

Katzenfütterungsanlage

**HTBLA Kaindorf an der Sulm
Grazer Straße 202, A-8430 Kaindorf an der Sulm
Ausbildungsschwerpunkt Mechatronik und
Automatisierungstechnik**

Florian Greistorfer, Florian Harrer, Dominik Pichler, Julian Wolf

Abgabedatum: 05.04.2018

Betreut von:

Dipl.-Ing. Manfred Steiner
Dipl.-Ing. Dr. Gerhard Pretterhofer
Otto Schuller BEd.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Arnfels, am 5. April 2018

Florian Greistorfer

Florian Harrer

Dominik Pichler

Julian Wolf

Danksagung

An dieser Stelle möchten wir uns bei allen bedanken, die uns im Rahmen der Diplomarbeit unterstützt und betreut haben. Bedanken möchten wir uns bei unseren Betreuern, Dipl.-Ing Manfred Steiner, Dipl.-Ing Dr. Gerhard Pretterhofer und Otto Schuller BEd. für die fachliche Unterstützung während dieser Arbeit und bei allen Verwandten und Freunden für Ihre linguistische sowie moralische Unterstützung.

Abstract

The aim of this project is to develop the hard- and software for a prototype of a cat-feeding-machine. The project had been split in four segments to achieve this. These segments are the following: mechanics, electronics and programming. The programming was split into two more segments, which are the programming of a webserver with webapplication and the programming of a sequence control with graphical user interface. The task of the mechanic part is to construct, calculate and simulate parts of the cat-feeding-machine. The task of the electronic part is to create a complete circuit to control two motors, to read two sensors with the help of a Raspberry. The task of the programming part is to create a control with a graphical user interface and to develop a suitable webapplication.

Zusammenfassung

Das Ziel dieser Arbeit ist es, die Hard- und Software für den Prototypen einer Katzenfütterungsanlage zu entwickeln. Um dieses Ziel zu erreichen wurde das Projekt in vier Teilbereiche unterteilt. Diese sind die Mechanik, Elektronik und Programmierung, wobei die Programmierung zusätzlich in zwei Teilbereiche unterteilt wurde, in die Programmierung eines Webservers mit Webapplikation und der Ablaufsteuerung mit Grafischer Benutzeroberfläche. Die Aufgabe des mechanischen Teils besteht aus der Konstruktion, Berechnung und Simulation von Teilen der Katzenfütterungsanlage. Die Aufgabe des elektronischen Teils lag in der Entwerfung eines kompletten Schaltplans zur Ansteuerung zweier Motoren, Abfragung zweier Sensoren, mithilfe eines Raspberries. Die Aufgabe der Programmierung lag darin, eine Ansteuerung mit Grafischer Benutzeroberfläche zu entwerfen und eine dazugehörige Webapplikation zu entwickeln.

Gender Erklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Arbeit die Sprachform des generischen Maskulinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

Über dieses Dokument

Diese Arbeit wurde in L^AT_EX verfasst. Diese Art der Dokumentation bietet gegenüber den normalen Textverarbeitungen gewisse Vorteile hinsichtlich der Formatierung und des Einbindens von Grafiken. Auch Formeln können sehr einfach und effizient angegeben werden. Die Rohfassung des Dokuments befindet sich auf Github.

Erklärung der Sprachen

Da viele Ausdrücke, Datenblätter und weitere technische Dokumente in Englisch verwendet werden, lässt sich eine teilweise Vermengung der Sprachen nicht vermeiden.

Projektteam

Florian Greistorfer

**Aufgabenbereich:**

Webserver

Webclient

Betreuer:

Dip.-Ing. Manfred Steiner

Florian Harrer

**Aufgabenbereich:**

Java-Programm

Betreuer:

Dip.-Ing. Manfred Steiner

Dominik Pichler



Aufgabenbereich:

Mechanik

Betreuer:

Dip.-Ing. Dr. Gerhard Pretterhofer

Julian Wolf



Aufgabenbereich:

Elektrotechnik

Mechanik

Betreuer:

Otto Schuller BEd.

