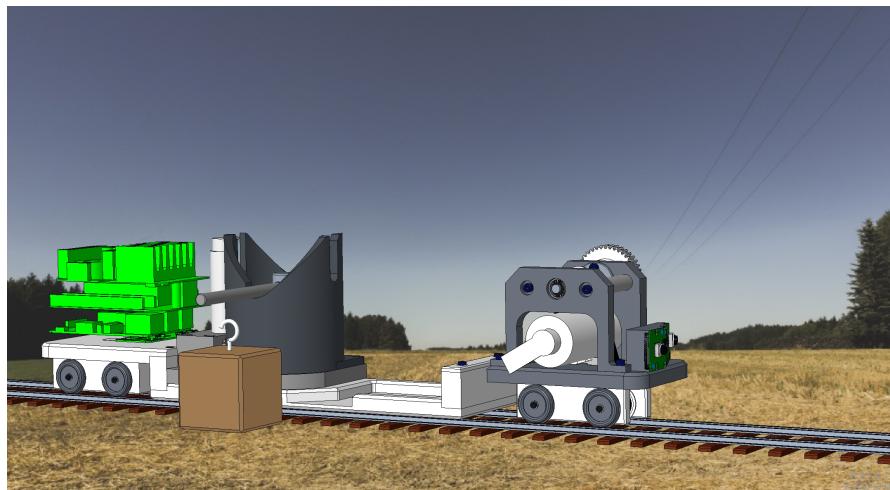


TA.BA\_PREN1.H1801

# Schlussdokumentation PREN1

## Gruppe 28

A. Rebsamen, J. Grepper, M. Omlin,  
M. Schöni, P. Marty, S. Ineichen



December 14, 2018, Luzern

## **Inhaltsverzeichnis**

# **1 Management Summery**

## **1.1 Management Summery**

Im Rahmen des Moduls PREN (Produktenwicklung) an der Hochschule Luzern wurde dieses Dokument durch das Team 28 erstellt. Das interdisziplinäre Team besteht aus Maschinenbau-, Elektrotechnik- und Informatikstudenten. In dieser Durchführung des Moduls wird in der Aufgabenstellung eine Entwicklung eines Schnellzuges gefordert. Dieser Zug muss eine definierte Strecke mit Geraden und Kurven in zwei Runden so schnell wie möglich passieren. Dabei muss der Zug einen Holzwürfel, welcher in der Startzone mittels einem Kran 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. Die erkannte Nummer soll akustisch ausgegeben werden. Der Zug muss autonom, also benutzerunabhängig, agieren. Im Rahmen des PREN1 wurden mit Technologierecherchen mehrere Lösungsvarianten erarbeitet. Aus diesen Lösungsvarianten wurde dann die optimale Kombination von Teilkomponenten ausgewählt und zu einem Gesamtlösungskonzept zusammengeführt.

Dieses Konzept beinhaltet eine mechanische Grundkonstruktion mit zwei Trägerwagen. Diese mechanische Grundkonstruktion wurde so entwickelt, dass der Schwerpunkt tief gehalten wird und somit hohe Kurvengeschwindigkeiten erreicht werden können. Angetrieben wird der Schnellzug mit einem DC- Motor mit eingebauter Encoderscheibe um präzise Geschwindigkeiten zu erreichen. Der Kran um den Würfel auf den Zug zu heben ist eine eigene Hub- Konstruktion mit DC- Motor. Die Signal- und Nummererkennung wird mittels Kamera und Machine- Learning Algorithmen realisiert. Als zentrale Recheneinheit kommt ein Raspberry PI 3+, welcher einem Raspberry PI Zero und einem Mikrocontroller unterstützt wird. Im PREN2 wird das entwickelte Konzept realisiert, getestet und optimiert.

## 2 Einleitung, Zielsetzung

### 2.1 Einleitung

In dieser Arbeit liegt das Gesamtkonzept für einen autonomen Schnellzug vor. Dieses Konzept wurde im HS18 im Rahmen des PREN1 entwickelt und im FS19 in PREN2 realisiert. Die Aufgabe besteht darin einen Schnellzug zu entwickeln, welcher so schnell wie möglich eine definierte Strecke mit Geraden und Kurven so schnell wie möglich zurücklegt. Im Startbereich muss der Zug mittels am Zug befestigten Konstruktion einen Holzwürfel auf den Zug transportieren. Danach muss der Zug durch eine Lichtschranke die Strecke in zwei Runden absolvieren. Dabei muss er ein seitlich befindendes Signal mit Nummer erkennen, welche die Halteposition verrät. Diese Halteposition muss dann in der dritten Runde angefahren werden und so nahe wie möglich angehalten werden. Die erkannte Nummer soll auch mittels akustischen Signal ausgegeben werden. Der Zug wurde mit folgenden Schwerpunkten entwickelt:

- Kompaktheit
- Einfachheit
- Gewicht
- Robustheit
- Kosten

Im Hauptteil dieser Arbeit wird das Konzept des Zuges beschrieben. Die Beschreibung ist aufgeteilt in die drei Fachgebiete Maschinenbau, Elektrotechnik und Informatik. Im Maschinenbau 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 Akustikausgabe und der Softwareaufbau beschrieben. Berechnung für die erwartete maximale Geschwindigkeit und die durchgeführten Versuchsaufbauten sind im hinteren Teil der Arbeit beschrieben.

In diesem Konzept wurde wie oben erwähnt mit Schwerpunkten wie Kompaktheit, Einfachheit und niedrigem Gewicht entwickelt um eine gute maximale Geschwindigkeit mit dem Zug zu erreichen. Dabei soll aber auch ein Schwergewicht auf Robustheit und Prozesssicherheit gelegt werden. Zusätzliche Optimierungen besonders in der maximalen Geschwindigkeit können während dem PREN2, der Realisierungsphase, erzielt werden.

### **3 Lösungskonzepte**

#### **3.1 Lösungskonzepte**

## 3.2 Ablauf

Dieses Kapitel gibt eine Übersicht über den Gesamten Ablauf, der nötig ist um die Aufgabenstellung zu erfüllen. Der Ablauf ist in ?? Grafisch dargestellt. Der Ablauf kann in fünf Zustände unterteilt werden.

- Initialisierung
- Würfelaufnahem
- Hochgeschwindigkeitsfahrt
- Parkplatzsuche
- Parkieren

Im folgenden werden die Zustände kurz beschrieben.

### **Initialisierung**

Alle Systeme werden gestartet sobald sie mit Strom versorgt werden. Alle Schnittstellen werden initialisiert und die Kommunikation beginnt. Beim Ende der Initialisierung soll über das Web Interface angezeigt werden, dass das System bereit ist und die Fahrt gestartet werden kann.

### **Würfelaufnahme**

Nach dem Startsignal über das Web Interface fährt der Zug in langsamer Fahrt vorwärts bis der Würfel von den Sensoren erfasst wird. Sobald der Würfel erkannt wurde hält der Zug an und der Schwenker mit dem Würfel wird eingefahren.

### **Hochgeschwindigkeitsfahrt**

Nachdem der Würfel aufgeladen ist kann der Zug beschleunigen. Der Zug soll immer die Höchstmögliche Geschwindigkeit fahren. Die Geschwindigkeit wird für die Kurven gemäss der Bilderkennung angepasst. Auch sucht das System dauernd das Infosignal und erkennt die Nummer darauf, sobald die Nummer erkannt wurde gibt das System diese Information über den Lautsprecher aus. Sobald das Signal der Ende der Zeitmessung erkannt wird verlangsamt der Zug und beginnt die Parkplatzsuche

### **Parkplatzsuche**

Um das korrekte Haltesignal zu finden kann der Zug eine so tiefe Geschwindigkeit fahren wie nötig. Die Bilderkennung sucht das korrekte Signal und entscheidet wo der Zug parkiert werden soll.

### **Parkieren**

Nachdem das korrekte Signal erkannt ist, wird der Parksensor ausgeklappt. Dieser misst die Distanz zum Haltesignal bis ein bestimmter Schwellwert erreicht ist. Beim Erreichen der Haltedistanz Stoppt der Zug und beendet damit die Fahrt.

## Zustand

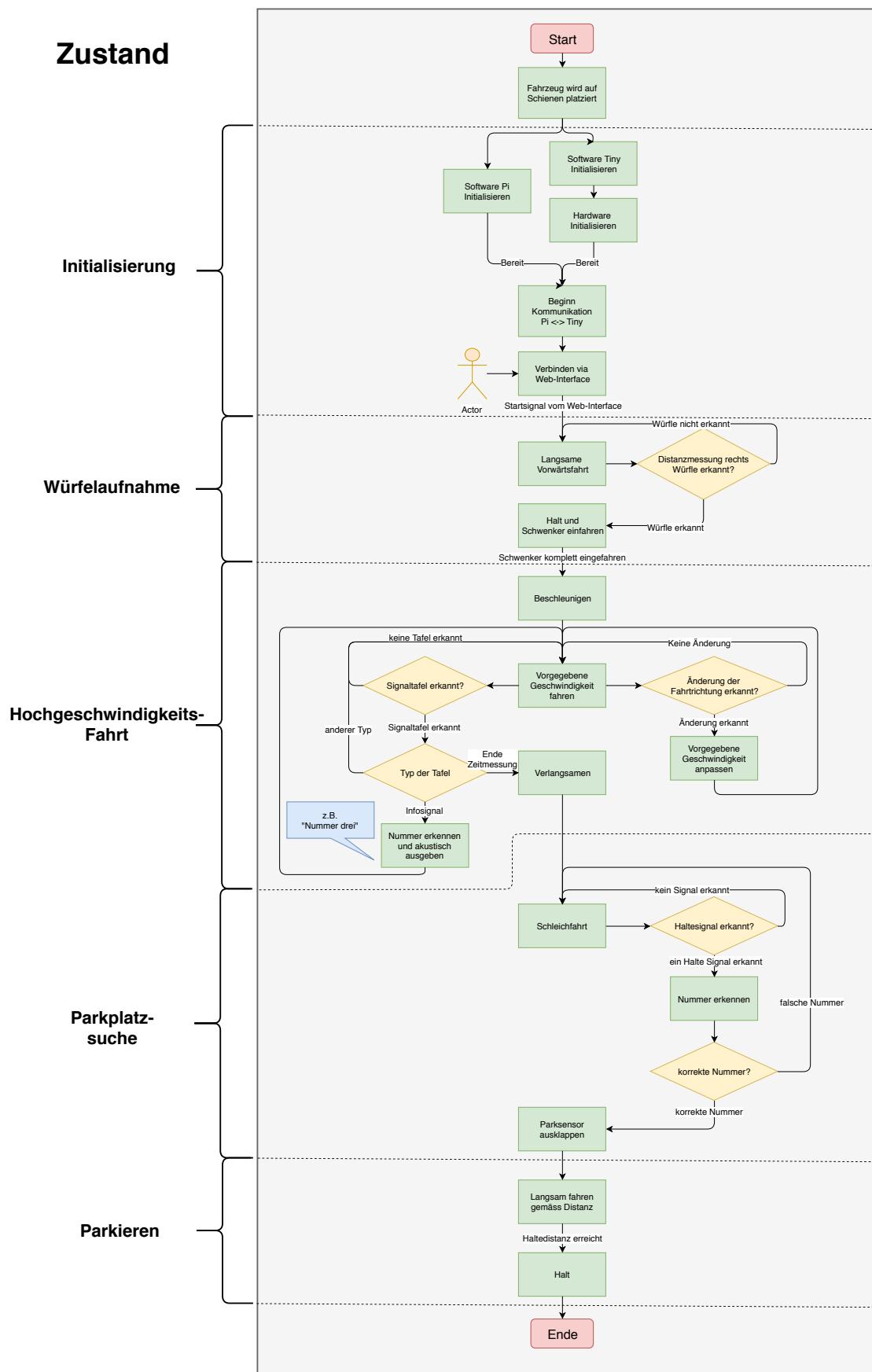


Abbildung 1: Ablaufdiagramm

### 3.3 Konzeptbeschreibung

Die Lokomotive in Abbildung ?? (siehe Tabelle ??) ist in die drei Unterbaugruppen Antriebswagen (Position 1), Führungswagen (Position 2) und Ladungsträger (Position 3) unterteilt. Der Antriebswagen enthält alle notwendigen Komponenten, um die Lokomotive zu beschleunigen und wieder abzubremsen. Zusätzlich sind die Kameras für die Spur- und Signalerkennung an ihm angebracht. Der Führungswagen hingegen dient lediglich als Abstützung für den Ladungsträger. Er bietet aber zusätzlichen Bauraum für elektronische Komponenten.

