	M, E, I
1.4.3	F Kurvenfahrt	Kurvenradien von 80 bis 150cm	M, E, I
1.4.4	M Bremsen	Anhaltgenauigkeit von +/- 1cm	M, E, I
1.4.5	W Dämpfung	Das Gerät verfügt über eine Dämpfungskonstante D > 1	M
<hr/>			
2	Sensorik		E,I
2.1	Fahrdaten auswertung		E, I
2.1.1	F Beschleunigungssensor Quer,Längs	muss ausgewertet werden	E, I
2.1.2	F Geschwindigkeitssensor	muss ausgewertet werden	E, I
2.1.3	W Positionsbestimmung	z.B um zum Startpunkt zurückzufahren und Anlauf holen	E, I
<hr/>			
2.2	Objekterkennung		
2.2.1	M Würfelerkennung		E, I
2.2.2	W Spurrichtung		E, I
2.2.3	W Zeitmessung	Start und Ende der Zeitmessung erkennen	E
2.2.4	W Lichtraumprofil erkennung		E
2.2.5	M Startvorrichtung		E
2.2.6	W Funkstart		E
2.2.7	M Signalerkennung (Info- und Haltesignal)	als unterstützung für Bilderkennung	E, I

Abbildung 33: Ausschnitt der Anforderungsliste PREN 1

3.13.1 Hardware

Für diese rechenintensiven Aufgaben wurde ein zusätzlicher Platinenrechner eingesetzt. Zum Einsatz kommt ein Raspberry PI 3 A+ [2]. Dieser verfügt wie sein «grosser Bruder» Raspberry PI 3 B+ [3] einen 64bit Cortex- A53 (ARMv8) SoC und einem 5GHz IEEE 802.11.ac Wireless LAN. Ihm stehen aber im Gegensatz zum B+ Model nur 512MB LPDDR2 RAM zur Verfügung.

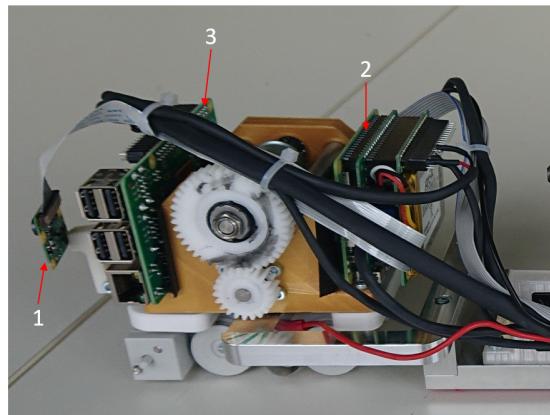


Abbildung 34: Ausschnitt der Anforderungsliste PREN 1

In PREN 1 wurde ein Raspberry PI Zero W für diese Aufgabe vorgesehen. Dieser musste aber aufgrund des schwachen Prozessors durch den 3 A+ ersetzt werden. Dank genügend vorhandenen Platz und des geringen Preisaufschlags konnte dieser Wechsel problemlos durchgeführt werden. Dafür kann nun auch die Nummererkennung komplett auf diesem Platinenrechner durchgeführt werden. Als Kamera kommt wie in PREN 1 evaluiert die Standardkamera vom Raspberry PI [1] mit der CSI (Camera Serial Interface) zum Einsatz.

3.13.2 Software: Ablaufsteuerung

Als Basisarchitektur des Slave- RPI (Raspberry PI 3 A+) kommt eine «State- Machine» zum Einsatz. Angebunden ist diese an der übergeordneten Middleware, welche die Verbindung zum Master- RPI (Raspberry PI 3 B+) sicherstellt. Mehr dazu im Kapitel 3.9. Der Ablauf, welcher den Slave- RPI tangiert, ist unterteilt in Phasen wie in Tabelle 10 ersichtlich.

Phase	Beschreibung
Startbereich	Erkennung des Startsignals
Runde 1	Erkennung der grossen Tafel & Nummererkennung & Erkennung des Startsignals
Runde 2	Erkennung der grossen Tafel & Nummererkennung & Erkennung des Stoppsignals
Runde 3	Erkennung der kleinen Tafel

Tabelle 10: Darstellung der Phasen für Slave- RPI

Daraus lässt sich die State- Machine wie in Abbildung 35 ableiten.

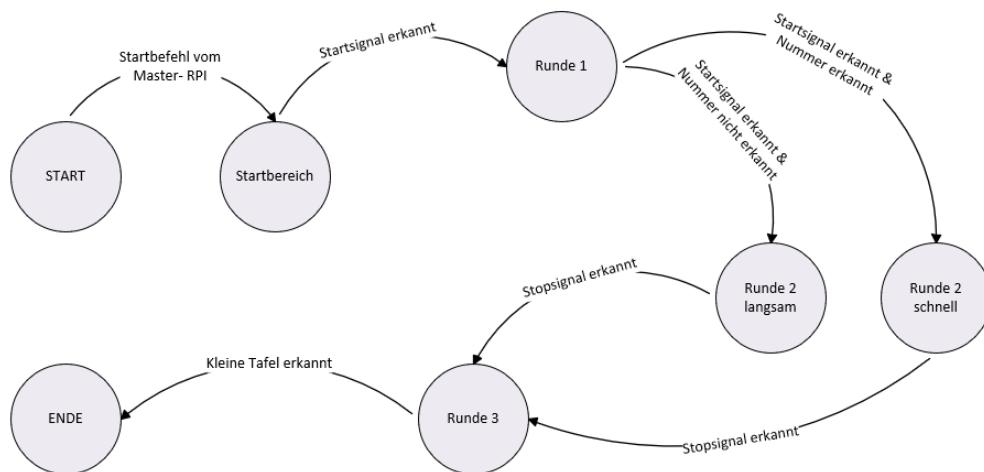


Abbildung 35: State- Machine Slave- RPI

State: START

Im Zustand *START* wartet der Slave- RPI bis vom Master- RPI der Befehl «Start» via Middleware gesendet wird. Dieser Befehl wird vom Master- RPI nach dem erfolgreichen Würfelaufnahme dem Slave- RPI gesendet. Anschliessend wechselt der Slave- RPI zum Zustand *Startbereich*

State: Startbereich

Im Zustand *Startbereich* versucht der Slave- RPI das Startsignal (blau) zu erkennen. Dabei fährt der Zug im normalen Tempo. Sobald das Startsignal erkannt wird sendet der Slave- RPI via Middleware dem Master- RPI den Befehl «11» für die erste Runde. Anschliessend wechselt der Slave- RPI zum Zustand *Runde1*.

State: Runde 1

Im Zustand *Runde 1* wird nun im normalen Tempo versucht die grosse Tafel mit der Nummer zu erkennen. Wird die Tafel und die Nummer erkannt und die Runde abgeschlossen (d.h. Startsignal wurde auch erkannt), wechselt der Slave- RPI zum Zustand *Runde 2 schnell*. Wird aber die Tafel oder die Nummer nicht erkannt und die Runde wird abgeschlossen, wechselt der Slave- RPI in den Zustand *Runde 2 langsam*.

State: Runde 2 schnell

Im Zustand *Runde 2 schnell* wird nun nicht mehr auf die grossen Tafeln und Nummern geachtet. Dem Master- RPI wird der Befehl «22» gesendet. Mit dem Befehl «22» an den Master- RPI, weiss dieser nun er kann mit vollem Tempo fahren. In der Runde 2 wird nur noch auf das Stoppsignal geschaut, damit der Slave- RPI in den Zustand *Runde 3* wechseln kann.

State: Runde 2 langsam

Im Zustand *Runde 2 langsam* wird nochmals in der 2. Runde versucht die grosse Tafel mit der Nummer zu erkennen. Der Zug fährt weiterhin mit gleichem Tempo wie in der Runde 1 weiter. Gleichzeitig wird auf das Stoppsignal für den Wechsel in den Zustand *Runde 3* geschaut.

State: Runde 3

Im Zustand *Runde 3* wird dem Master- RPI der Befehl «31» gesendet. Nun wird das Tempo des Zuges auf ein minimum verzögert, damit das Anhalten so genau wie möglich erfolgen kann. Dabei versucht der Slave- RPI die kleine Tafel mit der Nummer zu erkennen. Sobald diese erkannt wird, wechselt der Slave- RPI in den Zustand *ENDE*.

State: ENDE

Im Zustand *ENDE* wird dem Master- RPI der Befehl «0» gesendet. Nun wird der Zug gestoppt und das Programm des Slave- RPI ist abgeschlossen.