Inhaltsverzeichnis

1 Webserver und Client	1
1.1 Begriffserklärungen	2
1.1.1 Server	2
1.1.2 Client	2
1.2 Anforderungen	2
1.2.1 Webserver	2
1.2.2 Client	2
1.3 Voruntersuchung	2
1.3.1 HTTP/HTTPS	2
1.3.1.1 HTTP Status Codes	3
1.3.2 JavaScript	3
1.3.3 TypeScript	4
1.3.4 HTML	5
1.3.5 CSS	5
1.3.6 DOM	5
1.3.7 Node.js	5
1.3.8 express	6
1.3.9 JSON	6
1.3.10 JSON Web Token	7
1.3.11 MongoDB	7
1.3.12 Angular 2/4	8
1.3.12.1 Modules	8
1.3.12.2 Libraries	9
1.3.12.3 Components	9
1.3.12.4 Templates	10
1.3.12.5 Data binding	10
1.3.12.6 Services	11
1.3.13 Angular CLI	11
1.3.14 Bootstrap	11
1.4 Umsetzung	12
1.4.1 Projektstruktur	12

1.4.2 Client	12
1.4.2.1 Übersicht	12
1.4.2.2 Design	14
1.4.2.3 Funktion	19
1.4.2.4 Internationalization	22
1.4.2.5 Kommunikation mit dem Server	22
1.4.3 Server	24
1.4.3.1 Serverpfade	24
1.4.3.2 Funktion	24
1.4.3.3 MongoDB	27
1.4.3.4 Kommunikation mit dem Java Programm	28
1.4.4 Verbinden mit WLAN im Java Programm	29
1.5 Raspberry PI	29
1.6 Verbesserungsmöglichkeiten und Zusammenfassung	29
1.6.0.1 Verbesserungsmöglichkeiten	29
1.6.0.2 Zusammenfassung	30
2 Java-Programm	31
2.1 Anforderungen	31
2.1.1 Programm	31
2.1.2 Design - Benutzerinterface	31
2.1.3 Externe Steuerung	31
2.2 Voruntersuchung	32
2.2.1 Wieso Java und nicht C?	32
2.2.2 Wieso das Raspberry Pi 3 Model B?	32
2.2.3 Auswahl eines Touchdisplays	33
2.2.4 Wieso pi4j?	33
2.2.5 Wieso Mongodb?	33
2.2.5.1 Vorteil gegenüber Daten in Datei speichern	34
2.2.5.2 Vergleich mit anderen Datenbanken	34
2.2.6 Kommunikation mit der Web-Applikation	34
2.3 Umsetzung	35
2.3.1 Mongodb	35
2.3.1.1 Allgemeines	35
2.3.1.2 Datenbankmanagementsystem	36
2.3.1.3 Singleton	36
2.3.1.4 Code Beispiele	36
2.3.2 pi4j	38
2.3.2.1 Allgemeines	38
2.3.2.2 Pin Numbering Sheme	39

2.3.2.3	Gewählte Pin Belegung	39
2.3.2.4	Singleton	41
2.3.2.5	Code Beispiele	41
2.3.3	Server-Client-Kommunikation	42
2.3.3.1	Server	42
2.3.3.2	Übertragungsprotokoll	42
2.3.3.3	Response-GET	43
2.3.3.4	Response-PUT	43
2.3.3.5	Verarbeiten des Requests	44
2.3.4	Errors und Warnungsverarbeitung	44
2.3.4.1	Allgemeines	44
2.3.4.2	Errors und Warnungen aktivieren/deaktivieren	44
2.3.4.3	JSON-Object	45
2.3.5	SwingWorker	45
2.3.5.1	EDT	47
2.3.5.2	TimeUnit	47
2.3.5.3	Methoden des SwingWorkers	47
3	Mechanik	49
3.1	Einleitung	49
3.2	Aufgabenstellung und Zielsetzung	49
3.3	Problematik	49
3.3.1	Problematik des automatisierten Aufschneidens	50
3.3.2	Problematik der Dichtheit bei Klemmen	50
3.3.3	Problematik bei Entleerung der Verpackung	50
3.3.4	Problematik des Geruch	50
3.3.5	Problematik der Reinigung	50
3.3.6	Problematik beim Einfrieren des Futters	51
3.4	Konzepte	51
3.4.1	Variante 1: Automatisiertes Aufschneiden	51
3.4.1.1	Übersicht der Prozessschritte	51
3.4.1.2	Füllen des Futtermagazins	52
3.4.1.3	Führen zur Schneidplatte	52
3.4.1.4	Schnitt	54
3.4.1.5	Pressen	55
3.4.1.6	Entsorgen	56
3.4.1.7	Füttern	57
3.4.2	Variante 2: Vor aufgeschnittene Packung	58
3.4.2.1	Förderband und Kettenglieder	58
3.4.2.2	Walze	59