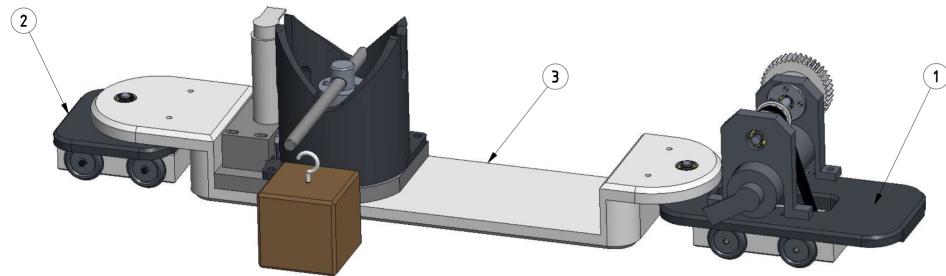


Abbildung 2: Baugruppe Lokomotive

Position	Bezeichnung
Position 1	Antriebswagen
Position 2	Führungswagen
Position 3	Ladungsträger
Position 4	Transportgut (Würfel)

Tabelle 1: Positionsnummern der Lokomotive mit Bezeichnungen

In der Querschnittsdarstellung in Abbildung ?? ist die Lagerungen zwischen dem Ladungsträger und dem Antriebswagen beziehungsweise dem Führungswagen besser sichtbar. Ebenfalls sieht man gut, wie der Antriebswagen mit Zahnriemen angetrieben wird. Der Aufbau der Unterbaugruppen wird in den nächsten Abschnitten vorgestellt.

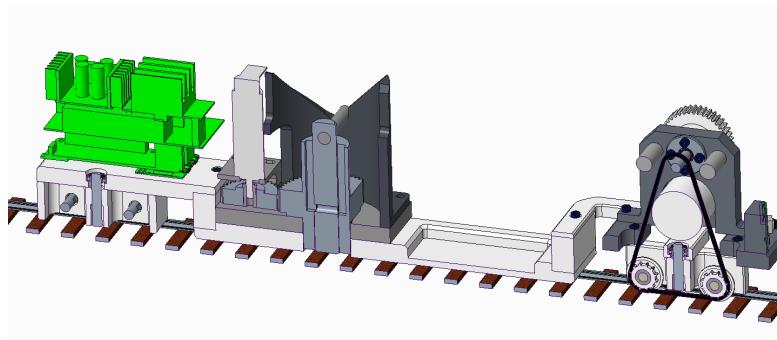


Abbildung 3: Schnittansicht der Lokomotivenbaugruppe

### 3.3.1 Antriebswagen

Der Grundaufbau der beiden Wägen ist derselbe, mit dem Unterschied, dass der Antriebswagen durch einen Motor angetrieben wird. Der Grundwagen beziehungsweise der Führungswagen in Abbildung ?? (siehe Tabelle ??) besteht aus einem Rahmen und einer Platte, welche miteinander verstiftet (Position 2) und verschraubt (Position 3) sind. Im Rahmen werden die beiden Achsen jeweils mit einem Los- (Position 7) und einem Festlager (Position 8) lagert und mit Sicherungsringen (Position 6) gesichert. Die Achsen sind an beiden Enden mit einem Gewinde versehen, damit die Räder bei Bedarf schnell und einfach gewechselt werden können, ohne dass der ganze Wagen auseinander genommen werden muss. Die Anfräsfäche auf der Welle (Position 5) dient für das bessere Befestigen der Räder mit einem Gabelschlüssel.

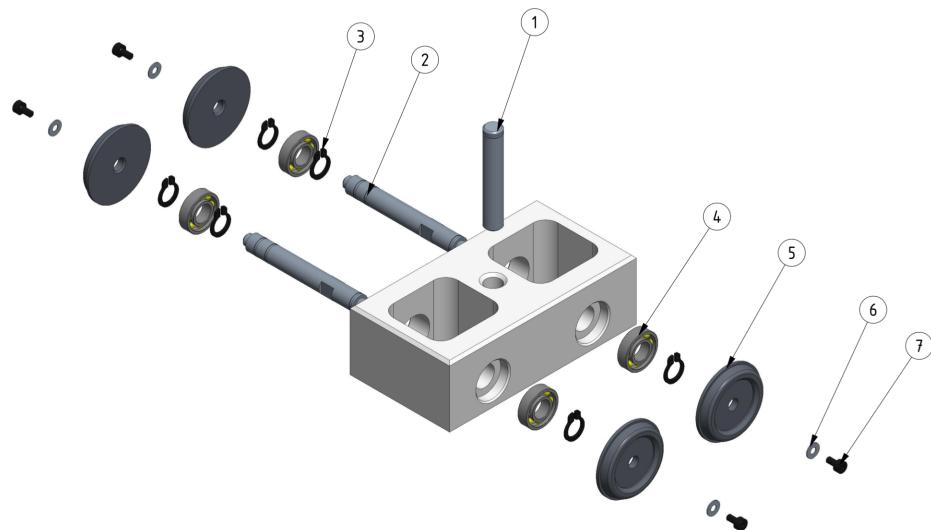


Abbildung 4: Explosionsdarstellung Führungswagen

Position	Bezeichnung
Position 1	Drehachse Wagen-Ladungsträger
Position 2	Achsen mit Gewinden an beiden Enden und Anfräsfäche für Gabelschlüssel
Position 3	Sicherungsring für Rillenkugellager
Position 4	Eingepresstes Rillenkugellager (Festlager) bzw. Loslager
Position 5	Rad
Position 6	Unterlagscheibe
Position 7	Zylinderschraube

Tabelle 2: Explosionsdarstellung Führungswagen

Der Antriebswagen in Abbildung ?? (siehe Tabelle ??) wie bereits erwähnt grundsätzlich gleich aufgebaut wie der Führungswagen, jedoch ist der Rahmen aufgrund des Riemenantriebs H-Förmig aufgebaut. Das Ziel dieser Konstruktion ist es hauptsächlich den Riemen, falls nötig, schnell wechselbar zu montieren. Durch die H-Form können die Achsen und die Räder für die Riemenmontage am Rahmen montiert gelassen werden. Die Platte, welche am Rahmen angebracht ist, ist für die Kameras für die Signal- und Spurerkennung grösser dimensioniert. Die Kamera für die Spurerkennung wird einstellbar befestigt, damit bei der Testphase des Prototyps Optimierungen des Winkels vorgenommen werden können. Der Stromübergang von der Schiene auf die Lokomotive erfolgt über vier Schleifkontakte, welche als Einkaufsteile von Lieferanten bezogen werden. Davon sind pro Wagen zwei und jeweils zwischen den Rädern angebracht.

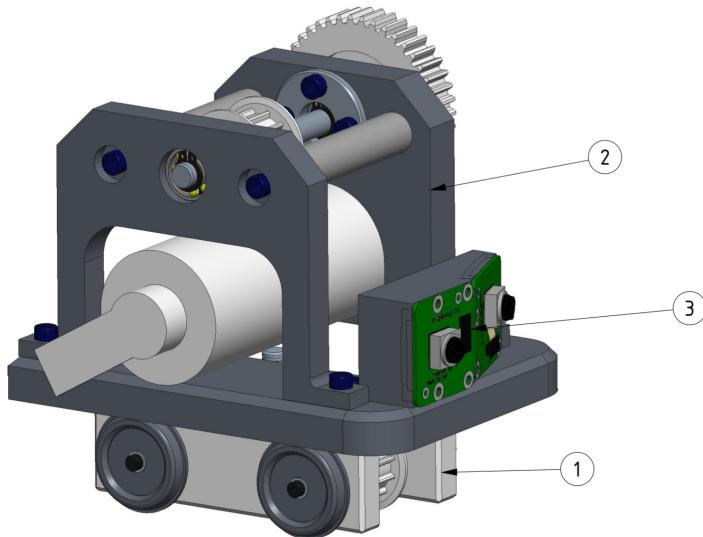


Abbildung 5: Baugruppe Antriebswagen

<b>Position</b>	<b>Bezeichnung</b>
Position 1	Wagen
Position 2	Antriebseinheit
Position 3	Kameras

Tabelle 3: Explosionsdarstellung Antriebswagen

Die Antriebseinheit in Abbildung ?? (siehe Tabelle ??) besteht aus einem Grundgestell, an welchem die Lagerung der Antriebswelle und der Motor angebracht sind. Das Drehmoment vom Motor wird über ein geradverzahntes Zahnrad mit einem Übersetzungsverhältnis von 1:2 auf eine Achse übertragen. Von dieser wird es über einen Riemen auf die beiden Radachsen und somit auf die Räder weitergeleitet. Durch die Berechnung der maximalen Geschwindigkeit wird entsprechend der Motor ausgelegt, was im nächsten Abschnitt genauer beschrieben wird.

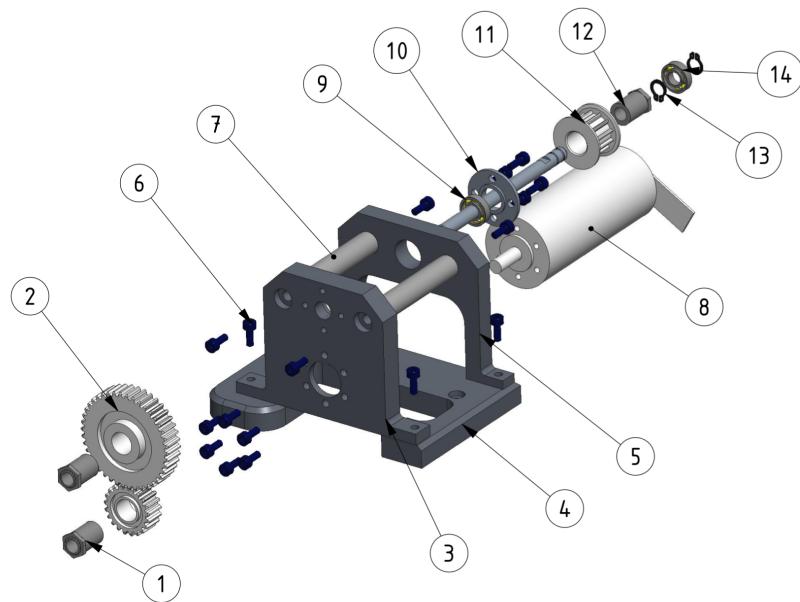


Abbildung 6: Explosionsdarstellung Antriebseinheit

Position	Bezeichnung
Position 1	Spannsatz
Position 2	Zahnräder Übersetzung 1:2
Position 3	Motorhalterung
Position 4	Grundplatte
Position 5	Achsenlagerplatte
Position 6	Zylinderschraube
Position 7	Distanzhülsen
Position 8	Motor
Position 9	Rillenkugellager (Loslager)
Position 10	Lagersicherungsplatte
Position 11	Zahnriemenrad
Position 12	Spannsatz
Position 13	Sicherungsring
Position 14	Rillenkugellager (Festlager)

Tabelle 4: Positionen Antriebseinheit

### Beschleunigungsberechnung

Wie schnell die Lokomotive beschleunigen kann hängt von der Reibung zwischen Rad und Schiene und der Masse des Zuges ab. Die Grunddefinition der Beschleunigung ist der Quotient von Kraft und Masse. Die Reibung ist durch den Reibkoeffizienten bestimmt, welcher sich von Materialpaarung zu Materialpaarung unterscheidet. Zusätzlich ist der Reibkoeffizient von der Oberflächenbeschaffenheit, der Rauigkeit, abhängig. Je die Rauigkeit, desto mehr Reibung entsteht und desto schneller kann beschleunigt werden. Die Kraft, um das Fahrzeug aus dem Stillstand zu beschleunigen ist grösser, als wenn es bei einer gewissen Geschwindigkeit weiter beschleunigt wird. Dies ist auf Grund des Trägheitsmomentes des Zuges.

Material Schiene	Material Rad	Reibungskoeffizient
Stahl	Stahl	0.05
Stahl	Holz	0.12
Stahl	Kunststoff	0.08
Stahl	Gummi	0.3

Tabelle 5: Reibungskoeffizienten verschiedener Materialpaarungen

Um die maximale Beschleunigung zu berechnen, wird das Gesamtgewicht der Lokomotive auf die vier Räder aufgeteilt. In der Tabelle ?? sind die gegebenen Größen für die nachfolgenden Berechnungen aufgelistet.