3.13.3 Software: Tafel- und Nummererkennung

Im PREN1 konnte schon viele Versuche im Bereich der Tafel- und Nummererkennung durchgeführt werden. Die daraus resultierenden Erkenntnisse konnten im PREN2 gut verwendet werden. Im folgenden werden die einzelnen Aufgaben (Tafel-, Start/ Stop- und Nummererkennung) einzeln behandelt und erläutert.

Start-/ Stop- Signalerkennung

Das Start- / Stop- Signal (im weiteren nur noch Startsignal genannt) befindet sich nach dem Startbereich und signalisiert den jeweiligen Start der nächsten Runde. Das Signal besteht wie in Abbildung 36 aus zwei blauen Rechtecken auf weissem Hintergrund.

Um das Signal zu erkennen wird dafür das Bildverarbeitungsframework OpenCV verwendet. Das Startsignal verfügt mit den blauen Streifen über ein Alleinstellungsmerkmal. Dies wird für die Erkennung ausgenutzt. Mittels **HSV- Filter** kann auf den genauen Farbraum gefiltert werden. So kann die Farbe Blau extrahiert werden. Als zweiter Schritt wird anschliessend Konturen erfasst und auf bestimmte Form und Grösse gefiltert. Da zwei blaue Streifen vorhanden sind, sollten jeweils zwei Treffer vorhanden sein. Wenn dies zutrifft wurde das Startsignal erkannt. Im Abbildung 37 links ist die Filterfunktion mit HSV- und Konturfilter ersichtlich.

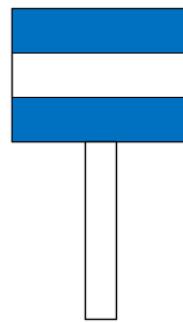


Abbildung 36: Startsignal

```
def filterAndMorphOPStart(self, frame, threshold, kernel, iterator):
    kernel1 = np.ones((15, 25), np.uint8)
    farb = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_blue = np.array([85, 151, 30])
    upper_blue = np.array([255, 255, 255])
    mask = cv2.inRange(farb, lower_blue, upper_blue)
    res = cv2.bitwise_and(frame, frame, mask=mask)

    color = cv2.cvtColor(res, cv2.COLOR_HSV2BGR)
    gray = cv2.cvtColor(color, cv2.COLOR_BGR2GRAY)
    ret, thresh1 = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)

    opening = cv2.morphologyEx(thresh1, cv2.MORPH_OPEN, kernel1)

    return opening
```

Abbildung 37: Codesnipped: Filter Startsignal

Filter	Beschreibung
HSV	Extraktion des blauen Farbraumes
Konturen	Auf Grösse und Form der Konturen filtern

Tabelle 11: Filter für Startsignalerkennung

Tafelerkennung

Die Tafel kann in zwei Ausführungen vorkommen. Weisse Zahl mit schwarzem Hintergrund oder umgekehrt. Für das Erkennen der Tafel kommen andere Filter wie beim Startsignalerkennung zum Einsatz. Anstatt einem HSV- Filter kommt bei der Tafelerkennung eine Kombination von drei Filtern zum Einsatz. Als erstes kommt ein **Blur-Filter** zum Einsatz. Dieser sorgt dafür, dass alle «harten» Kanten im Bild *weichgezeichnet* werden und somit können Störeinflüsse reduziert werden. Anschliessend kommt die Kombination **Binary- / OTSU- Filter** zum Einsatz. Diese Kombination dient dazu das Bild in ein *Binary-Picture* zu transformieren und somit die Konturen zu extrahieren. Dabei wird ähnlich wie beim Startsignal die Konturen auf Grösse und Form gefiltert.

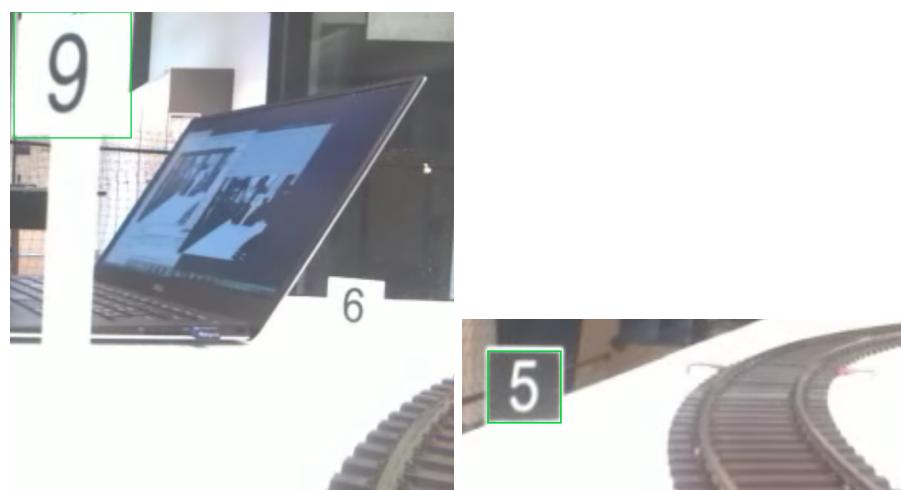


Abbildung 38: Tafelerkennung

Nummerkennung

Um nun die Nummer zu erkennen, wird ein Machine- Learning Framework eingesetzt. Erste Versuche im PREN 1 wurde mit der Bibliothek Keras und Backend Tensorflow durchgeführt. Für die Finale Version wurde aber auf die Software *Tesseract OCR* von Google gesetzt, dies aus den Gründen der Effizienz. Tesseract OCR verfügt über einen Python Wrapper welcher einfach über den Packetmanager *pip* installiert werden kann. Tesseract selber kann einfach mittels Sourcecode kompiliert und installiert werden. In Abbildung ist ein Codesnippet für den TesseractWorker abgebildet. Ein grosser Vorteil von Tesseract OCR ist auch die Einfachheit. Nachdem die Konfigurationen gesetzt sind kann mittels einem Funktionsaufruf die erkannte Nummer zurückgegeben werden.

```
def tesseractWorker(self):
    while True:
        try:
            frame = self.ImageQueue.get()
            config = ('-l eng --oem 1 --psm 3')
            print(pytesseract.image_to_string(frame, config=config))
        except:
            pass
```

Abbildung 39: Codesnipped Tesseract

Worker- Ablauf

Ein grosser Vorteil bei der gewählten Architektur mit zwei Raspberry PIs ist die verfügbare Rechenleistung für die Bildverarbeitung. Dieser Vorteil wird auch in der Architektur der Applikation genutzt. Wie in der Abbildung 40 ersichtlich werden die verschiedenen Funktionen parallel in mehreren Threads abgearbeitet. Somit erhält jeder Worker einen eigenen Thread und kann somit unabhängig voneinander arbeiten.

Worker	Beschreibung
Camera- Worker «Kamera»	Mittels Kamera wird ein Videostream in die beiden Worker- Queue «Startsignal- Queue» und «Tafel- Queue» aufgenommen.
Startsignal- Worker	Nimmt Frame für Frame aus der «Startsignal- Queue» und versucht wie oben beschrieben das Startsignal zu erkennen.
Tafel- Worker	Nimmt Frame für Frame aus der «Tafel- Queue» und versucht wie oben beschrieben die Tafel (klein / gross) zu erkennen. Wenn ein Treffer erzielt wurde wird das gefilterte Frame in die «Number- Queue» verschoben.
Numberdetection- Worker	Nimmt Frame für Frame aus der «Number- Queue» und versucht wie oben beschreiben die Nummer zu erkennen.
StateMachine- Worker	Dieser Worker beinhaltet die Ablaufsteuerung und somit die State- Machine.

Tabelle 12: Worker Beschreibung

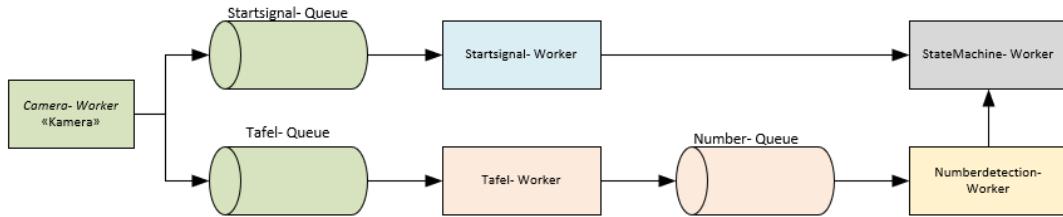


Abbildung 40: Architektur: Worker- Ablauf

3.13.4 Testing

Wie am Anfang des Kapitels erwähnt, beschränkt sich die definierte Anforderung vom PREN 1 auf die Erfüllung der Funktionalität und das Erreichen einer Minimalgeschwindigkeit von 0.5 m/s. Daraus ergibt sich für das Testing folgende Prioritäten:

- 1. Prio: Erfüllung der Funktionalität
- 2. Prio: Zuverlässigkeit erhöhen
- 3. Prio: Geschwindigkeit erhöhen

Aus diesen Prioritäten wurde ein Testplan 13 erstellt. Dieser beinhaltet verschiedene Testcases und der Grad der Erfüllung wurde dabei dokumentiert. Da direkt mit der physischen Umgebung (Kamera) getestet werden muss, konnten nur beschränkt automatisierte Tests durchgeführt werden. Die im Testplan erstellten erfassten Testcases sind alle manuelle Tests. Automatisierte Tests wurden mittels einzelnen Bildern realisiert.