3.4.2.3	Futterplatte	60
3.4.3	Variante 3: Gefrorenes Futter	60
3.5	Aufbauten und Tests	60
3.5.1	Fütterungsexperiment	61
3.5.1.1	Schlussfolgerung des Fütterungsexperiments	63
3.5.2	Schneideversuch 1.Art der 1.Variante	63
3.5.2.1	Schlussfolgerung des Schneideversuchs 1.Art der 1.Variante . .	64
3.5.3	Schneideversuch 2.Art der 1.Variante	65
3.5.3.1	Schlussfolgerung des Schneideversuchs 2.Art der 1.Variante . .	65
3.5.4	Dichtheitsexperiment der Hebelklemme	66
3.5.5	Zustand der Futterpackung nach 5 Tagen	68
3.6	Vergleich der Varianten	69
3.6.1	Klemmen	69
3.6.1.1	Einfache Klemme	70
3.6.1.2	Hebel-Klemme	70
3.6.1.3	Gummiband-Klemme	71
3.6.2	Klemmen Wahlvariante	71
3.6.3	Futterschüsseln	72
3.6.3.1	Drehfutterplatte	72
3.6.3.2	Futterplatte Zylinder	72
3.6.3.3	Platte mit einer Schüssel	73
3.6.4	Futterschüssel Wahlvariante	73
3.6.5	Futtermagazine	73
3.6.5.1	Futtermagazin Horizontal	74
3.6.5.2	Futtermagazin Vertikal	74
3.7	Konstruktion der Wahlvariante und Details	75
3.7.1	Drehplatte	75
3.7.2	Förderband, Kettenglied, Kettenrad und Welle	76
3.7.3	Walze	78
3.7.4	Hebelklemme	79
3.7.5	Gesamtansicht der Konstruktion	80
3.7.5.1	Ansicht des Pressvorganges und Positionierung der Walze . . .	80
3.7.5.2	Öffnen und befestigen der Futterpackung	80
3.7.5.3	Gesamtkonstruktion	81
3.8	Berechnung und Dimensionierung	82
3.8.1	Berechnung der Hebel-Klemme	82
3.8.2	Berechnung der Welle zur Bewegung der Drehplatte	83
3.8.3	Berechnung der Umfangskraft des Förderbandes	84
3.8.4	Berechnung der Federkonstante	85

3.9	Simulation	86
3.9.1	Simulation der Klemme	86
3.9.2	Simulation der Feder	86
3.10	Bedienung und Wartung	87
3.11	Selbstkritische Analyse und Ausblick	88
4	Elektronik und Mechanik	89
4.1	Anforderung Elektronik Variante 1	89
4.1.1	Aktoren	89
4.1.2	Sensoren	89
4.1.3	Ansteuerungen	89
4.2	Anforderung Elektronik Variante 2	90
4.2.1	Aktoren	90
4.2.2	Sensoren	90
4.2.3	Ansteuerungen	90
4.3	Variante 1	90
4.3.1	Aktoren	90
4.3.1.1	Hubmagnete	90
4.3.1.2	Motoren	91
4.3.2	Sensoren	91
4.3.3	Ansteuerungen	92
4.4	Variante 2	93
4.4.1	Aktoren	93
4.4.1.1	Hubmagnete	93
4.4.1.2	Motoren	93
4.4.1.3	Lüfter	94
4.4.2	Sensoren	94
4.4.3	Ansteuerung	94
4.4.3.1	Motoransteuerung Variante 1	95
4.4.3.2	Motoransteuerung Variante 2	97
4.4.3.3	PWM-Signal-erzeugung für Motoransteuerung	98
4.4.3.4	Optokoppler	103
4.4.3.5	Gesamtschaltplan	104
4.4.3.6	Verkabelung der Motoransteuerung mit dazugehörigen Elementen	105
4.4.3.7	Verkabelung des Sensors mit dazugehörigen Elementen	107
4.4.3.8	Spannungsversorgung	109
4.4.3.9	Raspberry Verkabelung	110
4.4.3.10	Arduino Verkabelung	110
4.5	Mechanische Konstruktion	111
4.5.1	Motorhalterung Futterschlüsseldrehplatte	111