Grösse	Wert
Durchmesser Rad [D]	30 Millimeter
Reibungskoeffizient [k]	0.3

Tabelle 6: Größen für die Beschleunigungsberechnung

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

$$F_{\text{Reibung}} = F_{\text{Rad}} * k = 0.375N * 0.3 = 29.4N = 0.1125N$$

$$M_{\text{Rad}} = F_{\text{Rad}} * 2 * D_{\text{Rad}} = \frac{m * g = 3kg * 9.81m/s^2}{8} = 0.375Nm$$

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

### Geschwindigkeitsberechnung

Die Fahrgeschwindigkeit der Lokomotive wird durch zwei Faktoren bestimmt. Einerseits muss die maximale Geschwindigkeit der Anforderungsliste eingehalten werden und andererseits wird die Geschwindigkeit in der Kurvenfahrt durch den Schwerpunkt des Fahrzeugs eingeschränkt. Nachdem die Lokomotive auf die Fahrtgeschwindigkeit beschleunigt wurde, ist die Rollreibung das einzige, was der Motor mit seiner Leistung kompensieren muss.

In der Anforderungsliste wurde eine minimale Geschwindigkeit von 0.5 Meter pro Sekunde festgelegt. Der begrenzende Faktor der Geschwindigkeit in der Kurve ist der Schwerpunkt der Lokomotive. Je tiefer dieser ist, umso schneller kann die Kurve abgefahren werden. Über die Zentripedalkraft und die Gewichtskraft der Lokomotive wird die Momentengleichung aufgestellt und Anhand der gegebenen Werten in Tabelle ?? wird die maximal erreichbare Geschwindigkeit in der Kurve, ohne aus den Gleisen zu kippen, berechnet.

Das Kippmoment wird durch den Aufbau der Lokomotive minimiert. Da der Schwerpunkt durch die beiden Wagen mehr in das Zentrum des Kreismittelpunktes rückt.

Grösse	Wert
Minimaler Radius [r]	0.8 Meter
Masse [m]	3 Kilogramm
Schwerpunkt in x-Achse (maximaler Wert) [x]	0.025 Meter
Schwerpunkt in y-Achse (maximaler Wert) [y]	0.08 Meter

Tabelle 7: Größen für die Geschwindigkeitsberechnung

Die Gewichts- und Zentripedalkraft, welche das Gleichungssystem bilden, sind wie folgt definiert:

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

$$F_{\text{max } z} = \frac{m*v^2}{r} = \frac{3kg*(1.28m/s)^2}{0.8}$$

Da das Drehmoment eine vektorielle Größe ist, müssen die beiden entstehenden Momente am Drehpunkt "P" am Gleis zusammen Null ergeben. 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 Zentripedalkraft entsteht. Nun werden die ob

$$F_{\text{max } z} = \frac{F_G * x}{y} = \frac{29.4N * 0.025m + 9.2N * a}{0.08m} = 9.2N$$

### 3.3.2 Ladungsträger

Der Ladungsträger ist das Verbindungselement von Antriebswagen und Führungswagen. Er wird an beiden Enden drehbar mit je einem Radial- und Axialkugellager gelagert. Der Träger besteht aus drei Teilen (siehe Abbildung ??). Der Hauptgrund ist die einfache, materialsparende sowie kostengünstigere Herstellung. Andererseits muss der Träger

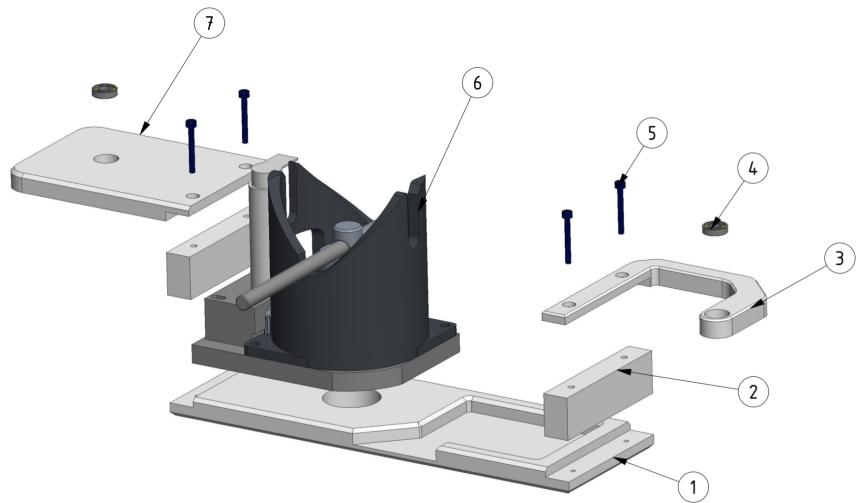


Abbildung 7: Explosionsdarstellung Ladungsträger

Position	Bezeichnung
Position 1	Grundplatte
Position 2	Zwischenplatte
Position 3	Bügelgelenk
Position 4	Rillenkugellager
Position 5	Zylinderschrauben
Position 6	Würfelkran
Position 7	Plattengelenk

Tabelle 8: Positionen des Ladungsträgers

### 3.4 Würfelaufnahme/Transport

Um den Würfel rechts neben der Gleisstrecke aufzunehmen wird eine einfache Lösung angestrebt, steuerungstechnisch sowie mechanisch. Aus dem morphologischen Kasten (Anhang) und der Nutzwertanalyse geht hervor, dass die Würfelaufnahme mit einem Draht und einem Stab durchgeführt wird. 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.

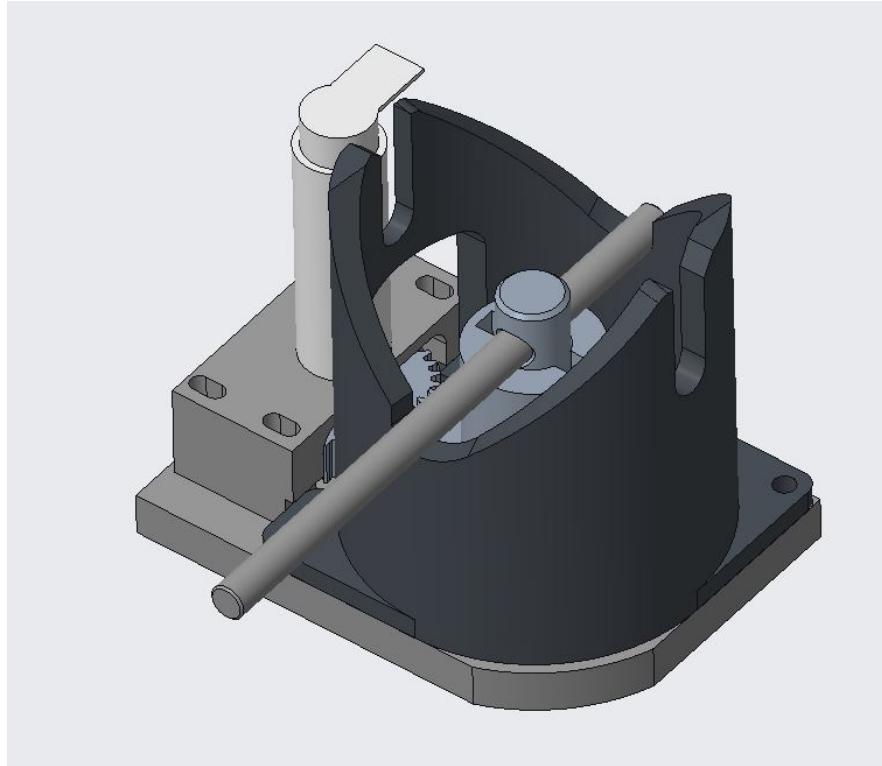


Abbildung 8: Baugruppe

#### 3.4.1 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 längsnutten in einem 3D gedruckten, modifizierten Zahnrad gelagert.

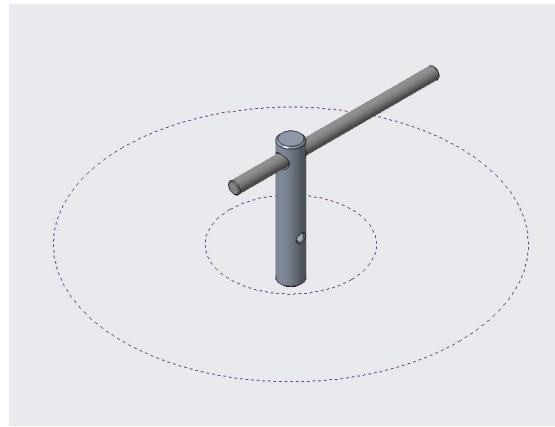


Abbildung 9: Draufsicht

### 3.4.2 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 Mädlar eingekauft. Das zweite Zahnrad jedoch wurde nur als STEP von Mädlar 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 als Drehmomentübertragung.

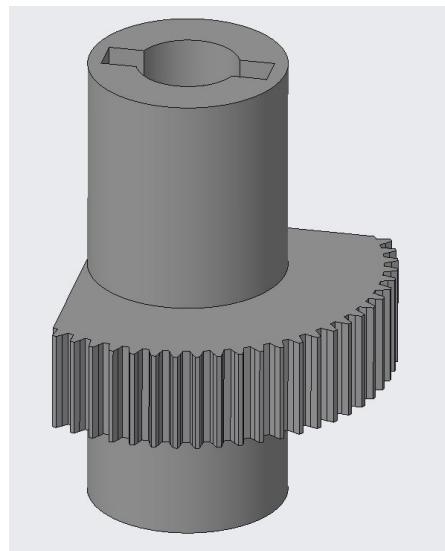


Abbildung 10: Zahnrad

### 3.4.3 Kurvenscheibe

Die Kurvenscheibe ist ebenfalls ein 3D-Druckteil. Der Grundkörper ist ein Rohr mit dem Aussendurchmesser 80 mm. An diesem wurden zwei Bahnführungen mittels Frei-

formflächen 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.