TestNr.	Testcase	Beschreibung	Erfüllt
1	Funktionsüberprüfung Tafelerkennung statisch	Die Funktion der Tafelerkennung wird statisch überprüft und diese funktioniert einwandfrei (100 % Trefferwahrscheinlichkeit)	Erfüllt
2	Funktionsüberprüfung Nummererkennung mit statischer Tafelerkennung	Die Funktion der Nummererkennung wird überprüft (mittels statischer Tafelerkennung) und diese funktioniert einwandfrei (100 % Trefferwahrscheinlichkeit)	Erfüllt
3	Funktionsüberprüfung Startsignal statisch	Die Funktion Startsignalerkennung wird statisch überprüft und diese funktioniert einwandfrei (100 % Trefferwahrscheinlichkeit)	Erfüllt
4	Funktionsüberprüfung Tafelerkennung dynamisch langsam	Die Funktion der Tafelerkennung wird dynamisch mit einer Geschwindigkeit von $v = 0.1 \text{ m/s}$ überprüft und diese funktioniert einwandfrei (100 % Trefferwahrscheinlichkeit)	Erfüllt
5	Funktionsüberprüfung Nummererkennung mit dynamischer Tafelerkennung langsam	Die Funktion der Nummererkennung wird überprüft (mittels dynamischer Tafelerekennung langsam) und diese funktioniert einwandfrei (100 % Trefferwahrscheinlichkeit)	Nicht erfüllt (80 %)
6	Funktionsüberprüfung Startsignal dynamisch langsam	Die Funktion Startsignalerkennung wird dynamisch mit einer Geschwindigkeit von $v = 0.1 \text{ m/s}$ überprüft und diese funktioniert einwandfrei (100 % Trefferwahrscheinlichkeit)	Erfüllt
7	Funktionsüberprüfung Tafelerkennung dynamisch schnell	Die Funktion der Tafelerkennung wird dynamisch mit einer Geschwindigkeit von $v = 0.7 \text{ m/s}$ überprüft und diese funktioniert einwandfrei (100 % Trefferwahrscheinlichkeit)	Nicht erfüllt (70 %)
8	Funktionsüberprüfung Nummererkennung mit dynamischer Tafelerkennung schnell	Die Funktion der Nummererkennung wird überprüft (mittels dynamischer Tafelerekennung schnell) und diese funktioniert einwandfrei (100 % Trefferwahrscheinlichkeit)	Nicht erfüllt (70 %)
9	Funktionsüberprüfung Startsignal dynamisch schnell	Die Funktion Startsignalerkennung wird dynamisch mit einer Geschwindigkeit von $v = 0.7 \text{ m/s}$ überprüft und diese funktioniert einwandfrei (100 % Trefferwahrscheinlichkeit)	Erfüllt
10	Funktionsüberprüfung Ablauf (State- Machine)	Der Ablauf (State- Machine) funktioniert bei allen Geschwindigkeiten und lässt kein Fehlverhalten beim Ablauf zu.	Erfüllt

Tabelle 13: Testplanbeschreibung

4 Bewertung

Das Konzept aus PREN1 konnte in den meisten Punkten im PREN2 so wie geplant umgesetzt werden. Nötige Änderungen sind in diesem Kapitel beschrieben und begründet. Einige Sachen mussten angepasst werden, andere wurden aus Zeitgründen weggelassen. Die weggelassenen Teilkonzepte sind allerdings alle nicht zwingend notwendig um die Aufgabenstellung zu erfüllen.

4.1 Elektronik

In diesem Kapitel wird das Konzept für die Elektronik bewertet und allfällige Änderungen beschrieben.

4.1.1 Bewertung

Ausser den Änderungen, welche in Kapitel 4.1.2 beschrieben sind konnte das Konzept wie geplant umgesetzt werden. Die Teilkonzepte für die Aktoren und Sensoren konnten erfolgreich umgesetzt werden.

4.1.2 Änderungen

Weggelassen wurden die Strommessung und der Parksensor. Eine weitere Anpassung ist das Prinzip wie der Quadraturencoder für den Antrieb ausgewertet wird und daraus die Geschwindigkeit bestimmt wird. Auch wurde eine Änderung für den Quadraturencoder für den Schwenker durchgeführt, damit auch erkannt wird wenn sich der Schwenker rückwärts dreht.

Im Folgenden sind die Gründe dafür erläutert.

Strommessung

Die Strommessung ist auf der ersten Version des PCB nicht möglich, da ein Fehler bei der Pinbelegung des Operationsverstärkers ist. Dieser Fehler konnte aus Zeitgründen nicht mehr korrigiert werden und da die Information zum Strom nicht nötig ist um die Aufgabe zu erfüllen fiel der Entscheid dieses Teilkonzept ganz zu verwerfen.

Parksensor

Die Distanz zum Haltesignal kann direkt über die Auswertung der Kamera bestimmt werden. Dies reduziert den Hardwareaufwand für die Elektronik und Mechanik.

Quadraturencoder Antrieb

Das im PREN1 angedachte Konzept basierte darauf die Zeit zwischen zwei Impulsen zu messen und daraus die Geschwindigkeit zu bestimmen. Dies stellte sich in der Umsetzung jedoch als zu wenig zuverlässig heraus. Vor allem im Zusammenhang mit der Regelung (beschrieben in Kapitel 3.6.3) wurde es mit der unzuverlässigen Auswertung des Encoders unmöglich eine konstante Geschwindigkeit zu halten. Daher wurde das Prinzip angepasst, dass die Impulse des Encoders über eine bestimmte Zeit gezählt werden und daraus die Geschwindigkeit bestimmt wird (siehe Kapitel 3.6.3).

Quadraturencoder Schwenker

In den ersten Tests kam es vor, dass der Schwenker während des Einfahrens wieder ein wenig rückwärts rutschte. Mit dem Angedachten Prinzip zum Zählen der Impulse an einem Kanal kann aber die Richtung nicht festgestellt werden und ein Impuls rückwärts wird gleich interpretiert wie ein Impuls vorwärts. Dies verfälscht die Bestimmung der Position und macht es unmöglich sicherzustellen, dass der Schwenker die richtige Position erreicht. Um zu erkennen in welche Richtung sich der Schwenker dreht wurde der zweite Kanal des Quadraturencoders auf einen Pin vom Mikrocontroller verbunden. Durch das Auslesen dieses Pins bei einer positiven Flanke am anderen Kanal kann die Drehrichtung bestimmt werden. (siehe Abbildung 20 auf Seite 19) Mit dieser Information kann also sichergestellt werden, dass der Schwenker weit genug dreht.

4.2 Informatik

Tafel- & Nummererkennung

Das im PREN 1 erstellte Konzept im Bereich der Tafel- & Nummererkennung musste während PREN 2 im Bereich der Hardware sowie im Bereich der Software Anpassungen vorgenommen werden. Wie im Kapitel 3.13 beschrieben, musste im Bereich der Hardware auf einen rechenstärkeren Raspberry PI 3 A+ gewechselt werden. Dank der Budgetreserve und der minimalen Hardwareanpassung konnte dies ohne Probleme vorgenommen werden. Im Bereich der Software wurde bei der Nummererkennung auf die Software Tesseract OCR gewechselt. Dies aus den Gründen der Einfachheit und der Effizienz.

Akustik

Der Buzzer wurde bereits in Pren1 getestet, dementsprechend ist das Ergebnis gemäss den Erwartungen. Mit der kompakten Struktur und dem geringen Stromverbrauch ist er eine ideale Komponente für Projekte dieser Größenordnung.

Beschleunigungssensor

Die Messung der Beschleunigung erfolgt wie erwartet. Bei der Umrechnung herrschen noch kleinere Unsicherheiten bezüglich der Genauigkeit. Der Beschleunigungssensor findet leider keine Anwendung im endgültigen Produkt, weil wir bereits die Geschwindigkeitsdaten des Motors auswerten.