4.5.2	Motorhalterung Förderband	112
4.5.3	Sensorhalterung Futterschüsseldrehplatte	113
4.5.4	Sensorhalterung Förderband	113
4.5.5	Halterung Raspberry	114
4.5.6	Lüfterhalterung für Raspberry	114
4.5.7	Förderband Stützen	114
4.6	Zusammenfassung und Selbstkritik	116
4.6.1	Relevante Meilensteine	116
4.6.2	Probleme	116
4.6.3	Zukünftige Verbesserungen	116
4.6.4	Tatsächlicher Zeitpunkt der erreichten Meilensteine	117
4.7	Stückliste	117
4.8	Quellenverzeichnis	118
A	Abbildungsverzeichnis	121
B	Tabellenverzeichnis	125
C	Listings	127
D	Abkürzungsverzeichnis	129
E	Literatur	131

KAPITEL 1

Webserver und Client

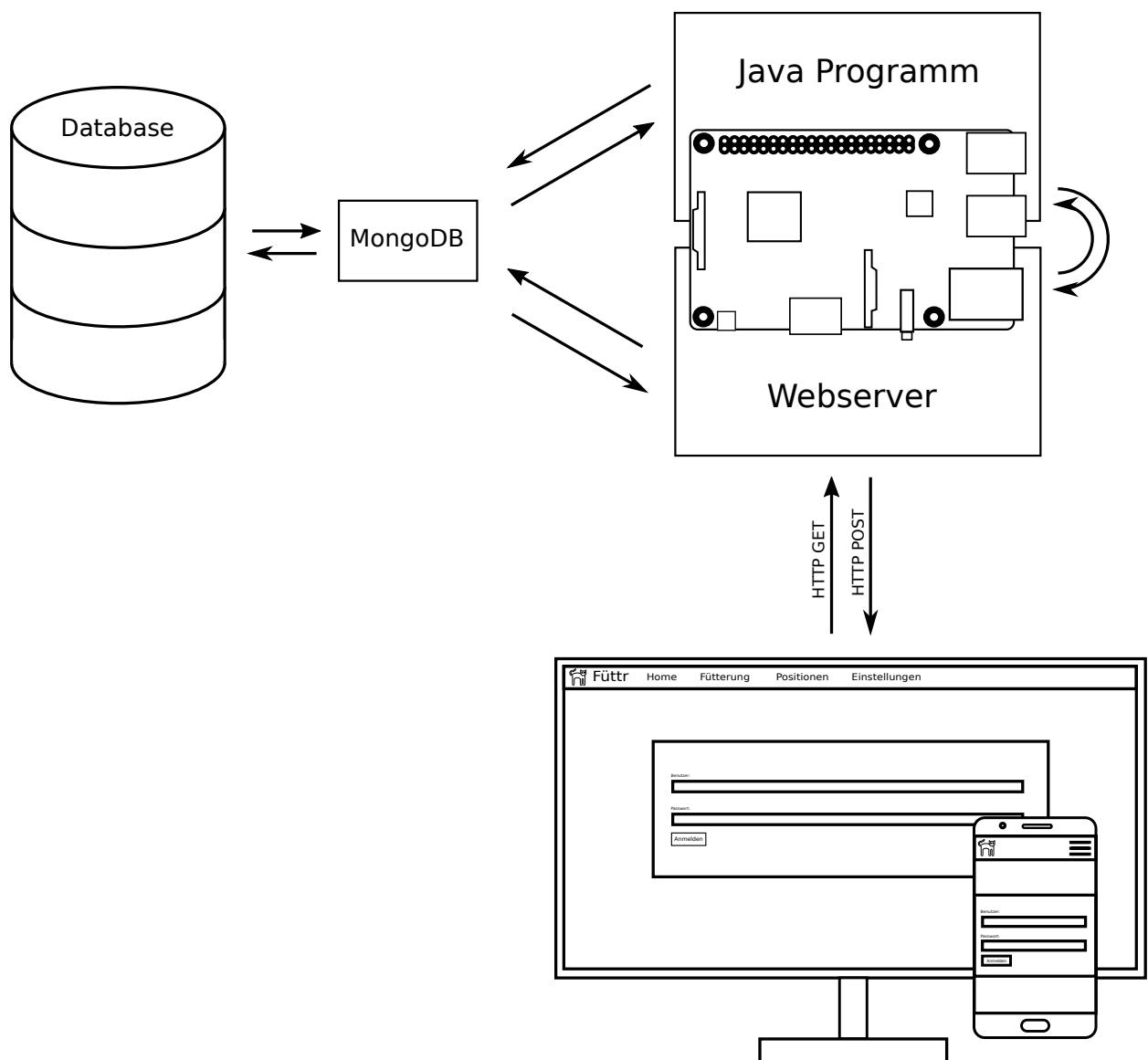


Abbildung 1.1: Kommunikationen innerhalb des Projekts

1.1 Begriffserklärungen

1.1.1 Server

Ein Programm, das Zugriff auf eine Ressource oder einen Dienst in einem Netzwerk ermöglicht

1.1.2 Client

Ein Programm, das auf einen Server zugreift

1.2 Anforderungen

1.2.1 Webserver

Auf der Katzenfütterungsanlage läuft ein Webserver, der es ermöglicht, dass der Benutzer das Gerät über das Internet erreichen kann. Hauptaufgaben des Servers sind dabei, Daten bereitzustellen, zu verarbeiten und zu speichern und den Webclient zur Verfügung zu stellen.

1.2.2 Client

Der Client soll dem Benutzer ermöglichen, die Katzenfütterungsanlage über einen Webbrower zu steuern. Ein Benutzername und ein Passwort sind erforderlich, damit man das Gerät bedienen kann. Das Design soll eindeutig und übersichtlich gehalten sein. Auf der Startseite sollen die eingestellten Fütterungszeiten und eine allgemeine Übersicht zu sehen sein. Über eine Navigationsleiste sollen die weiteren Seiten erreichbar sein:

- Fütterungszeiten
- Positionsinfo
- Geräteinfo
- Update

1.3 Voruntersuchung

1.3.1 HTTP/HTTPS

Das HyperText Transfer Protocol (HTTP)¹ ist der Kommunikationsstandard auf dem das Internet basiert. Eine HTTP Session wird über eine Anfrage über das Transmission Control Protocol (TCP)

¹Weitere Informationen zu HTTP unter <https://developer.mozilla.org/de/docs/Web/HTTP>

an einen Server auf den Port 80 initiiert. Der Server, der auf diesem Port auf eine Anfrage wartet, sendet eine Statusmeldung wie z.B. `HTTP/1.1 200 OK` und eventuell eine eigene Nachricht zurück. Diese Nachricht ist meist die angeforderte Resource oder eine Fehlermeldung. HyperText Transfer Protocol Secure (HTTPS) ist die verschlüsselte Version von HTTP. Die meist gebrauchten Anfragen sind:

- **GET**: Fordert eine Repräsentation der Ressource an. Ein GET Request darf nur Daten abfragen und darf keinen anderen Einfluss haben.
- **PUT**: Fordert das Speichern der Daten, die sich im Request-Body befinden, an. Wenn bereits eine Ressource an der angegebenen Uniform Resource Identifier (URI) existiert, so wird diese aktualisiert, sonst wird die Resource erstellt.
- **POST**: Fordert das Speichern der Daten, die sich im Request-Body befinden, unter der angegebenen URI an.
- **DELETE**: Fordert das Löschen der Ressource unter der angegebenen URI an.

1.3.1.1 HTTP Status Codes

Bei jeder Anfrage sendet ein Webserver zumindest einen Statuscode als Antwort zurück.