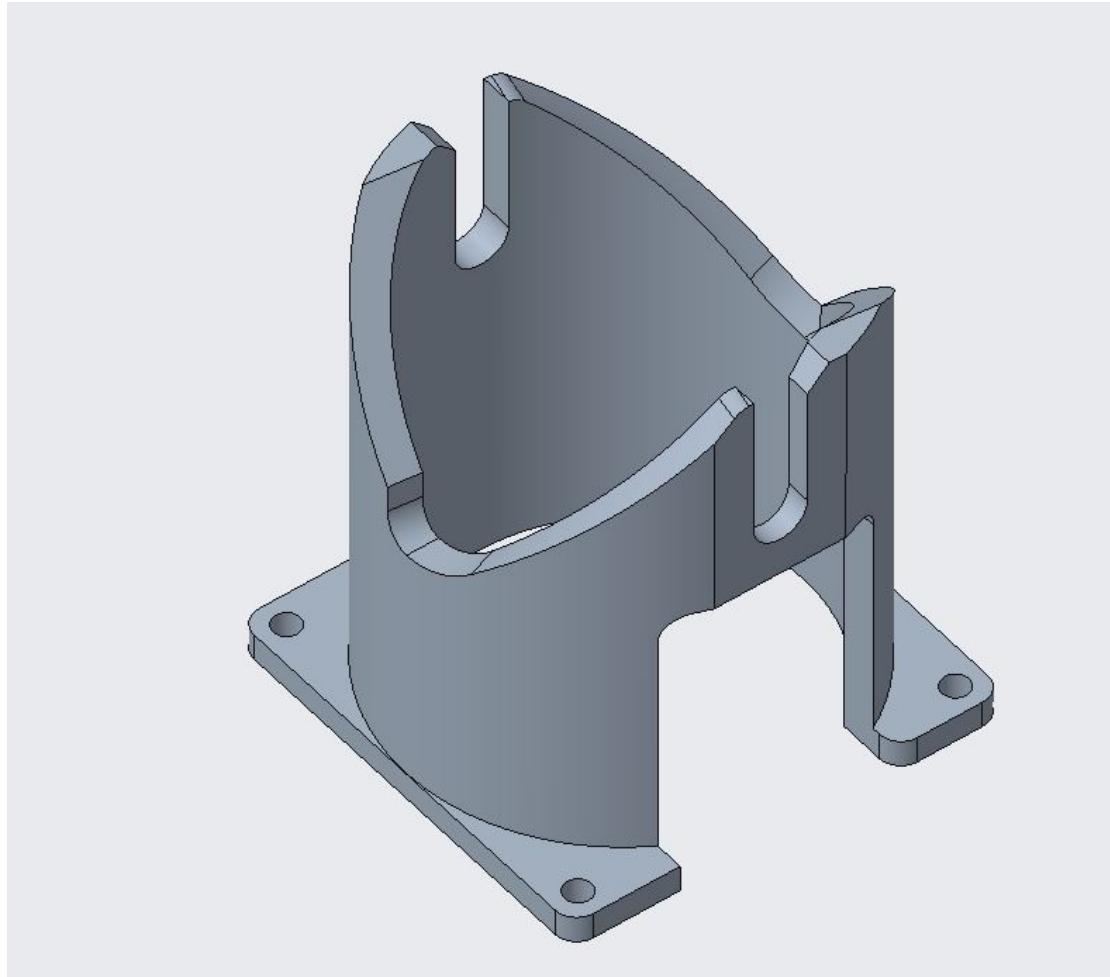


Abbildung 11: Kurvenscheibe

#### 3.4.4 Testaufbau

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

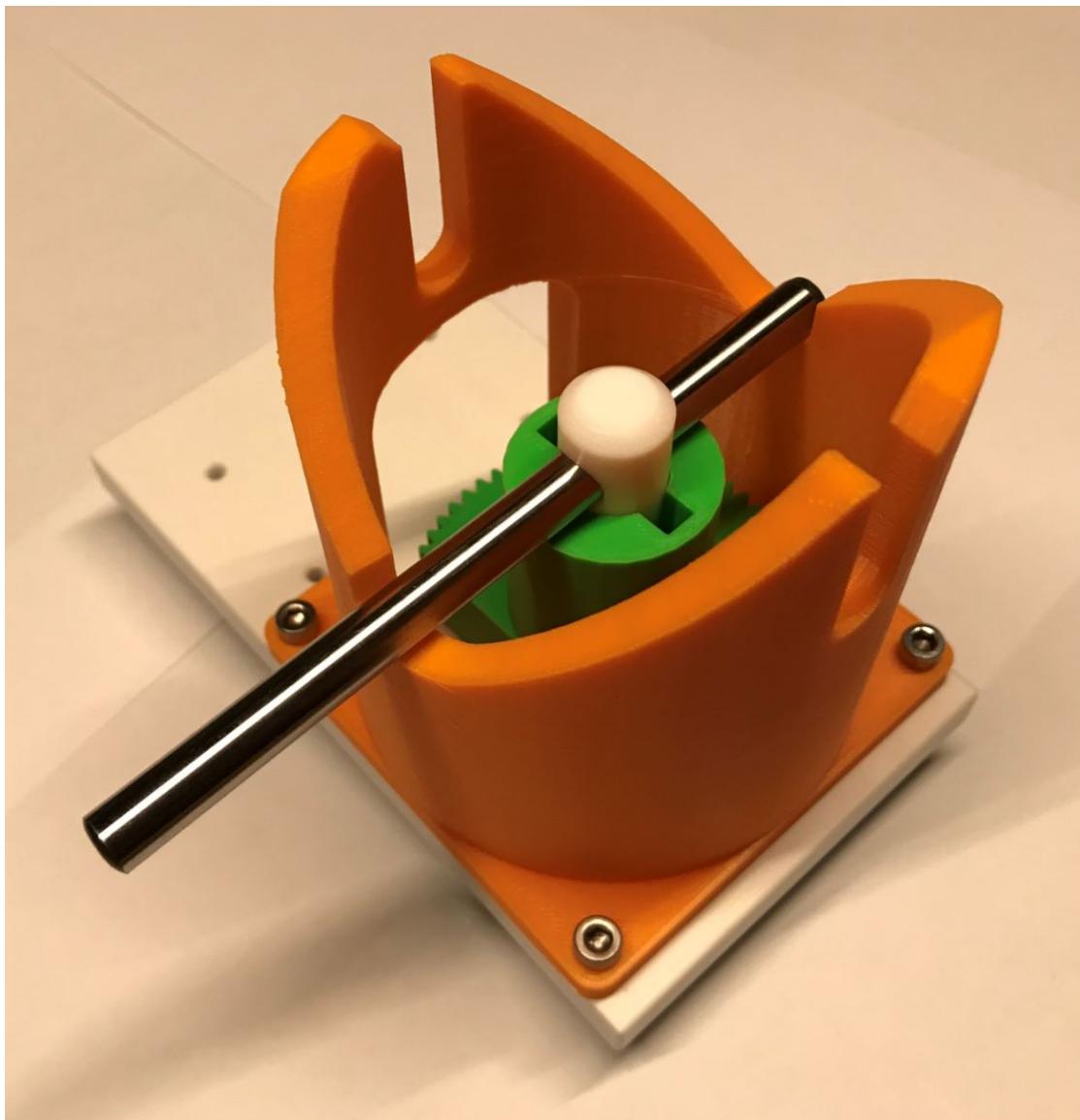


Abbildung 12: Testaufbau

### 3.5 Motorauslegung

Um die Aufgabenstellung der schnellen Fahrt auf Schienen best möglichst zu erfüllen, wird ein zuverlässiger starker Motor benötigt. Um einen solchen aus einer Vielzahl von Auswahlmöglichkeiten zu definieren hat man sich auf den Katalog des 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 wird 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 entstehen Trägheitskräfte, welche durch ein Moment vom Motor überwunden werden müssen. 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 Leistung zur Verfügung. Diese ergibt sich aus der Spannung 20 Volt und dem Strom von 3 Ampère. 60 Watt stehen also theoretisch zur Verfügung. Für eine preiswerte Lösung in Form eines DC Motors kommen bei maxon motor ag nur die zwei Produktreihen DCX und RE in Frage. Maxon motor ag bietet ein Sponsoring an in Form von Motoren mit kleinen Makeln, die nicht mehr ausgeliefert werden dürfen. Die Wunschmotoren der Gruppe 28 sind auf Grund der Leistung der DCX 32 oder der RE 30.



Abbildung 13: DC Motoren

### 3.6 Akustik

Während der Fahrt wird ein Signal mit einer Nummer gelesen, diese Nummer soll am Ende der Fahrt akustisch wiedergegeben werden. In diesem Kapitel wird das Lösungskonzept für unsere akustische Komponente aufgezeigt.

#### Anforderungen

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

**Konzept** Das Audiosignal wird über einen Buzzer wiedergegeben. Der Buzzer wird über den GPIO Bus angesprochen. Die 3.3V des Raspberry Pi reichen aus um den Buzzer zu versorgen.

**Komponente** Als Komponenten verwenden wir einen 3.3V Passiv/Aktiv Buzzer. Der Buzzer ist mit seinen 25mm x 25mm sehr kompakt und sollte ohne Probleme auf dem Zug Platz finden. Auch ist er für unter 5Fr zu erwerben und schont somit das Budget. Die Verbindung von Buzzer zu Raspberry Pi ist online gut dokumentiert und sollte keine unerwarteten Probleme mit sich bringen.

Name	Passiver Buzzer / Speaker, 3.3V
Preis	5Fr
Länge	25mm
Breite	25mm
Höhe	7mm
Gewicht	10g
Versorgungsspannung	3.3V

**Bauplan / Interface** Über den GPIO Bus des Raspberry Pi kann der Buzzer direkt angesprochen und versorgt werden. Ein einzelnes Signalkabel reicht für die Kommunikation aus und hält den Aufbau weitestgehend simpel.

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

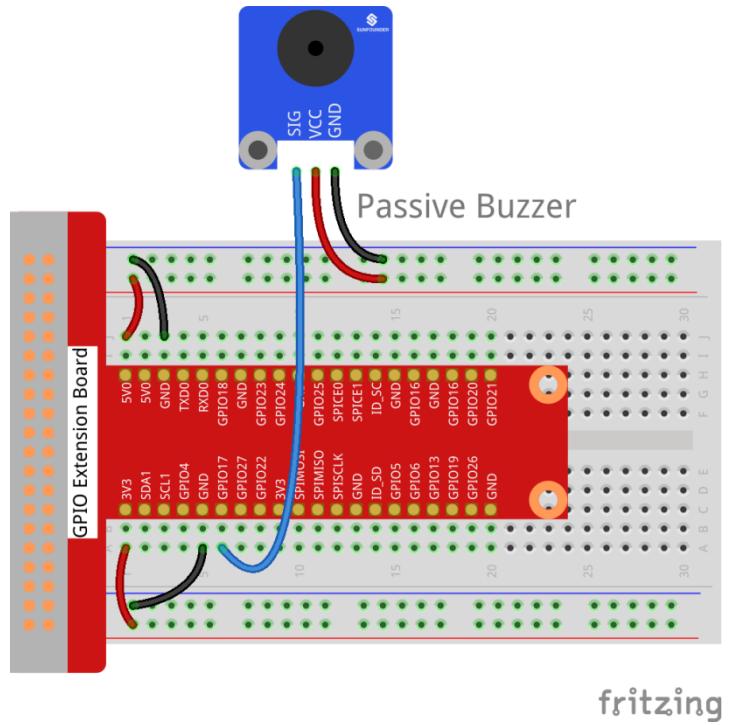


Abbildung 14: Verkabelung Buzzer

**Daten** Dem Buzzer werden verschiedene Frequenzen gesendet, welche dieser dementsprechend abspielt. Somit ist das Spielen einer Melodie möglich. Für unsere Benutzung reicht jedoch eine einzelne Frequenz. Die Frequenz wird im Intervall von 500ms an den Buzzer gesendet und ist somit in der Lage die höchste mögliche Zahl “9” innerhalb von 4.5s abzuspielen.

Die Frequenz wird in Form einer Zahl(von 100 bis 1000) gesendet, der Frequenzbereich kann je nach Buzzer variieren.

**Realisierung** Der Code wird in Python realisiert und macht Verwendung von den Bibliotheken GPIO und time. Es wird in einem Intervall (500ms) eine Frequenz auf den GPIO Port ausgegeben. Alle Module/Komponente werden asynchron ausgeführt, das Ausführen von time.sleep(ms) sollte somit kein Problem sein.

Das Buzzern wird aus einer selbstimplementierten “Sound” Bibliothek über die Funktion "buzz\_by\_number(number)" ausgeführt. Dabei wird über eine Schnittstelle das gewünschte Signal an den Buzzer gesendet.

### 3.7 Beschleunigung

Mithilfe elektronischer Komponente kann Beschleunigung, Geschwindigkeit und Distanz(vom Startbereich) gemessen und analysiert werden. Zusätzlich kann eine approximierte Position des Zuges auf der Strecke berechnet werden. Die Berechnungen dazu sind in Kapitel ?? beschrieben.

#### Anforderungen

- Momentane Beschleunigung auslesen
- Beschleunigung in Geschwindigkeit und Distanz umrechnen
- Approximierte Position berechnen (wo auf der Strecke befinden wir uns)

**Konzept** Über ein Beschleunigungssensor werden Beschleunigung und Rotation ausgelernt. Die Beschleunigung kann zusätzlich in Geschwindigkeit und Distanz umgerechnet werden.

**Komponente** Bei der Komponentenwahl fiel der Entscheid auf einen Adafruit  $I^2C$  3-Achsen Beschleunigungssensor, dieser kann Beschleunigung sowie Rotation berechnen. Er weist eine sehr kompakte Bauform auf und ist relativ günstig zu erwerben. Online haben verschiedene Nutzer mit dieser Komponente positive Erfahrung gesammelt. Auch ist die Komponente gut dokumentiert und man findet verschiedene Tutorien wie man diese mit einem Raspberry Pi kombinieren kann.

Name	Adafruit ADXL 345
Preis	1Fr
Länge	25mm
Breite	19mm
Höhe	3.14mm
Gewicht	1.27g
Versorgungsspannung	3-5V

**Bauplan / Interface** Der Sensor wird über die  $I^2C$  Schnittstelle angesprochen und verwendet somit einen Datenport (SDA) und eine Clock (SCL). Für die Stromversorgung reicht der 3.3V Port (3V3) und der Ground (GND) des GPIO Buses.

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

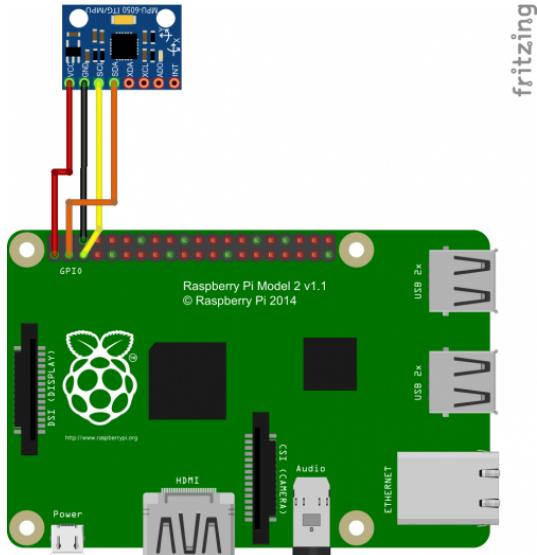


Abbildung 15: Verkabelung Beschleunigungssensor

**Daten** Mithilfe eines Skripts erhalten wir eine gute Übersicht der erhaltenen Daten.

#### Gyroskop

```
gyroskop\_xout: -260  skaliert: -2
gyroskop\_yout: -154  skaliert: -2
gyroskop\_zout: 78    skaliert: 0
```

#### Beschleunigungssensor

```
beschleunigung\_xout: -1048  skaliert: -0.06396484375
beschleunigung\_yout: -676   skaliert: -0.041259765625
beschleunigung\_zout: 16644  skaliert: 1.01586914062
X Rotation: -2.32121150537
Y Rotation: 3.59994842011
```

**Realisierung** Daten werden in einem von uns festgelegten Intervall über die  $i^2c$  Schnittstelle eingelesen. Die  $i^2c$  Schnittstelle muss in der Raspberry Pi Konfiguration aktiviert werden. Weiter müssen die benötigten Tools “i2c-tools” sowie “python-smbus” installiert werden. Dem Raspberry Pi wird eine Adresse auf dem  $i^2c$  Datenbus zugewiesen, welche über `sudo i2cdetect -y 1` abgerufen werden kann.