WebApp

Die WebApp wurde in PREN2 entworfen und entwickelt, somit gibt es keine Erwartungen an diese Komponente. Das Endergebnis entspricht ziemlich genau unseren Vorstellungen.

Würfelaufnahme

Das Grundkonzept wurde in PREN1 definiert und umgesetzt. Für das endgültige Produkt wurden kleine Änderungen vorgenommen. Der Verlauf der Kurvenscheibe wurde angepasst und die Führung des Würfels optimiert. Nun kann der Aufbau den Anforderungen gerecht werden.

4.3 Maschinentechnik

Teststrecke

Die grösste Herausforderung an das Fahrwerk besteht bei einer Doppelkurve mit engem Radius (siehe oben Mitte in Abbildung 41). Ansonsten erfüllt die Lokomotive die Anforderungen an die Strecke. Sie berührt das Lichtraumprofil und kein anderes Element auf der Teststrecke, mit einer Ausnahme, dem Holzwürfel. Ein Problem für die Mechanik stellt die enge Doppelkurve dar, da dadurch grosse Kräfte an der Lokomotive entstehen, welche aber durch die Kurvenführung weitgehend behoben wird.



Abbildung 41: Aufbau Teststrecke

Antriebswagen

Der Antriebswagen (siehe Abbildung 42) wie auch der Führungswagen sind im Allgemeinen wie konzipiert und konstruiert realisiert. In der Umsetzung wurden beim Antriebswagen zusätzlich zwei weitere Schleifkontakte für die bessere Gewährleistung der Stromabnahme und ein Vorrichtung für die Kurvenführung angebracht. Die Grösse und Form der Räder (siehe Abbildung 43) wurde etwas angepasst, da die Gefahr bestand, dass die Lokomotive an den Gleisen hängen bleibt. Die genauen Änderungen der Räder werden im nächsten Absatz ausführlicher vorgestellt.

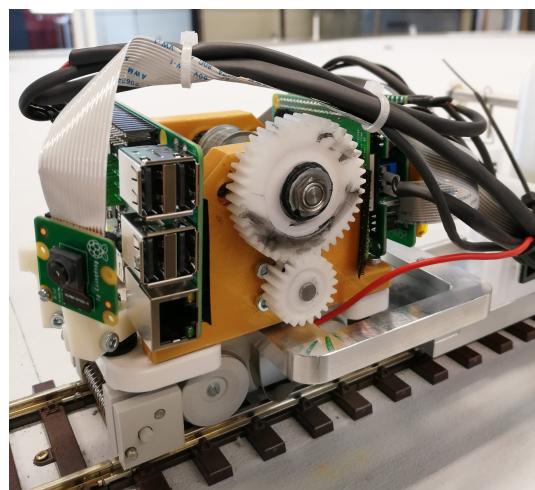


Abbildung 42: Antriebswagen

Beschleunigung und Geschwindigkeit

Bei den Probefahrten auf der vorgegebenen Teststrecke (siehe Abschnitt "Teststrecke") hat sich eine maximale Beschleunigung von 1 Meter pro Sekunde im Quadrat herausgestellt. Diese ergibt sich durch das Moment welches über das Getriebe auf die Räder übertragen wird, die Normalkraft des Zuges und der Reibwert zwischen Räder und Gleis. Dieser Wert entstand durch Messungen mit einem Beschleunigungssensor. Der berechnete Wert lag 2.94 Meter pro Sekunde im Quadrat. Die Differenz lässt sich durch den abweichenden Reibwert und die Masse erklären.

Gemäss den Messungen auf der Teststrecke ist eine maximale Geschwindigkeit von 2.2 Meter pro Sekunde erreichbar. Der limitierende Faktor ist nicht das Kippen beziehungsweise entgleisen in der Kurve, sondern der Motor. Der Motor erreicht dieser Geschwindigkeit seine maximale Leistung mit den gegebenen Werten des Stromes und der Spannung. Somit fährt die Lokomotive 1.7 Meter pro Sekunde schneller als im Anforderungskatalog deklariert wurde und etwa 0.7 Meter pro Sekunde schneller als der theoretisch berechnete Wert. An Hand dieser Ergebnisse ist ersichtlich, dass der Schwerpunkt optimaler liegt, als angenommen wurde und zusätzlich die Kurvenvorrichtung dazu beiträgt, dass die Lokomotive in der Kurve schneller fahren kann.

Wagenräder

Die Räder der Lokomotive sind für zwei Bereiche vorgesehen. Einerseits gibt es Räder für den Antriebswagen und andererseits solche für den Führungswagen. Die Räder für den Führungswagen haben im Gegensatz zu denen des Antriebswagens keine spezielle Bearbeitung. In einem ersten Versuch wurden die Reifen mit einem O-Ring versehen. Da der Anpressdruck durch das Gewicht der Lokomotive zu gering war und die Fuge nicht gleichmässig mit der Gummimasse gefüllt wurde, kippte der Zug leicht auf den Gleisen hin und her, da zusätzlich der Absatz für die Führung zu kurz war. In einem nächsten Schritt wurde ein Gummispray Schicht für Schicht auf die Räder gesprayt. Jedoch weissen diese bei der Testfahrt bereits grossen Verschleiss auf und ist somit für die Forderung nicht ausreichend. Als letzter und funktionsfähiger Versuch wurden die Räder mit einem speziellen Gummiband umwickelt. Die Funktion ist erfüllt und der Verschleiss hält sich in einem vernünftigen Rahmen. Bei den Antriebsrädern wie auch bei den Führungsrädern ist der Durchmesser etwas grösser und die Kontur leicht angepasst.

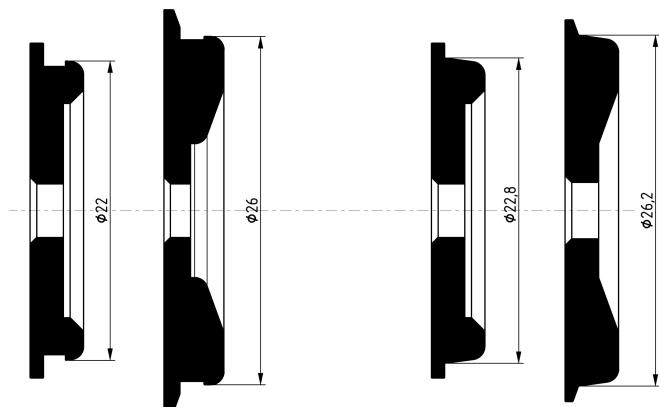


Abbildung 43: Räder Antriebswagen bzw. Führungswagen - Alt vs. Neu

Stromabnahme

In einem ersten Versuch wurde die Stromabnahme nur an einem Ort, dem Führungs-wagen, abgenommen. Jedoch ist die Gefahr gross, dass die Schleifkontakte bereits bei kleinsten Unebenheiten der Strecke den Kontakt verlieren und das System zum Absturz bringen. Somit wurden zusätzlich zwei Schleifkontakte am Antriebswagen befestigt, welche die Gefahr minimiert. Die Stromabnahme funktioniert nun mit der gewünschten Zuverlässigkeit.

Kurvenführung

Die Kurvenführung wurde in mehreren Testfahrten getestet und zeigte die gewünschten Effekte auf. Lediglich die Feder beziehungsweise die Federrate musste angepasst werden, damit der Anpressdruck der Stahlstifte genügend hoch ist und so der Zentripetalkraft der Lokomotive entgegenwirken kann.

5 Bedienungsanleitung

5.1 Inbetriebnahme Mechanik

Die Bedienung der Lokomotive "Soultrain" kann in vier einfachen Teilschritten vorgenommen werden:

1. Lokomotive auf die Gleise stellen

Die Lokomotive soll innerhalb des Startbereichs auf die Gleise gestellt werden und korrekt auf allen Rädern aufliegen. Es ist darauf zu achten, dass die Schleifkontakte die Gleise richtig berühren und die nötige Vorspannung aufweisen.