- **1xx: Information** Alle Statuscodes die mit 1 beginnen sind rein informelle Codes.
- **2xx: Erfolg** Statuscodes die mit 2 beginnen sagen dem Empfänger, dass seine Anfrage verstanden und akzeptiert wurde.
- **3xx: Umleitung** Diese Statuscodes indizieren dem Client, dass er weiter Schritte durchführen muss, um die Anfrage zu vervollständigen.
- **4xx: Client Fehler** Ein Statuscode, der mit 4 beginnt, bedeutet, dass ein Fehler auf der Seite des Clients aufgetreten ist.
- **5xx: Server Fehler** Wenn der Statuscode mit einer 5 beginnt, bedeutet das, dass auf der Seite des Servers ein Fehler aufgetreten ist.

1.3.2 JavaScript

JavaScript ist die Programmiersprache des Internets. Jeder herkömmliche Browser ist in der Lage, JavaScript auszuführen. Mit JavaScript ist es möglich, das Aussehen einer Webseite während der Laufzeit zu ändern, Dinge zu entfernen, hinzuzufügen und zu animieren. JavaScript ist heute eine objektorientierte Sprache. Es hebt sich von anderen Sprachen vor allem dadurch ab, dass die Datentypen von Variable nicht fix sind. Das bedeutet, wenn eine Variable den Datentyp *number* hat und es wird ein String angehängt, verändert sich der Datentyp automatisch auf *string*. JavaScript kann nur in einem Thread laufen, das bedeutet, es gibt kein echtes Multithreading. Damit etwas

Ähnliches erzielt werden kann, gibt es die Möglichkeit von Callback Methoden. Diese werden ausgeführt, sobald etwas vorher abgeschlossen wurde. Damit ist es möglich, Dinge, die länger brauchen, im "Hintergrund" laufen zu lassen².

1.3.3 TypeScript

TypeScript ist eine, von Microsoft entwickelte, Weiterentwicklung von JavaScript. Das bedeutet, jeder gültige JavaScript Code ist auch ein gültiger TypeScript Code. TypeScript wird vom TypeScript Compiler in sauberes JavaScript übersetzt. TypeScript ist sehr gut für größere Anwendungen geeignet. Typescript hat strenge Datentypen, Klassen und Vererbung. Die Datentypen von TypeScript sind:

- **string**: eine Unicode codierte Zeichenkette, von JavaScript übernommen
- **number**: eine vorzeichenbehaftete Gleitkommazahl, kann auch hexadezimal, octal oder binär sein, von JavaScript übernommen
- **boolean**: true oder false, von JavaScript übernommen
- **array**: eine Liste von Elementen des gleichen Datentyps, von JavaScript übernommen
- **tuple**: eine Liste von Elementen unterschiedlichen Datentyps, deren Anzahl bekannt ist
- **enum**: eine Möglichkeit, numerischen Werten Namen zu geben
- **any**: Datentyp unbekannt, wird behandelt wie in JavaScript
- **void**: kein Datentyp, meist Rückgabewert bei Funktionen
- **null**: leerer Wert, kann allen anderen zugewiesen werden, von JavaScript übernommen
- **undefined**: kein Wert, kann allen anderen zugewiesen werden, von JavaScript übernommen
- **never**: wenn ein Wert niemals auftreten kann z.B. eine Funktion die immer einen Fehler produziert

Damit JavaScript Module von TypeScript verwendet werden können, benötigen sie sogenannte Type Annotations. Diese können bei den meisten bekannten Modulen über den Node Package Manager (npm) installiert werden. Diese Pakete haben den Namenspräfix `@types/`, das bedeutet, dass zum Beispiel die Type Annotations des Express Moduls über `npm install --save-dev @types/express` installiert werden können. Sollten keine Type Annotations für ein Modul vorhanden sein, muss man diese selbst erstellen.

²Mehr Information zu Callbacks, Promises und JavaScript unter <https://www.w3schools.com/js/default.asp>

1.3.4 HTML

HyperText Markup Language (HTML) ist eine Markup Language, zu Deutsch Auszeichnungssprache, dies bedeutet, sie ist keine Programmiersprache, sondern beinhaltet nur Informationen, wie etwas dargestellt werden soll. Eine HTML Datei ist rein textuell und wird erst durch ein Programm, z.B. einen Browser, formatiert dargestellt. HTML ist der Standard für alle Websites im Internet. Eine HTML Datei kann aus drei Teilen bestehen. Dem eigentlichen HTML, JavaScript und Styles. JavaScript und Styles können zur besseren Übersicht in eigenen Dateien ausgelagert werden.