Softwaretechnisch wird die Schnittstelle in Python realisiert, denn Python verfügt über die mächtigsten Bibliotheken in den Bereichen Mathematik und  $i^2c$ . Der Beschleunigungssensor kann direkt über seine Adresse angesprochen werden und gibt die Daten in Form eines Words zurück. Der Beschleunigungssensor muss für jedes Word(z.B. `gyroскоп_xout`) eine Anfrage auf den Datenbus schreiben und lesen.

Die erhaltenen Daten werden in momentane Geschwindigkeitsdaten(Beschleunigung, Geschwindigkeit, Distanz) umgerechnet und anschliessend analysiert.

### 3.8 Elektronik Komponenten

In diesem Kapitel wird der Aufbau der Elektronik des Fahrzeugs beschrieben.

Die Abbildung ?? veranschaulicht den Aufbau der Elektronik. Darauf sind alle logischen Verbindungen eingezeichnet. Weitere Elektrische Verbindungen für die Stromversorgung werden im Kapitel ?? weiter erläutert.

Zentral dabei 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 ?? beschrieben.

Die Schnittstelle über den Debugger zum PC wird hier nicht weiter beschrieben, da diese Verbindung nur zum Entwickeln benutzt wird und für das Endgültige Produkt nicht mehr von Bedeutung ist sobald alles funktioniert.

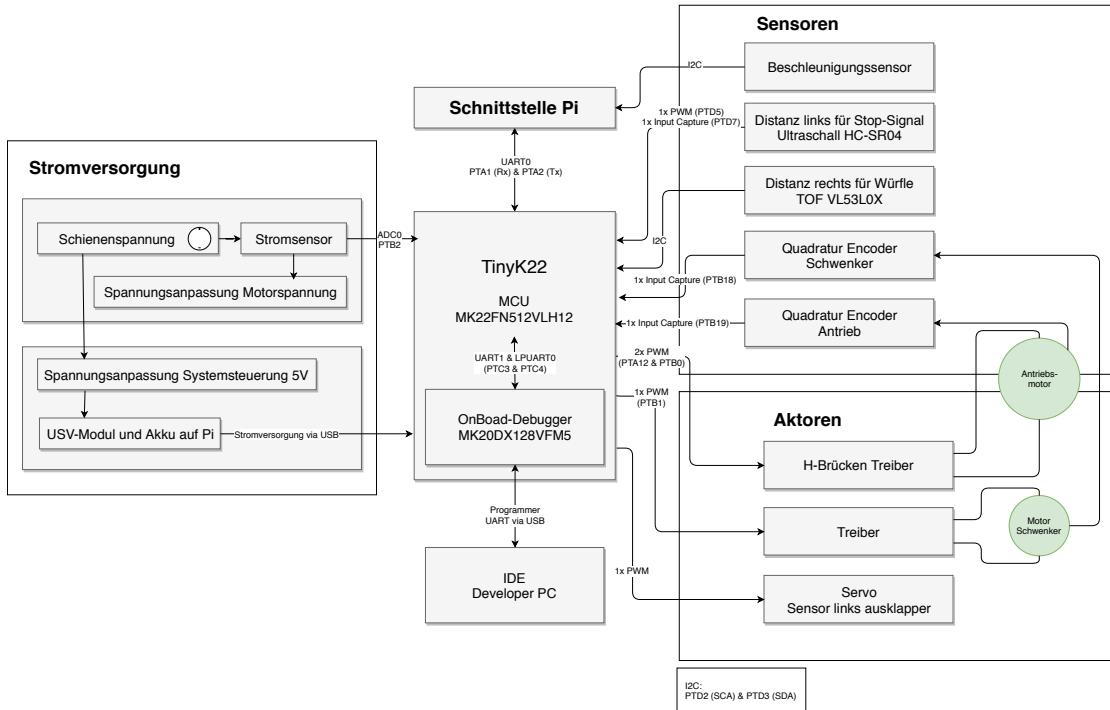


Abbildung 16: Komponentendiagramm Elektronik

#### 3.8.1 Stromversorgung

Die Energie für das System wird über die Schienen bezogen.

Die Antriebsenergie wird direkt von den Schienen bezogen und lediglich auf die korrekte Motorenspannung angepasst. Die Systemsteuerung wird über ein StromPi 3 mit Strom versorgt. Als Primärstromquelle wird dafür die Schienenspannung benutzt. Bei der Hochgeschwindigkeitsfahrt wird diese Quelle abgeschalten, damit die Gesamte Energie für den Antrieb zur Verfügung steht. Das StromPi schaltet sofort auf die sekundäre Stromversorgung für das Pi. Diese sekundäre Quelle wird durch einen LiFePO4-Akku auf dem StromPi realisiert. Sobald die Primärstromquelle von den Schienen wieder zur Verfügung steht wird der Akku wieder nachgeladen.

Das Pi zero und das TinyK22 mit diversen Sensoren und Aktoren werden über die USB-Anschlüsse des Pi versorgt.

## Antriebsenergie

Die Energie für den Antrieb wird direkt von den Schienen bezogen. Bei der Hochgeschwindigkeitsfahrt werden alle anderen Verbraucher von den Schienen deaktiviert um die gesamte Energie für den Antrieb nutzen zu können.

## Verpolungsschutz

Auf den Schienen steht eine Gleichspannung zur Verfügung, wobei eine Schienenseite der + Pol und die andere Seite der – Pol ist. Daraus ergibt sich das Problem, dass beim Platzieren des Zuges entgegen der vorgesehenen Fahrtrichtung eine Verpolung stattfindet. Auch ist die Zuordnung der Pole in der Aufgabenstellung noch nicht spezifiziert. Um sicherzustellen, dass das System unabhängig von der Polung der Schienen funktioniert muss eine Gleichrichtung realisiert werden. Dies kann mit einem Brückengleichrichter realisiert werden. (siehe Abbildung ??)

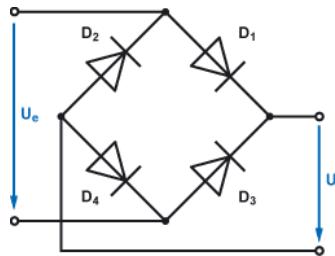


Abbildung 17: Brückengleichrichter

Quelle: <https://www.elektronik-kompendium.de/sites/slt/1807181.htm>

Die Ausgangsspannung  $U_a$  hat dabei immer dieselbe Polarität, unabhängig welche Polarität  $U_e$  hat.

Für den Hochgeschwindigkeitszug wird ein "B40C 3700-2200" verwendet. Dabei handelt es sich um eine Integrierte Schaltung für einen Brückengleichrichter. Der maximal zulässige Strom ist dabei 3.7A. Bei diesem Strom fällt über dem Gleichrichtung eine Spannung von ca 1V ab.

## Spannungsanpassung

Gemäß der Aufgabenstellung stehen  $20 \pm 2V$  bei bis zu 3A zur Verfügung. Aufgrund der grossen Toleranz von 4V muss die Spannung angepasst werden um eine stabile Spannung zu sicherzustellen. Dafür wird ein DC-DC Converter verwendet. Zu beachten dabei ist, dass der Converter für den maximalen Strom von 3A ausgelegt ist.

## Unterbrechungssicherheit

Um sicherzustellen, dass ein allfälliger Wackelkontakt der Schleifkontakte überbrückt werden kann und um grössere Spannungsschwankungen zu vermeiden wird ein Kondensator als Stützung eingesetzt.

## Stromüberwachung

Um den Stromverbrauch des Antriebs zu überwachen wie ein Strommesswiderstand eingesetzt. Eine Differenzverstärkerschaltung bereitet die Spannung über dem Strommesswiderstand auf, damit der Mikrocontrollers mit dem Analog-Digital Wandler den Strom bestimmen kann. Mit dieser Information kann die Software des Mikrocontrollers auf Stromspitzen reagieren und z.B. die Geschwindigkeit reduzieren. Der Stromverbrauch ist auch für Entwicklungs- und Testphase eine sehr wertvolle Information, damit das System optimal ausgelegt werden kann.

Zur Messung wird ein "13FR200E - Strommesswiderstand" verwendet. Dieser hat einen Widerstand von  $0.2\Omega$ . Der Strom kann daraus mit dem Ohmschen Gesetz bestimmt werden.

$$I = \frac{U_R}{R} = \frac{U_R}{0.2\Omega}$$

### Akku

Während der Hochgeschwindigkeitsfahrt wird die Systemsteuerung über einen Akku mit Strom versorgt. Die benötigte Leistung der einzelnen Komponenten ist in Tabelle ?? aufgeführt. Im schlimmsten Fall soll der Akku das System dabei mindestens für zwei Durchläufe mit Energie versorgen können.

Eine Runde darf maximal vier Minuten dauern, das bedeutet die Energie vom Akku muss für mindestens 8 Minuten reichen. Daraus ergibt sich für die Energie

$$E = P \cdot t = 14.2W \cdot 8min = 113.6Wmin = 1.9Wh$$

Komponente	Leistung (max)
Raspberry Pi 3 Model B	12.50W
Raspberry Pi zero W	1.20W
Tiny K22	0.10W
Encoder	0.07W
H-Brückentreiber	0.02W
Treiber Schwenker Motor	0.5mW
Beschleunigungssensor	0.3mW
TOF-Sensor	20mW
Ultraschall Sensor	75mW
<b>Total</b>	<b>14.20W</b>

Tabelle 9: benötigte Leistung der Komponenten

Das StormPi Modul wird mit einem LiFePO4-Akku ergänzt. Der Standardakku hat  $1000mAh$ . Bei einer Spannung von  $3.7V$  entspricht dies  $3.7Wh$ .

Dies ist genügend für zwei Durchläufe und da der Akku außerhalb der Hochgeschwindigkeitsfahrt nachgeladen werden soll, reicht dieser Akku gut.

### 3.8.2 Sensoren

Mit diversen Sensoren sollen folgende Daten aufgenommen werden. Folgende Daten sollen aufgezeichnet werden:

- Beschleunigung
- Geschwindigkeit
- Position
- Distanz rechts (für Erkennung Würfel)
- Distanz links (für Erkennung Haltesignal)

#### Beschleunigung

Die Beschleunigung wird mit einem Beschleunigungssensor ADXL345 aufgenommen.

Dieser kommuniziert über eine  $I^2C$  Schnittstelle mit dem Mikrocontrollers. Er liefert jeweils die Beschleunigung in x-, y-, und z-Richtung. Der Sensor wird parallel zur Fahrtrichtung montiert, sodass es reicht eine Beschleunigungsrichtung für die Querbeschleunigung und eine Richtung für die Längsbeschleunigung auszuwerten. Die Verwendung des Sensors ist in Kapitel ?? detailliert beschreiben.

### Geschwindigkeit

Die Geschwindigkeit wird hauptsächlich über den Quadratur Encoder am Antriebsmotor aufgenommen. Zusätzlich wird die Geschwindigkeit über die Beschleunigung zur Plausibilitätskontrolle nachgerechnet.