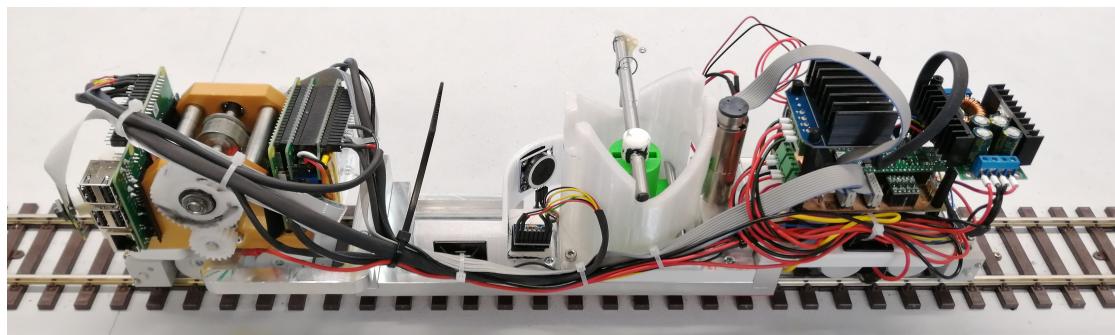


Abbildung 44: Lokomotive auf die Gleise stellen

2. Kurvenvorrichtung hinten und vorne von der Lokomotive einhakken

Die Zapfen an beiden Enden müssen hineingedrückt werden, um die Vorrichtung korrekt in die Gleise einzuhacken. Es ist darauf zu achten, dass die Kontur der Stahlstifte korrekt an derer der Gleise ausgerichtet ist.

Dieser Arbeitsschritt muss vorne wie auch hinten an der Lokomotive ausgeführt werden.

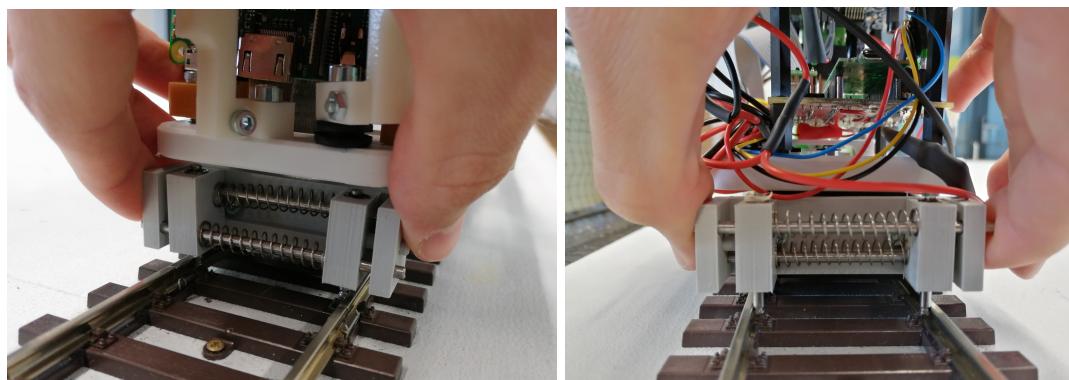


Abbildung 45: Kurvenvorrichtung vorne und hinten von der Lokomotive einhakken

3. Kranhebel in die Ausgangslage bringen

In diesem Teilschritt soll sichergestellt werden, dass der Kranhebel ausgefahren und an der Startposition platziert ist. Dafür drückt man den Hebel, welcher in Richtung mittelpunkt der Teststrecke zeigt, an den Anschlag unten.

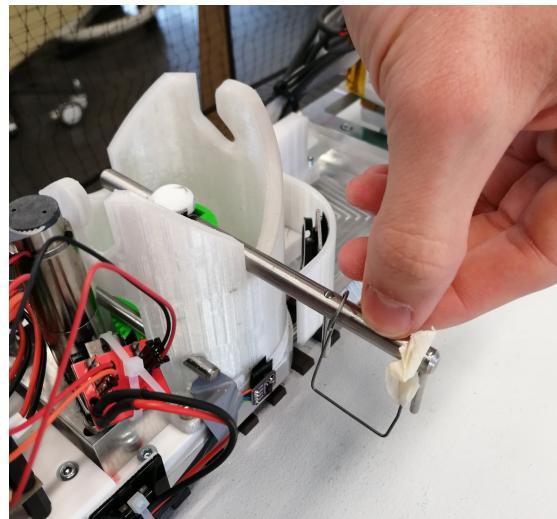


Abbildung 46: Lokomotive auf Gleis

4. Kontrolle und Freigabe des Startvorgangs

Am Ende soll nochmals kontrolliert werden, ob die Lokomotive korrekt auf den Rädern aufliegt, alle Schleifkontakte die Gleise berühren und die Kurvenvorrichtung korrekt eingehackt ist. Anschliessend kann die Freigabe für den Startvorgang erfolgen.

5.2 Bedienungsanleitung Elektronik und WebApp

In dieser Bedienungsanleitung werden folgende Begriffe verwendet:

Raspi	Raspberry 3 B+
SSH	Secure Shell. Ein Protokoll, welches sichere Verbindungen über das Netzwerk ermöglicht.
Soultrain	Unser in PREN1&2 entwickelte Zug/Lokomotive

Für eine erfolgreiche Inbetriebnahme werden folgende Komponenten benötigt:

SSH Client	Wir empfehlen PuTTY.
Laptop	Oder Computer mit Hotspot Funktionalität
Soultrain	Mit allen Komponenten.
Teststrecke	Ideal mit Nummernschildern, Lichtschränken, Kurven etc.
(Optional)Python 3.4+	Für den Start vom WebServer auf der lokalen Maschine

Inhalt dieser Anleitung setzt einen voll Funktionsfähigen Soultrain voraus. Raspi und dazugehörige Komponenten liegen bereits korrekt installiert und konfiguriert vor. Der Fokus liegt auf Benutzermaschine und den Befehlen für eine korrekte Inbetriebnahme.

5.2.1 Inbetriebnahme

Stromquelle Der Raspberry Pi 3 B+ besitzt keine eigene Stromquelle und muss somit über eine externe Quelle gespeist werden. Hierfür kennen wir 3 Möglichkeiten:

- Anschluss der Schleifkontakte an einen Power Supply
- Aufgleisung des Zuges auf eine gespeiste Schiene (siehe Bedienungsanleitung MT)
- Den Raspi über Micro USB speisen. Hierbei kann der Zug nicht betrieben werden, sondern nur der Raspi.

Verbindung

Ist die Stromversorgung zum Raspi gewährleistet kann man auf diesen über eine SSH Verbindung zugreifen. Hierfür erstellen wir einen WLAN Hotspot.

- Netzwerk und Interneteinstellungen öffnen
- Reiter 'Mobiler Hotspot' wählen
- 'Bearbeiten' klicken
 - Netzwerkname = 'team28master'
 - Netzwerkennwort = 'team28master'
 - 'Speichern' klicken
- Ganz oben den Schalter unterhalb von 'Mobiler Hotspot' einschalten

Wenn wir alles richtig gemacht haben sollte nun folgendes Fenster erscheinen:



Abbildung 47: WLAN Hotspot

Nach einigen Sekunden sollte ein Eintrag mit Informationen über den Raspi erscheinen.

Die IPv4 Adresse (Format ähnlich wie '192.168.1.12') in den Zwischenspeicher kopieren (Ctrl + C). Die IP-Adresse werden wir später noch verwenden, speichert diese in einem Textdokument (Z.B. Word).

Mithilfe der erhaltenen IPv4 Adresse eine Verbindung mit Putty und SSH erstellen:

- Starte PuTTY
- Host Name (or IP address) = <IPv4> (Ctrl + V)
- Port = '22'
- Connection type = 'SSH'
- 'Open' klicken

Wenn wir alles richtig gemacht haben sollte nun folgendes Fenster erscheinen:

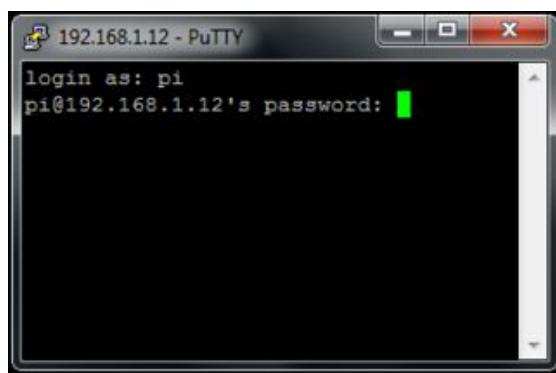


Abbildung 48: Raspi Login Maske

- login = 'pi'
- password = 'team28'

Raspberry Pi

Von unserem Home Verzeichnis navigieren wir weiter zum Code:

```
cd team28/src/raspi
```

Von hier aus haben wir verschiedene Möglichkeiten die Applikation zu starten.

Applikation

Die Applikation besteht aus mehreren Modulen, welche miteinander kommunizieren. Jede dieser Module kann eigenständig ein- und ausgeschaltet werden. Zum Beispiel können wir das 'Akustik' Modul folgendermassen starten:

```
python3 /acoustic/sound.py
```

Gestoppt wird das Modul mit (Ctrl + C).