**Quadratur Encoder:** Der Quadratur Encoder MR, Typ L gibt 1024 Impulse pro Umdrehung. Der Verlauf eines Impulses ist in Abbildung ?? dargestellt. Der Encoder stellt drei Kanäle zur Verfügung. Über den Kanal 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 eines Impulses gemessen werden oder es können die Anzahl Impulse in einer bestimmten Zeit gezählt werden. Für dieses Projekt soll die Dauer des Impulses gemessen werden. Der Vorteil dieser Methode ist die bessere Präzision, da nach jedem einzelnen Impuls die durchschnittliche Geschwindigkeit seit dem letzten Impuls sofort bestimmt werden kann. Das Risiko ist, dass bei hohen Umdrehungszahlen der Mikrocontroller nicht schnell genug ist mit dem Zählen oder dass der Zähler des Mikrocontrollers bei sehr tiefen Umdrehungszahlen überläuft. Die maximalen Umdrehungszahl des Antriebsmotors für die angestrebte Maximalgeschwindigkeit liegt bei  $5200\text{min}^{-1}$ . Mit 1024 Impulsen pro Umdrehung ergibt dies

$$5200\text{min}^{-1} \cdot 1024\text{Impulse} = 5'324'800 \frac{\text{Impulse}}{\text{min}}$$

. Dies sind dann

$$5'324'800 \frac{\text{Impulse}}{\text{min}} / 60\text{s} = 88'747 \frac{\text{Impulse}}{\text{s}}$$

. Dies ergibt eine Impulsdauer von

$$\frac{1}{36'651 \frac{\text{Impulse}}{\text{s}}} = 11.29\mu\text{s}$$

. Bei der schnellsten möglichen Timer Einstellung auf dem TinyK22 ( $60\text{MHz}$ ) ergibt dies noch  $676 \frac{\text{Ticks}}{\text{Impuls}}$ . Bei der Implementation muss also beachtet werden, dass bei hoher Geschwindigkeit möglichst wenig Zeit in der Interrupt routine verbracht wird, da diese dann sehr oft aufgerufen wird. Sollte sich zeigen, dass der Mikrocontroller durch die hohe Impulsrate überlastet ist, muss ein Encoder mit weniger Impulsen pro Umdrehung gewählt werden oder ein Frequenzteiler dazwischen geschaltet werden.

Damit der Timer nicht überläuft muss eine Impulspause fertig sein, bevor der Timer den Wert

$$2^{32} \approx 4.29\text{Mrd}$$

erreicht wird. Dies ergibt bei  $60\text{MHz}$  eine Zeit von

$$2^{32} \cdot \frac{1}{60\text{MHz}} = 71.58\text{s}$$

. Also ist ein tiefe Umdrehungszahl bis zu

$$\frac{1}{71.58s \cdot 1024} = 13.6 \cdot 10^{-6} s^{-1}$$

Also sollte diese Umdrehungszahl nicht unterschritten werden. Diese Zahl ist so klein, dass dies als Stillstand gewertet werden kann.

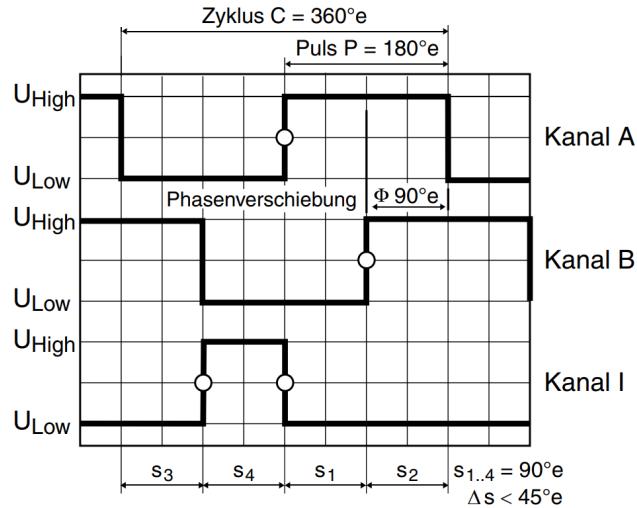


Abbildung 18: Signalverlauf Encoder

Quelle: Katalogseite 420, Encoder MR Typ L, maxon sensor

Die Geschwindigkeit des Zuges ( $v_{Zug}$ ) kann somit aus der Umdrehungszahl des Motors ( $N_{Motor}$ ) mittels der Mechanischen Übersetzung und der Grösse der Räder berechnet werden.

$$v_{Zug}(N_{Motor}) = \frac{N_{Motor}}{2} \cdot d \cdot \pi$$

$d$  : Raddurchmesser 22mm

**Nachrechnen der Geschwindigkeit:** Unter der Annahme, dass zu beginn der Messung zum Zeitpunkt  $t = 0$  die Geschwindigkeit 0 ist ( $v(t = 0) = 0$ ) kann die Geschwindigkeit zum Zeitpunkt  $t$  bestimmt werden mit

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

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

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

## Position

Über den Beschleunigungssensor kann man die aktuelle Position auf der Fahrbahn berechnen. Die zurückgelegte Strecke errechnet sich durch Integration der Geschwindigkeit

oder durch zweifache Integration der Beschleunigung.

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

Auf beiden Seiten des Zuges wird eine Distanzmessung benötigt. Auf der rechten Seite des Zuges muss der Würfel erkannt werden und auf der Linken Seite soll ein ausklappbarer Sensor am Schluss die genaue Distanz zum Haltesignal bestimmen.

**Würfelerkennung:** 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.  $20\text{mm}$  und  $80\text{mm}$  erkennen können. Der exakte Wert der Distanz ist dabei nicht entscheidend, da nur ein bestimmter Schwellwert erkannt werden muss. Die Distanz zum Würfle wird mit einem TOF (Timo-Of-Flight) Sensor VL53L0X ermittelt.

**Haltesignalerkennung:** Um möglichst präzise anhalten zu können muss die Systemsteuerung den exakten Wert der Distanz zum Haltesignal kennen. Dies wird mit einem Distanzsensor ermittelt. Diese Distanz wird mit einem Ultraschall Sensor HC-SR04 gemessen. Es ist entscheidend, dass der Sensor korrekt und mit wenig Toleranz auf der Mechanik befestigt wird um eine exakte Ausrichtung auf das Haltesignal sicherzustellen.

### 3.8.3 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 Ansteuer 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. Dabei zu beachten ist, dass der Anlaufstrom bis zu 70A betragen

kann. Dieser Strom muss begrenzt werden indem der Mikrocontrollers die Beschleunigung gemäss dem gemessenen Stromverbrauch anpasst. ? 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  $5200 \text{min}^{-1}$  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 mindestens 14.8V versorgt werden.

### **Motorentreiber für Schwenker-Motor**

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

### **Schwenkermotor**

Für den Schwenker wir ein Maxon DCX 19 S verwendet. Um die Position des Schwenkers zu bestimmen ist auch an diesem Motor ein Encoder befestigt. Durch die Bestimmung der nötigen Umdrehungen bis der Schwenker eingefahren ist kann genau festgestellt werden wie viele Impulse abgewartet werden müssen bis der Der Motor anhalten muss. ?

### **Distanzsensor links ausklappen**

Um den Sensor beim Parkieren auszuklappen zu können wird er an einem Servo befestigt. Sobald die Systemsteuerung entscheidend, dass das korrekte Haltesignal das nächste ist gibt sie den Befehlt den Sensor auszuklappen. Dafür wird ein "Tower Pro Micro Servo SG90" verwendet. Dieser hat ein Drehmoment von bis zu  $2.5 \text{kg} - \text{cm}$ . ?

#### **3.8.4 Proof-of-Concept**

Um die Funktionsfähigkeit der einzelnen Komponenten sicherzustellen wurden diverse Tests gemacht. Das ziel dieser Tests war es die einfachste Form jeder Funktionalität zu realisieren. Auf eine quantitativ genaue Anpassung wurde in diesem ersten Schritt verzichtet, dies ist für die Realisierung des Projekts vorgesehen. Das Ziel der Tests war somit nur zu Zeigen, dass die Komponente wie vorgesehen funktionieren kann. Eine Übersicht der durchgeführten Tests ist in Tabelle ?? ersichtlich.

Komponente	Beschreibung des Tests	Ok
Tiny K22	Der Mikrocontroller konnte erfolgreich programmiert werden und Hardware kann darüber angesteuert werden.	✓
UART Kommunikation	Mittels dem "Raspberry Pi APROG HAT" konnte erfolgreich zwischen dem Pi und dem Tiny mittels UART kommuniziert werden.	✓
Beschleunigungssensor	Der Beschleunigungssensor wurde über die Schnittstelle angesteuert und die Werte für die x-, y- und z-Richtung konnten erfolgreich ausgelesen werden.	✓
Motorenansteuerung	Vom Mikrocontroller wurde ein PWM generiert. Der Pin mit dem PWM wurde mit dem Motorentreiber Arduion IBT 2 und dieser dann mit einem Motor verbunden. Durch Vorgaben des Mikrocontrollers konnte die Geschwindigkeit und die Richtung des Motors eingestellt werden.	✓
Encoder	Der Encoder wurde vom Mikrocontroller ausgelesen und die Zeiten und Impulse ausgewertet. Es zeigte sich dass der Encoder korrekt 1024 Impulse pro Umdrehung liefert. Daraus lässt sich dann die Umdrehungszahl berechnen.	✓
Ultraschallsensor	Der Ultraschallsensor wurde vom Mikrocontroller gemäss dem Datenblatt angesteuert. Mittels der gemessenen Zeit konnte eine approximative Distanz berechnet werden. Für einen exakten Wert sind noch genauere Abstimmungen nötig. Der Sensor reagiert jedoch zuverlässig auf Objekte in der Grösse des Würfels oder der Haltesignale.	✓
TOF-Sensor	Mit einer Code-Bibliothek, die für das Tiny K22 angepasst wurde, konnte der TOF-Sensor erfolgreich angesteuert werden. Damit kann eine approximative Distanz berechnet werden. Für einen exakten Wert sind noch genauere Abstimmungen nötig. Der Sensor reagiert jedoch zuverlässig auf Objekte in der Grösse des Würfels oder der Haltesignale.	✓
Spannungsanpassung	Der DC-DC Converter wurde mit einer Spannungsquelle verbunden und die Ausgangsspannung bei diversen Einstellungen korrekt gemessen.	✓
Stromüberwachung	Eine Differenzverstärkerschaltung wurde mit unterschiedlichen Lastwiderständen simuliert und durch die Messspannung den Strom bestimmt und mit dem Strom der Simulation verglichen.	✓

Tabelle 10: Übersicht Proof-of-Concept Elektronik

### 3.9 Signalerkennung

In der Aufgabenstellung wird gefordert, dass der Schnellzug während der Bewältigung der Strecke ein Signal mit aufgedruckter Nummer erkennt wird. Die Nummer ist auf einer 3x3cm Tafel mit weissem Hintergrund, schwarz aufgedruckt. Die Aufgabe Signalerkennung wird in zwei Teilaufgaben unterteilt:

- Erkennung der Signalkennung mit Tafel
- Erkennung der aufgedruckten Nummer

Wie bereits in der Übersicht beschrieben, wird im Gesamtkonzept zwei Kameras verwendet. Eine Kamera wird zur Erkennung der Schienenrichtung verwendet. Die zweite Kamera wird nun für die Signalerkennung eingesetzt. Der Raspberry PI 3+, welcher als Hauptrecheneinheit geplant ist, verfügt nur über einen CSI- Anschluss (Camera Serial Interface). Für die Signalerkennung wird auf einen weiteren kleineren Raspberry PI 3 Model A+ gesetzt. Dieser verfügt wie der Raspberry PI 3+ einen vollwertigen CSI-Anschluss und kann somit die zweite Kamera bedienen. Dies führt auch dazu, dass der Raspberry PI 3+ zusätzlich entlastet werden kann.

#### Architektur

Die Signalerkennung in zwei Teilaufgaben zu trennen hat noch einen weiteren vorteil. Die beiden Teilaufgaben werden auf beide Raspberry PIs verteilt. Der Raspberry PI 3 Model A+ ist für die Bilderaufnahme Verarbeitung und für die Signalerkennung mit Tafel zuständig. Der Raspberry PI 3+ übernimmt dann die Erkennung der aufgedruckten Nummer auf der Tafel. So kann die Ressourcen des Raspberry PI 3+ gezielter eingesetzt werden.

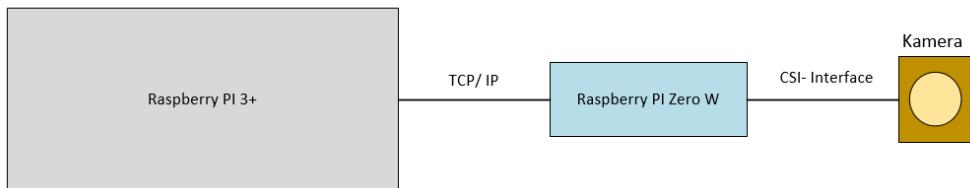


Abbildung 19: Architektur Hardware Signalerkennung

Für die Nummererkennung auf der Tafel Machine- Learning Algorithmus verwendet. Damit die Trainingsdaten des Models eingelesen werden können und somit ausführbar wird, braucht es eine gewisse größe des Arbeitsspeichers. Der Arbeitsspeicher des Raspberry PI 3 Model A+ (512Mb) reicht dafür nicht aus. Der Raspberry PI 3+ verfügt über mehr speicher (1Gb) und kann somit den Algorithmus bearbeiten. Für die Kameraaufnahmen und die Bearbeitung der Bilder und schlussendlich die Konturenerkennung der Signalkennung ist der Raspberry PI 3 Model A+ gut geeignet.