Für ein funktionierendes Zusammenspiel der jeweiligen Module, benötigen wir jedoch alle Module gleichzeitig. Hierfür haben wir das Bash Skript 'run-master.sh'.

```
./run-master.sh
```

Das Skript startet zudem einen WebServer, über welchen wir die Applikation bedienen können.

Web Server

Der WebServer wird auf dem Port :2828 gestartet und kann jederzeit über die IP-Adresse des Raspi aufgerufen werden.

- Browser starten (Firofox, Chrome, IE ...)
- In URL-Leiste folgendes eingeben: <IPv4>:2828/

Wenn wir alles richtig gemacht haben sollte nun folgendes Fenster erscheinen:

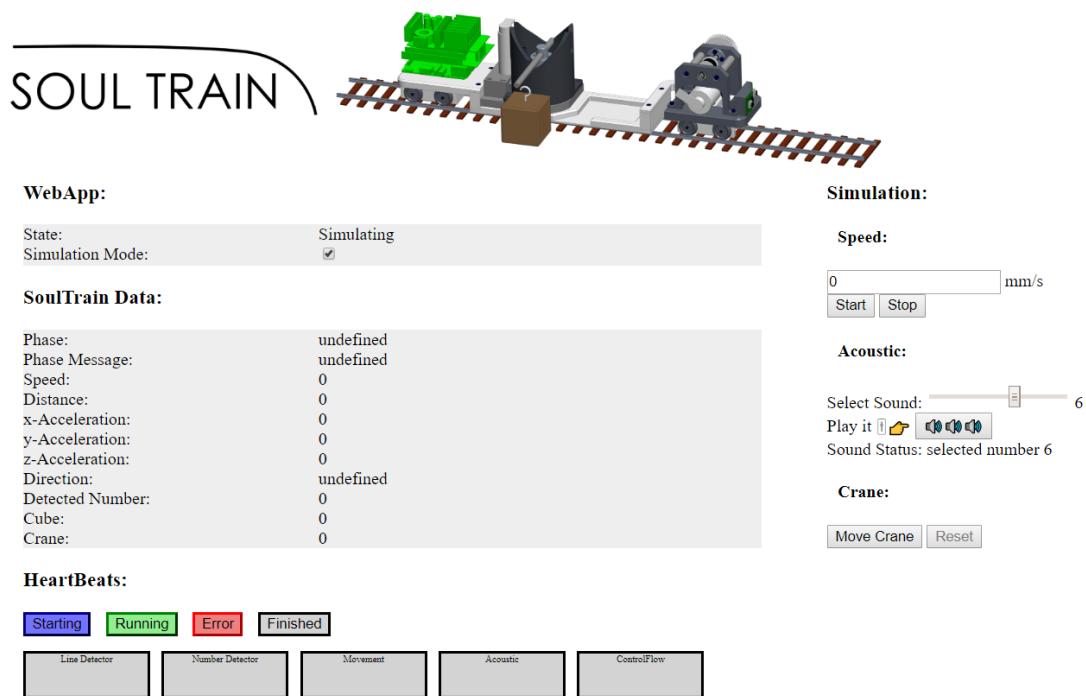


Abbildung 49: Web Client

5.2.2 Web Client

Der Web Client ist in 3 Teilbereiche aufgeteilt.

- Übersicht
- Simulation
- Start

Zwischen den beiden Bereichen 'Simulation' und 'Start' kann über die Checkbox 'Simulation Mode' in der 'Übersicht' gewechselt werden. **Übersicht**

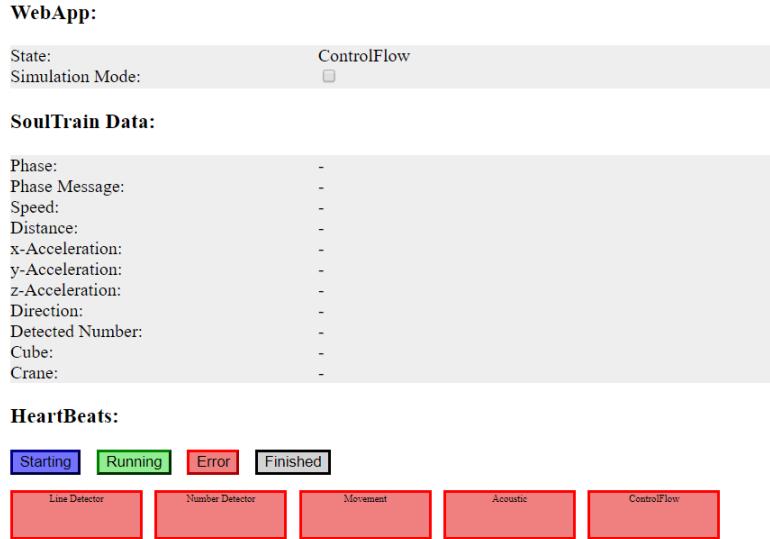


Abbildung 50: Übersicht

In der Übersicht werden die wichtigsten Daten in Echtzeit abgebildet. Die Daten repräsentieren Zug, Status des Zuges und dessen Umgebung. Man unterscheidet zwischen den Soultrain Daten und dem Hearbeat. Soultrain Daten:

Phase	Die momentane Etappe des Zuges
Phase Message	Nachricht welche die Phase repräsentiert
Speed	Geschwindigkeit, mit welcher der Zug unterwegs ist
Distance	Distanz, welche der Zug zurückgelegt hat
x-Acceleration	Beschleunigung auf der x-Achse
y-Acceleration	Beschleunigung auf der y-Achse
z-Acceleration	Beschleunigung auf der z-Achse
Direction	Voraussichtliche Richtung der Gleise
Direction Number	Nummer, welche die Richtung repräsentiert
Cube	Würfel Status
Crane	Kran Status

Hearbeat:

Jedes Modul besitzt einen Hearbeat, welches in regelmässigen Abständen gesendet wird. Ein Modul kann folgende Zustände annehmen:

Starting	Während der Initialisierungsphase des Moduls
Running	Solange das Modul läuft
Error	Sobald ein Fehler gemeldet wird oder das Modul keinen Hearbeat mehr sendet
Finished	Nach Abschluss seiner Tätigkeit

Folgende Module werden überwacht:

Line Detector	Liest die Richtung der aufkommenden Gleise während der Fahrt
Number Detector	Liest die Nummern während der Fahrt
Movement	Liest Geschwindigkeit und regelt die momentane Geschwindigkeit anhand dessen
Acoustic	Gibt die eingelesene Nummer akustisch wieder
Control Flow	Regelt den Ablauf der verschiedenen Phasen

Simulation

Simulation:

Speed:

0	mm/s
Start	Stop

Acoustic:

Select Sound: 6

Play it 

Sound Status: selected number 6

Crane:

Move Crane	Reset
------------	-------

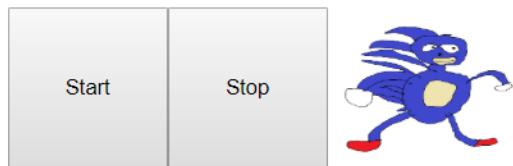
Abbildung 51: WebApp

Über die Simulation kann man Komponenten während der Laufzeit testen. Folgende Komponenten kann man testen

Speed	Geschwindigkeit mit welcher der Zug fahren soll
Acoustic	Zahl, welche die Akustische Komponente wiedergiebt
Crane	Die Kranbewegung starten

Start

ControlFlow



- Find Cube:
- Grab Cube:
- Round One:
- Round Two:
- Find Stop:
- Stopping:

Abbildung 52: Start

Start und Stop des Soultrain Ablaufs. Ausgeführt, werden nur Phasen welche gecheckt sind.

6 Schlussdiskussion

6.1 Kosten

Gemäss der Aufgabenstellung steht ein Budget von 500 Franken zur Verfügung. In Tabelle 14 wurden alle Komponenten in einer Kostenübersicht zusammengetragen.

In dieser Übersicht nicht enthalten sind Komponenten, welche aus der Mechanikwerkstatt oder dem Elektronik Labor der HSLU bezogen werden können. Auch sind Stundenan- sätze für Werkstattpersonal oder 3D-Drucker nicht berücksichtigt.

Beschreibung	Lieferant	Anzahl	Stückpreis	Gesamtpreis
Raspberry Pi 3 Model B+	pi-shop	1	CHF 39.00	CHF 39.00
Raspberry Pi Zero W	pi-shop	1	CHF 10.80	CHF 10.80
Raspberry Pi Kamera	pi-shop	2	CHF 32.90	CHF 65.80
Passiver Buzzer	play-zone	1	CHF 5.00	CHF 5.00
StromPi 3	Conrad	1	CHF 49.95	CHF 49.95
StromPi 3 Battery Pack, 1000 mAh	Conrad	1	CHF 31.95	CHF 31.95
Beschleunigungssensor ADXL345	aliexpress	1	CHF 1.00	CHF 1.00
Tiny K22	hslu	1	CHF 26.00	CHF 26.00
Arduino IBT_2 (DC-Motoren Treiber) (43A)	wish	1	CHF 9.00	CHF 9.00
DC-DC CC CV Buck Converter (12A)	wish	1	CHF 3.00	CHF 3.00
Ultraschallsensor HS-SR04	aliexpress	1	CHF 0.77	CHF 0.77
TOF Sensor VL53L0X	aliexpress	1	CHF 3.69	CHF 3.69
LM2596 DC-DC Schritt-down Converter (2.5A)	aliexpress	1	CHF 0.90	CHF 0.90
Logic-Level-Converter Bi-Directional Modul 5 v zu 3,3 v	aliexpress	1	CHF 0.90	CHF 0.90
Tower Pro Micro Servo SG90	wish	1	CHF 1.60	CHF 1.60
Mini Modul PWM Speed Control Über L298N	aliexpress	1	CHF 0.69	CHF 0.69
Antriebsmotor mit Encoder	Sponsor	1	CHF 30.00	CHF 30.00
Schwenker Motor mit Encoder	Sponsor	1	CHF 25.00	CHF 25.00
Zahnriehmen	Mädler	2	CHF 13.19	CHF 26.38
Zahnriehmenrad	Mädler	3	CHF 8.00	CHF 24.00
Kupplungen	Mädler	2	CHF 30.00	CHF 60.00
Zahnräder	Mädler	1	CHF 14.35	CHF 14.35
Zahnräder	Mädler	1	CHF 3.25	CHF 3.25
Zahnräder	Mädler	1	CHF 4.77	CHF 4.77
Total		29	CHF 345.71	CHF 437.80

Tabelle 14: Kostenübersicht Gesamtkonzept

6.2 Zeitlicher Entwicklungsaufwand

Die im Stundenplan gegebenen zwei Halbtage wurden vom Team genutzt um die Entwicklung des Projekts voranzutreiben. Zusätzlich wurden von allen Teammitgliedern diverse Abende und Wochenende investiert um die Ziele zu erreichen. Eine genau Stundenzusammenstellung wurde über die zwei Semester nicht geführt. Daher wird hier darauf verzichtet eine genaue Anzahl Stunden für den Entwicklungsaufwand zu nennen.

6.3 Lessons learned

Im Rahmen beider PREN Module konnten im ganzen Team viele Wertvolle Erfahrungen gesammelt werden. Es konnte viel in der jeweils eigenen Disziplin gelernt werden, jedoch auch im interdisziplinären Kontext konnten alle Teammitglieder viel Lernen. Durch die enge Zusammenarbeit der verschiedenen Disziplin konnte das Verständniss über die Arbeit der jeweils anderen Disziplin gestärkt werden.

6.3.1 Elektrotechnik

In der Elektronik findet man sich oft als Schnittstelle zwischen der abstrakten Softwarewelt der Informatik und der physikalischen Mechanik. Dieses Projekt konnte gut aufzeigen, wie eine ständige Kommunikation mit allen Teammitgliedern unerlässlich ist. Dies erfordert auch eine saubere Planung um einen möglichst fließenden Projektablauf sicherzustellen. Zum Beispiel sollte die Motorensteuerung bereit sein um die Funktionsweise der Mechanik zu Testen sobald diese produziert ist, andererseits sollte z.B. die Kommunikation vom Mikrocontroller zum Pi bereit sein, sobald die Informatik erste Software-Komponenten bereit hat.

Die Unerlässlichkeit dieser interdisziplinären Planung und Kommunikation ist wohl die wichtigste und grösste Lektion aus den PREN Modulen.

6.3.2 Informatik

Die Informatik ist der «Leim», welcher schlussendlich den Zug zum fahren bringen soll. Viele der Module wie Nummererkennung, Akustik oder Kommunikation zwischen Raspberry PI und Tiny konnten schon während PREN1 und auch am Anfang des PREN2 getestet werden. Doch der Ablauf oder die Stabilität der Algorithmen kann erst am Schluss getestet werden. Obwohl der Zug relativ früh in PREN2 bereit war, haben wir den Aufwand der verschiedenen «Integrationstests» der Module unterschätzt. Hier haben wir sicherlich gelernt, früher die verschiedenen Tests disziplinierter und ausführlicher durchzuführen.

Weiters muss auch während den Schlusstests mehr Reservezeit eingeplant werden. Dies lernten wir, da aufgrund von starken Abnutzungsscheinungen der Räder - neue Räder hergestellt werden mussten. Dies führte zu verzögerungen während den Schlusstests.

6.3.3 Maschinenbau

Im Maschinenbau wurde klar, dass eine gute Planung im Voraus einem sehr viel Ärger ersparen kann. Grundsätzlich wurde das von unserem Team auch so umgesetzt. Umso mehr Zeit man in der Konstruktion investiert, umso besser lässt sich das Ganze schlussendlich zusammenbauen und funktioniert optimaler weise auch. Ohne einige Anpassungen kamen auch wir nicht durch. Doch im Grossen und im Ganzen sind wir zufrieden mit unserer Vorarbeit. Schlussendlich kann man sagen, dass die Mechanik die Anforderungen mehr als erfüllt. Wie schon im Elektrotechnikteil erwähnt lohnt es sich früh mit den anderen Disziplinen abzusprechen und deren Wünsche probiert in der Konstruktion einzubauen.

6.4 Offene Punkte

Das Konzept aus PREN1 konnte genug gut umgesetzt werden damit das Team zuverlässig ist, dass der Zug die Aufgabenstellung erfüllen kann. Es gibt noch einige Punkte, die man als Erweiterung umsetzen könnte. Auf einige Punkte, die nicht umgesetzt werden konnten wird in Kapitel 4 eingegangen. Die Umsetzung dieser offenen Punkte wäre aber noch mit einem bedeutenden Zeitaufwand verbunden und dies ist leider im Rahmen dieses Moduls nicht mehr möglich.

6.5 Ausblick

Wie gut sich das Konzept bewährt wird sich bei einem Wettbewerb mit allen anderen Teams zeigen. Dabei müssen alle Teams mit ihrem Zug die Aufgabenstellung bewältigen und erhalten Punkte für jede erfüllte Teilaufgabe. Zusätzlich wird die gemessene Zeit mit allen Teams verglichen und bewertet.

7 Verzeichnisse

Abbildungsverzeichnis

1	Komponentendiagramm allgemein	5
2	Baugruppe	6
3	Links: Kran Planung / Rechts: Kran Umsetzung	7
4	Links: Kran Antriebsstrang / Rechts: Kurvenscheibe	7
5	Würfelaufnahme	8
6	DC Motoren	9
7	Konzept und Ergebnis des Fahrwerkes	10
9	Antriebswagen Endprodukt	11
10	Kurvenvorrichtung in unmontiertem und montiertem Zusatztand	11
11	Schwerpunkt der Lokomotive	13
12	Führwagen	14
13	Konzept des Führwagens	14
14	Elektrokomponenten	15
15	Schleifkontakte	15
16	Komponentendiagramm Elektronik	16
17	PCB mit TinyK22	17
18	Soul Train mit Elektronik	17
19	Kabelverbindungen Rückseite	18
20	Signalverlauf Encoder (www.maxonmotor.ch)	19
21	Distanzmessung mit TOF Sensor	20
22	Winkelmessung Schwenker	20
23	Modulaufteilung TinyK22	23
24	Flussdiagramm Task Scheudler	24
25	Ablauf mit Frame Handler	26
26	Ablauf für die Würfelerkennung	28
27	Allgemeines Prinzip eines PID-Reglers Wikipedia	29
28	Kommunikation	37
29	Verkabelung Beschleunigungssensor (http://fritzing.org)	39
30	Verkabelung Buzzer	42
31	Directory Tree	43
32	REST API	44
33	Ausschnitt der Anforderungsliste PREN 1	46
34	Ausschnitt der Anforderungsliste PREN 1	46
35	State- Machine Slave- RPI	47
36	Startsignal	49
37	Codesnipped: Filter Startsignal	49
38	Tafelerkennung	50
39	Codesnipped Tesseract	50
40	Architektur: Worker- Ablauf	51
41	Aufbau Teststrecke	55
42	Antriebswagen	55
43	Räder Antriebswagen bzw. Führwagen - Alt vs. Neu	56
44	Lokomotive auf die Gleise stellen	58
45	Kurvenvorrichtung vorne und hinten von der Lokomotive einhaken	58
46	Lokomotive auf Gleis	59
47	WLAN Hotspot	60