## Software Tafelerkennung

Für die Erkennung der Tafel wird auf das Framework OpenCV gesetzt. OpenCV hat sich als Standartframework im Bereich der Echtzeitbildverarbeitung. Weiter zeichnet sich OpenCV in seiner Effizienz und seiner breiten Community, welche im Internet mit vielen Tutorial zu hilfe stehen. In der Abbildung ?? ist der Ablauf der Tafelerkennung und in der Tabelle ?? die jeweilige Funktion beschrieben.

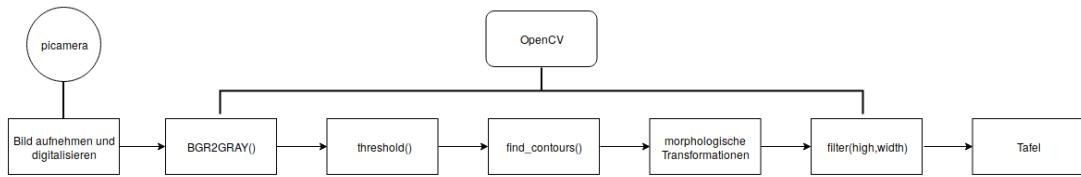


Abbildung 20: Ablauf Erkennung der Tafelerkennung

Funktion	Beschreibung
Bild aufnehmen und digitalisieren	Die Umgebung wird mittels der Raspberry PI Kamera und der Bibliothek picamera aufgenommen und dem OpenCV übergeben
BGR2GRAY	Die Farbinformation vom Bild werden nicht benötigt und deshalb entfernt (Effizienz)
threshold	Für die Konturenerkennung braucht es nur die Schattierungen des Bildes. Aus diesem Grund wird ein Threshold auf das Bild gelegt damit ein Binery- Picture generiert werden kann.
find-contours	Nun wird die OpenCV Funktion find-contours angewendet. Dabei werden benachbarte Schwarz oder Weiss Pixel miteinander zusammengefügt bis eine Kontur entstehen kann.
morphologische Transformationen	Hier wird das Bild mittels morphologische Transformationen nachbearbeitet
filter	Die erkannten Konturen werden gefiltert nach der Grösse und Form der Tafel.
Tafel	Nun ist das Bild mit der Tafel erkannt worden und wird zur Nummererkennung vorbereitet.

Tabelle 11: Beschreibung der OpenCV Funktionen

In der Abbildung ?? links sieht man die erkannte Position der Nummer auf der Tafel. Rechts sieht man die gleiche Aufnahme nach der Filterung. Dieses Bild wird nun auf die Grösse des rechten Erkannten Rechteckes zugeschnitten und dem Raspberry PI 3+ zur Nummererkennung übergeben.

## Konfiguration Raspberry PI Kamera

Eine weitere Schwierigkeit bei der Nummererkennung ist die Aufnahme mittels Kamera. Bei hohen Geschwindigkeiten neigt das Bild dazu unscharf zu werden. Dies liegt daran, dass die Verschlusszeit der Kamera zu lange ist. Mit der Raspberry PI Kamera und der Schnittstelle piCamera können viele Parameter eingestellt werden. Hier liegt der Schlüssel zur Behebung von unscharfen Bildern. Wenn Parameter wie Belichtungszeit, Helligkeit,

Sättigung und Weissabgleich fest eingestellt werden, können in erster Linie reproduzierbare Bilder aufgenommen werden und die Belichtungszeit kann auf ein Minimum gesetzt werden. Wenn die Belichtungszeit kürzer gesetzt wird, ist das Ergebnis aber dunklere Bilder. Mit digitaler Aufhellung oder sogar mit externer Beleuchtung kann aber diesen Effekt entgegengesetzt werden.



Abbildung 21: Nummerposition- und Nummererkennung

### Software Nummererkennung

Damit nun die Nummer auf der Tafel erkannt werden kann, wird ein Machine-Learning Algorithmus verwendet. Dafür wird die high-level API Keras verwendet. Als Backend kommt Tensorflow zum Einsatz. Als Trainingsdatabase kommt die frei Verfügbare MNIST- Database zu Einsatz. Die MNIST- Datenbank verfügt über 60'000 Zahlen in Handschrift. In Tabelle XX ist die Zusammenfassung der Tensorflowbackend bezüglich dem ausgewählten Keras Model. In den ersten Tests hat die Erkennung gut funktioniert auch auf dem Raspberry PI. Probleme sind in erster Linie mit der Nummer "1" aufgetaucht. Die MNIST Datenbank verfügt nur über die amerikanisch geschriebene "1". Die "1" in Arial wird dadurch nicht erkannt. Sobald in PREN2 die Schriftart der Zahlen bekannt gegeben wird, muss eventuell einen eigenen Trainingsdatensatz erstellt werden oder auf einen anderen Datensatz wie zum Beispiel von UCI Machine Learning Repository.

Layer(type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_1 (MaxPooling2)	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 128)	1179776
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

Tabelle 12: Auflistung aller Testklassen mit jeweiliger Testarten

## 3.10 Gleiserkennung

Ein Beschleunigungssensor für die Erfassung der Fahrdaten ist bereits in der Aufgabenstellung vorgeschrrieben. Um aber eine ideale Regelung zu ermöglichen wird zusätzlich eine Gleiserkennung implementiert. Mit welcher schon im vorraus erkennt werden kann ob eine Kurve bevorsteht oder ob sich der Hochgeschwindigkeitszug auf einer geraden befindet. Während der Konzept Phase wurden verschiedene Verfahren evaluiert und zum Teil ausprobiert. Jedoch gab es keine Lösung welche zusätzlich zur Richtung noch den Radius der Kurve bestimmen kann. Die Momentane Lösung welche angestrebt wird kann also nur sagen ob es Geradeaus, nach Links oder nach Rechts geht. Diese Information wird in den Regelungskreis eingespielen, welcher dadurch auf die maximale Geschwindigkeit die im engsten schienen Radius möglich ist herunterregelt. Anschliessend wird mithilfe des Beschleunigungssensor versucht an das Limit der Zentripedalkraft zu regeln. Somit kann die bestmögliche Regelung implementiert werden ohne genauere Angaben zum aktuellen Kurvenradius zu haben. Mehr Details zum Regelkreis und dem Ganzen Ablauf sind im unter ?? zu finden.

### 3.10.1 Verfahren

Nachfolgend werden die einzelnen Schritte zur bearbeitung eines Einzelnen Bild aufgelistet. Die einzelnen Teilschritte werden für jedes Bild welches aufgenommen wird neu berechnet.

- Bild Skalieren
- Umrechnen zu Schwarz Weiss
- Kantenglettung Anwenden (Canny Algorithmus)
- Konturen Erkennung (Suzukui85 Algorithmus)
- Kleine Konturen erkennen und verwerfen
- Bild in zwei Hälften unterteilen
- Berechnen der Anzahl Konturen in beiden Hälften
  - Beide Hälften etwa gleich viele Konturen -> Gleis geht geradeaus
  - Linke Hälfte hat mehr Konturen -> Gleis geht nach Links
  - Rechte Hälfte hat mehr Konturen -> Gleis geht nach Rechts

#### Bild Skalieren

Das Bild muss für die Berechnungen herunterskaliert werden um die Rechenzeit zu reduzieren. Das Format wird dabei beibehalten, um verzerrungen zu vermeiden. Zuerst wird also das Seitenverhältnis berechnet um dann die kürzere Seite auf einen Konfigurierbaren wert zu setzen. Die längere Seite wird dann mit dem Seitenverhältnis Faktor berechnet.

#### Umrechnen zu Schwarz Weiss

Der Canny Algorithmus für die Kantenglettung muss mit einem Schwarz-Weiss Bild gefüttert werden deshalb wird das farbige Bild zu einem Schwarz-Weiss Bild umgerechnet.

#### Kantenglettung Anwenden (Canny Algorithmus)

Der Canny Algorithmus sucht Kanten im Bild und erzeugt ein Binäres Bild (Schwarz oder Weiss), wobei die Weissen Pixel eine Kanten beschreiben.

### **Konturen Erkennung (Suzuki85 Algorithmus)**

Das Binäre Bild welches mithilfe vom Canny Algorithmus erzeugt wird kann mit dem Suzuki85 Algorithmus analysiert werden um Konturen zu erkennen. Die Konturen sind jeweils eine Liste von Punkten welche zusammen eine Kontur abbilden.

### **Kleine Konturen erkennen und verwerfen**

Die Anzahl Punkte einer Kontur werden mit einem Schwellwert verglichen. Sind die Anzahl Punkte innerhalb der Kontur zu gering wird sie verworfen. Somit können kleine Störungen im Bild bzw. kurze Linien ignoriert werden.

### **Bild in zwei Hälften unterteilen**

Das Bild wird nun in zwei Hälften geteilt um die Entscheidung zu treffen ob Geradeaus, nach Links oder nach Rechts gefahren wird.

### **Berechnen der Anzahl Konturen in beiden Hälften**

Um die Entscheidung zu treffen wird einfach die Anzahl Punkte in jeder Hälfte verglichen. Hat es in der Linken Hälfte mehr geht das Gleis nach Links. Sind es in der Rechten Hälfte mehr geht es nach Rechts. Ist die Anzahl ungefähr identisch, dann geht es Geradeaus. Da es nie genau viele Punkte haben wird gibt es eine minimal Differenz die es geben muss zwischen Rechts und Links damit eine Kurve detektiert wird.

### 3.11 Aufgabentrennung zwischen Pi und Tiny

Dieses Kapitel beschreibt die Aufgabentrennung zwischen der Systemsteuerung auf dem Raspberry Pi (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 Kamera aufnahmen
  - Entscheidung über bevorstehende Kurven
  - Entscheidungen gemäss erkannter Schilder
- Entscheidung der Fahrtgeschwindigkeit
- Auswertung des Beschleunigungssensor
- 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
  - Objekterkennung Haltesignal
  - Stromverbrauch
  - aktuelle Ist-Geschwindigkeit

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

### 3.12 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.

Diese Kommunikation soll es den Systemen ermöglichen die zugewiesenen aufgaben gemäss Kapitel ?? zu erfüllen.

### 3.12.1 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-duplex Kommunikation.

### 3.12.2 Übertragungsprotokoll

Zu jedem Zeitpunkt wird Datenpaket (im folgenden Frame genannt) in einem fest vorgelegten Format ausgetauscht. Dabei gibt es ein Grundformat für die Informationen und zwei Formate für den Informationsinhalt eines für Frames von Pi zum Tiny und ein anderes Format für die Frames vom Tiny zum Pi. Der Informationsinhalt der beiden Formate unterscheidet sich gemäss der Aufgabentrennung in Kapitel ??.

#### 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 Informationen zur Geschwindigkeit folgendermassen aussehen:

$speed, 200$

Die gesamten Informationen werden als Zeichenkette (String) im Ascii Format verschickt. Die Zahlen werden aus dem String jeweils umgewandelt. Dies verbessert die Leserlichkeit für Menschen, was zum Testen und Simulieren der Kommunikation sehr hilfreich sein kann.

#### Frames: Pi $\Rightarrow$ Tiny

Das Pi schickt dem Tiny eine vorgegebene Soll-Geschwindigkeit und Richtung der nächsten Kurve. Auch Informationen über den aktuellen Status des Pi werden verschickt. Die genauen Bezeichner und die Bedeutung und Grösse der Zugehörigen Werte sind in Tabelle ?? aufgelistet.

Bezeichner	Wert Beschreibung	Wert Zahlengrösse
speed	Soll-Geschwindigkeit	signed 32-Bit Integer
dir	Kurvenrichtung	signed 32-Bit Integer
status	Moral und weitere Informationen des Pi	unsigned 8-Bit Integer

Tabelle 13: Kommunikation Frames: Pi  $\Rightarrow$  Tiny

#### Frames: Tiny $\Rightarrow$ Pi

Das Tiny nimmt die nötigen Sensordaten auf und schickt die nötigen Informationen daran dem Pi. Auch Informationen über den aktuellen Status des Tiny werden verschickt. Die genauen Bezeichner und die Bedeutung und Grösse der Zugehörigen Werte sind in Tabelle ?? aufgelistet.