48	Raspi Login Maske	61
49	Web Client	62
50	Übersicht	63
51	WebApp	64
52	Start	65

Tabellenverzeichnis

1	Positionsnummern	11
2	Größen für die Beschleunigungsberechnung	12
3	Größen für die Geschwindigkeitsberechnung	13
4	Positionsnummern	14
5	Module Software TinyK22	22
6	Komponente Modul pi	25
7	Komponente Modul cube	26
8	Komponente Modul drive	28
9	Schlüsselwörter und mögliche Werte Kommunikation Pi \Leftrightarrow Tiny	32
10	Darstellung der Phasen für Slave- RPI	47
11	Filter für Startsignalerkennung	49
12	Worker Beschreibung	51
13	Testplanbeschreibung	52
14	Kostenübersicht Gesamtkonzept	66

Literatur

- Maxon. *DCX 32 L Katalogseite 86*, 2018a.
- Maxon. *DCX 19 S Katalogseite 78*, 2018b.
- Wikipedia. Pid controller - wikipedia. https://en.wikipedia.org/wiki/PID_controller. (Accessed on 05/16/2019).
- Pi-Shop. Raspberry pi - raspberry pi 3 model b+, 2018. URL <https://www.pi-shop.ch/raspberry-pi-3-model-b>.
- Pi-Shop. Raspberry pi - raspberry pi zero w. <https://www.pi-shop.ch/raspberry-pi-zero-w>, a. (Accessed on 12/22/2018).
- Pi-Shop. Raspberry pi - pi supply bright pi - bright white und ir kamera licht für raspberry pi | pi-shop.ch. <https://www.pi-shop.ch/pi-supply-bright-pi-bright-white-und-ir-kamera-licht-fuer-raspberry-pi>, b. (Accessed on 12/22/2018).
- Pi-Shop. Raspberry pi - original raspberry pi kamera module v2 | pi-shop.ch. <https://www.pi-shop.ch/raspberry-pi-kamera-module-v2>, c. (Accessed on 12/22/2018).
- Play-Zone. Play-zone.ch passiver buzzer / speaker, 3.3v. <https://www.play-zone.ch/de/passiver-buzzer-speaker-3-3v.html>. (Accessed on 12/22/2018).
- Reichelt. Rasp strom pi 3: Raspberry pi - der strompi 3 bei reichelt elektronik. https://www.reichelt.de/raspberry-pi-der-strompi-3-rasp-strom-pi-3-p232866.html?&trstct=pos_0, a. (Accessed on 12/22/2018).
- Reichelt. Rpi strompi akku: Raspberry pi - strompi 3 battery pack, 1000 mah bei reichelt elektronik. https://www.reichelt.de/raspberry-pi-strompi-3-battery-pack-1000-mah-rpi-strompi-akku-p232867.html?&trstct=pos_2, b. (Accessed on 12/22/2018).
- AliExpress. I43 1 stücke gy 291 adxl345 digital dreiachsen beschleunigung der schwerkraft tilt modul iic/spi übertragung auf lager in i43 1 stücke gy-291 adxl345 digital dreiachsen beschleunigung der schwerkraft tilt-modul iic/spi übertragung auf lager aus integrierte schaltungen auf aliexpress.com | alibaba group. https://de.aliexpress.com/item/Free-Shipping-GY-291-ADXL345-digital-three-axis-acceleration-of-gravity-tilt-module-IIC-32346306872.html?spm=a2g0x.search0104.3.8.3a781d56R05pDY&ws_ab_test=searchweb0_0,searchweb201602_5_10065_10068_319_10059_10884_317_10887_10696_100031_321_322_10084_453_10083_454_10103_10618_10307_538_537_536_10134,searchweb201603_51,ppcSwitch_0&algo_expid=a352fecc-60ea-4079-b651-cade58a1b700-1&algo_pvid=a352fecc-60ea-4079-b651-cade58a1b700, a. (Accessed on 12/22/2018).
- wish. Wish | halbleiter-motor-treiber auto bts7960 43a h-bruecke pwm-antrieb fuer arduino hot. <https://www.wish.com/product/5b97621dab410c29feae20c2>, a. (Accessed on 12/22/2018).
- wish. Wish | dc-dc cc cv buck converter step-down power module 7-32v to 0.8-28v 12a 300w. <https://www.wish.com/product/5976f5aad35dc104728e52d>, b. (Accessed on 12/22/2018).

AliExpress. 1 stücke shengyang hc sr04 zu welt ultraschall welle detektor bis hin modul für arduino abstand sensor in 1 stücke shengyang hc-sr04 zu welt ultraschall welle detektor bis hin modul für arduino abstand sensor aus sensoren auf aliexpress.com | alibaba group. <https://de.aliexpress.com/item/1pcs-HC-SR04-to-world-Ultrasonic-Wave-Detector-Ranging-Module-for-arduino-Distance-Sensor-Module-10m-100m-1000m.html?spm=a2g0s.9042311.0.0.25dd4c4dbC08tr>, b. (Accessed on 12/22/2018).

AliExpress. Gy 530 vl53l0x welt kleinste zeit o f flug (tof) laser ranging sensor in gy-530 vl53l0x welt kleinste zeit-o f-flug (tof) laser ranging sensor aus sensoren auf aliexpress.com | alibaba group. <https://de.aliexpress.com/item/GY-530-VL53LOX-World-smallest-Time-o-f-Flight-ToF-laser-ranging-sensor-32773126352.html?spm=a2g0s.9042311.0.0.25dd4c4dbC08tr>, c. (Accessed on 12/22/2018).

AliExpress. 2 stücke lm2596 dc dc schritt down converter power supply module lm2596s einstellbare buck step down power module spannung regler in 2 stücke lm2596 dc-dc schritt-down converter power supply module lm2596s einstellbare buck step-down power module spannung regler aus wechselrichter & konverter auf aliexpress.com | alibaba group. <https://de.aliexpress.com/item/2-pcs-LM2596-Step-Down-Power-Module-LM2596S-DC-DC-3A-Adjustable-Buck-Step-Down-Power-32802169864.html?spm=a2g0s.9042311.0.0.25dd4c4dbC08tr>, d. (Accessed on 12/22/2018).

Distrelec. 13fr200e | kaufen strommesswiderstand | ohmite | distrelec. <https://www.distrelec.ch/de/strommesswiderstand-ohmite-13fr200e/p/30046857>. (Accessed on 12/22/2018).

5 stücke iic i2c uart spi logic level converter bi directional modul 5 v zu 3,3 v in 5 stücke iic i2c uart spi logic-level-converter bi-directional modul 5 v zu 3,3 v aus integrierte schaltungen auf aliexpress.com | alibaba group. <https://de.aliexpress.com/item/5PCS-IIC-I2C-UART-SPI-Logic-Level-Converter-Bi-Directional-Module-5V-to-3-3V-For-32826629975.html?spm=a2g0s.9042311.0.0.25dd4c4dbC08tr>. (Accessed on 12/22/2018).

wish. Tower pro micro servo sg90 | wish. <https://www.wish.com/search/Tower%20Pro%20Micro%20Servo%20SG90/product/57c8e0946f5941265132b849>, c. (Accessed on 12/22/2018).

Mädler. Zahnriemen profil at 5, breite 10 mm - mädler webshop. <https://www.maedler.ch/product/1643/1616/963/zahnriemen-profil-at-5-breite-10-mm>, a. (Accessed on 12/22/2018).

Mädler. Zahnriemenräder at5 für riemenbreite 10 mm - mädler webshop. <https://www.maedler.ch/product/1643/1616/996/zahnriemenraeder-at5-fuer-riemenbreite-10-mm>, b. (Accessed on 12/22/2018).

Mädler. Spannsätze sig, rostfrei, bohrung 4 bis 40mm - mädler webshop. <https://www.maedler.ch/product/1643/1621/spannstaetze-sig-rostfrei-bohrung-4-bis-40mm>, c. (Accessed on 12/22/2018).

Mädler. Stirnzahnräder kunststoff pom schwarz, gefräst, modul 1 - mädler webshop. <https://www.maedler.ch/product/1643/1618/1034/2545>

[stirnzahnraeder-kunststoff-pom-schwarz-gefraest-modul-1](#), d. (Accessed on 12/22/2018).