<b>Bezeichner</b>	<b>Wert Beschreibung</b>	<b>Wert Zahlengrösse</b>
is-speed	Ist-Geschwindigkeit (gemäss Encoder)	signed 32-Bit Integer
obj	Objekt erkannt (einzelne Bits 1 oder 0)	unsigned 8-Bit Integer
status	Moral und weitere Informationen es Tiny	unsigned 8-Bit Integer

Tabelle 14: Kommunikation Frames: Tiny  $\Rightarrow$  Pi

Der Wert der Moral ist bestimmt für Statusinformationen, welche den einzelnen Bits in der Zahl zugeordnet werden. Der genaue Inhalt ist kann sich je nach Status (gemäss Kapitel ??) unterscheiden. Der nötige Informationsinhalt um den Ablauf zu erfüllen ist ein Tabelle ?? aufgelistet. Für Tests soll es möglic sein den Inhalt je nach Bedarf zu erweitern.

<b>Informationen Pi <math>\Rightarrow</math> Tiny</b>	<b>Informationen Tiny <math>\Rightarrow</math> Pi</b>
Bereit	Bereit
Ablauf gestartet	Würfel erkannt
Befehl zum Schwenker einfahren	Schwenker vollständig eingefahren
Haltesignal erkannt (Parksensor ausklppen)	Haltestand erreicht
	Stromverbrauch zu hoch

Tabelle 15: Informationsinhalt des Statusbytes

### 3.13 Software Loesungskonzept

Die Steuerungssoftware welche auf dem Hauptrechner ausgeführt wird soll in mehrere Teilprogramme aufgeteilt werden. Einerseits können so die Teilprogramme unabhängig implementiert werden und andererseits einfacher getestet werden. Um anschliessend zwischen den Teilprogrammen zu kommunizieren wird eine Middleware verwendet. Die Middleware kann auch genutzt werden um die einzelnen Teilprogramme zu testen. Zusätzlich können verschiedene Programmiersprachen verwendet werden für die einzelnen Teilprogramme was zusätzliche Flexibilität

#### 3.13.1 Architektur

Im folgenden Diagramm wird die Architektur der Steuerungssoftware aufgezeigt

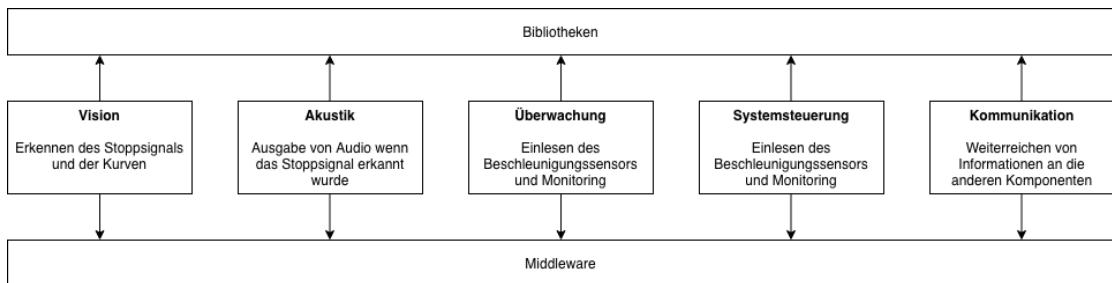


Abbildung 22: Software Architektur Middleware. Gezeichnet mit <https://draw.io>

#### Legende:

- Die Pfeile visualisieren die Abhängigkeiten innerhalb der Architektur

#### 3.13.2 Technologie

Während der Technologierecherche wurde eine Analyse gemacht um zu entscheiden welche Technologie eingesetzt werden soll. Bald wurde ZeroMQ ausgewählt da ZeroMQ schlank ist und ohne Performance Probleme auf kleinen Einplatinenrechner läuft. Außerdem ist die Middleware bekannt und hat eine hervorragende Dokumentation. Um die Daten welche über die Middleware geschickt werden mit verschiedenen Programmiersprachen zu nutzen wird Protobuffers. Protobuffers ist ein Format zur Beschreibung von Daten. Dieses Format kann anschliessend verwendet werden um Klassen bzw. Funktionen für fast jede Programmiersprache zu benutzen. Das heißt das die Daten einmalig definiert werden und anschliessend von allen Teilprogrammen verwendet werden können.

#### Beispiel Protobuf:

```
syntax = "proto3";  
  
message Direction {  
    string direction = 1;  
}
```

Man sieht nun das eine 'Direction' Mitteilung definiert wird welche ein String Attribut hat welches direction heißt. Mit dieser Definition nun mithilfe von einem 'protobuf'

compiler' Code generiert werden welcher die definierte Message Serialisieren und Deserialisieren kann.

## 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 'protobuffer compiler' erzeugt wurde.

### 3.13.3 Proof-of-Concept

Um zu testen ob eine Kommunikation zwischen zwei Prozessen mithilfe der Middleware möglich ist wurde eine kleine Testsoftware entwickelt. Dabei wurde das 'direction.proto' File verwendet um vom einten Prozess eine Richtung (Direction) zu einem anderen Prozess zu senden.

Der Sender der Richtung wurde im folgenden File implementiert: src/raspi/vision/linedetection/fakelinedetector.py Der Empfänger der Richtung wurde im folgenden File implementiert: src/raspi/webapp/middlewareadapter.py wobei der Empfänger von vom Webapp Server gestartet wird: src/raspi/webapp/server.py

### 3.13.4 Performance

Die Performance wurde nicht selbst gemessen und verifiziert. Jedoch wurden bei unseren Tests keine Limitationen gefunden und deshalb gehen wir davon aus das die Performance für unseren Anwendungsfall ausreicht. Ausserdem werden nur wichtige Events zwischen den Prozessen ausgetauscht und somit ist die Datenrate eher zweitrangig. Die Latenz ist jedoch sehr zentral um einen möglichst agilen Regelkreis zu implementieren. Hier bewegt sich ZeroMQ zwischen 450us - 700us.

#### Performance Tests:

- <http://zeromq.org/area:results>
- <http://nikolaveber.blogspot.com/2011/04/if-you-are-planning-large-or-not-even.html>

## 4 Projektmanagement

### 4.1 Projektmanagement

Dieses Kapitel beschreibt das Projektmanagement des Team 28. Es wird auf die Organisation im Team, die Aufteilung der Arbeitspakete und Zeitplanung eingegangen.

#### 4.1.1 Organigramm

Die Abbildung ?? zeigt die Organisation im Team auf. Das Team ist in die einzelnen Disziplinen Maschinenbau, Elektrotechnik und Informatik aufgeteilt. Zusätzlich gibt es einen Projektleiter, dieser ist Verantwortlich für die Kommunikation mit den Fachdozenten und ist bei allfälligen Abwesenheiten von Teammitgliedern informiert.

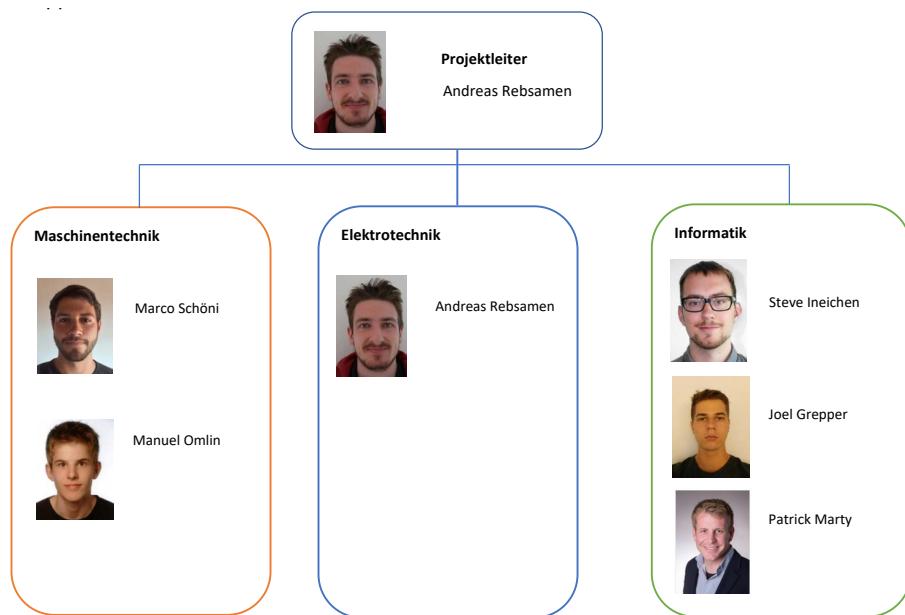


Abbildung 23: Organigramm Team 28

#### 4.1.2 Zeitplanung

Mit den groben Arbeitspaketen wurde eine zeitorientierte Projektstruktur (Abbildung ??) erstellt. In dieser ist festgehalten, welcher Person oder Disziplin ein Arbeitspaket zugeteilt ist, ob diese neu, in A-Breit oder abgeschlossen ist.

Abbildung 24: Zeitorientierte Projektstruktur Team 28  
Stand: SW 12

Die detailliertere Aufteilung dieser Arbeitspakete wird dann in Trello (Abbildung ??) vorgenommen. In der Projektstruktur wird lediglich eingetragen ob ein bestimmtes Arbeitspaket darin in Trello erfasst ist. Dort kann es in mehrere Teilpakete aufgeteilt werden und bestimmten Personen zugewiesen werden. Die Arbeitspakete in Trello werden je nach Status in die Kategorien "To do", "doing", oder "done". Separat werden auch noch die Meilensteine festgehaltne und als "done" gekennzeichnet sobald diese abgeschlossen sind.

The screenshot shows a digital whiteboard interface for the project PREN HS18. The top navigation bar includes tabs for 'Privat' (selected), 'AR', 'J', 'M', 'MS', and 'S.I.'. A 'Menü anzeigen' button is also present.

**TODO**

- Prototyp Akustik & Fahrdatenauswertung
- Schlussdokumentation: Zusammenfassung
- Schlussdokumentation: Einleitung / Zielseitung
- Schlussdokumentation: Lösungskonzept Produktbeschrieb, Funktion

**DOING**

- Schlussdokumentation: Lösungskonzepte Ablaufdiagramm / Blockdiagramm
- Schlussdokumentation: Lösungskonzept Komponenten
- Schlussdokumentation: Projektmanagement

**DONE**

- Zeitplan für V- Modell erstellen
- Blockdiagramm ET-Komponente
- CAD-Abklärungen mit maxon motor
- Risikomatrix erstellen

**Meilensteine**

- Testat 3 (Due: 14. Dez.)
- Ablieferung Dokumentation Gesamtkonzept (Due: 11. Jan. 2019)

At the bottom right, there is a button '+ Eine weitere Karte hinzufügen'.

Abbildung 25: Bildschirmausschnitt Trello Team 28  
Stand: SW 12

### 4.1.3 Dokumentation

Die Dokumentation wird in L<sup>A</sup>T<sub>E</sub>X geschrieben. Der Quell-Text wird in eine normale Text-Datei mit der Endung ".tex" geschrieben. Ein Programm übersetzt diese Quell-Datei dann in ein Dokument wie z.B. ein PDF oder PostScript. ?

Für die Dokumentation dieses Projekts wird aus dem L<sup>A</sup>T<sub>E</sub>X-Code ein PDF erstellt. Der Austausch der Daten im Team erfolgt gemäss Kapitel ??.

#### **4.1.4 Datenaustausch**

Um Daten im Team auszutauschen wird auf zwei verschiedene Cloud Plattformen gesetzt. Alle Teammitglieder haben vollständigen Zugriff auf beide Plattformen.

##### **One Drive**

Für diverse Ablagen, Word- und Exceldokumente oder Ähnliches steht ein Ordner in Microsoft One Drive zur Verfügung. Diese Daten werden jederzeit mit allen Teammitgliedern synchronisiert. In Dokumenten von Microsoft Office ist es auch möglich, dass mehrere Personen zur selben Zeit am selben Dokument arbeiten.

##### **Github**

Es steht ein Github Repository zur Verfügung. Mit dem Programm Git kann man auf die Daten zugreifen und seine Änderungen hochladen. Dabei können alle Änderungen von allen Personen verfolgt werden. Die genaue Verwendung und die benutzten Tools sind für die Teammitglieder direkt im Repository beschrieben.

Dieses Repository wird für die Dokumentation und für den Source-Code benutzt. Es gibt einen Ordner für die Dokumentation (alle L<sup>A</sup>T<sub>E</sub>XDateien, Bilder, Zeichnungen, usw.) und einen weiteren für den Quell-Code. Dort ist der Programmcode für das Raspberry Pi und das Tiny K22 abgelegt.

Das Repository kann online unter <https://github.com/Inux/pren> eingesehen werden.

## **5 Schlussdiskussion**

### **5.1 Schlussdiskussion**

- Entwicklungskosten, zeitlicher Entwicklungsaufwand - Erfahrungen, „Lessons learned“, kritische Würdigung der Arbeiten - offene Punkte, Risiken und Ausblick auf PREN 2