1.3.5 CSS

Cascading Style Sheets (CSS) sind die Style Dateien einer Website. In einer CSS Datei steht, wie ein HTML Element aussieht. Mithilfe von Styles kann z.B. die Farbe, Größe, Platzierung, Schriftart und vieles mehr eines Elementes bestimmt werden. Da das Design in eine eigene Datei ausgelagert wurde, muss es eine Zuweisung zwischen CSS und HTML geben. Dafür gibt es drei Arten. Entweder man weist es jedem Element dieses Typs zu, nur einem Element mit einer bestimmten ID oder jedem Element einer bestimmten Klasse. Mit dem Schlüsselwort `!important` kann man bestimmen, welcher Style bevorzugt genommen wird, wenn eine Doppelbelegung existiert.

```
1 body { // Zuweisung an alle Elemente eines Typs
2     background-color: cyan;
3 }
4
5 #title { // Zuweisung an ein Element mit einer ID
6     color: red;
7 }
8
9 .text { // Zuweisung an alle Elemente einer Klasse
10    border: 1px solid black;
11 }
```

1.3.6 DOM

Das Document Object Model (DOM) ist die Objektrepräsentation des HTML Dokuments. Durch das DOM kann eine JavaScript Anwendung HTML Elemente ändern, entfernen oder hinzufügen, sowie HTML Attribute ändern, entfernen oder hinzufügen und CSS Styles ändern. Das DOM wird vom Browser beim Laden der Website erstellt.

1.3.7 Node.js

Node.js ist eine Laufzeitumgebung, die es ermöglicht, dass Javascript direkt auf einem Rechner ausgeführt werden kann. Node.js kommt mit dem npm. Mithilfe dieses Tools ist es möglich,

Module zu installieren, updaten, löschen und veröffentlichen. Diese Module werden im Ordner `/node_modules` installiert und in der Datei `package.json` unter `dependencies`, oder mit der option `--save-dev` unter `dev-dependencies` eingetragen. Ein neues Projekt erstellt man mit `npm init`. Dieses Tool erstellt die Datei `package.json`, in der alle Abhängigkeiten und Informationen über das Projekt stehen. Wenn man ein Projekt kopiert, braucht man den `/node_modules` Ordner nicht mitkopieren. Man muss nur im Zielordner einmal `npm install` aufrufen.

1.3.8 express

Express ist ein Javascript Modul, das auf dem Node.js Modul `http` bzw `https` aufbaut. In diesen Modulen ist bereits alles enthalten, was benötigt wird, um einen Webserver zu programmieren. Express nimmt uns die meiste Arbeit ab und bietet viele weitere Möglichkeiten³.

1.3.9 JSON

JavaScript Object Notation (JSON) ist die Textrepräsentation eines JavaScript Objekts⁴. Die möglichen Datentypen sind:

- **string**: 0 oder mehrere unicode Zeichen innerhalb Doppelhochkommas
- **boolean**: true oder false
- **number**: Eine vorzeichenbehaftete Zahl, die auch die E Notation unterstützt z.B. 0.2E4 (=2000)
- **Array**: Eine geordnete Liste von 0 oder mehreren Werten innerhalb eckiger Klammern, Elemente sind getrennt durch Kommas.
- **Object**: Eine ungeordnete Sammlung von Name-Wert-Paaren, wo die Namen, die auch Keys genannt werden, Strings sind. Jeder Key sollte eindeutig sein. Befindet sich innerhalb geschwungener Klammern. Paare sind durch Komma getrennt
- **null**: Ein leerer Wert

```

1  {
2      "Object": {
3          "string": "name",
4          "number": 10E5,
5          "boolean": true,
6          "Array": [
7              {
8                  "string": "wert",
9                  "number": 1

```

³Weitere Information zu express unter <https://expressjs.com/>

⁴Weitere Informationen zu JSON unter <https://www.json.org/>

```

10      },
11      {
12          "string": "wert",
13          "number": 2
14      }
15  ]
16 }
17 }
```

Listing 1.2: JSON Beispiel

1.3.10 JSON Web Token

JSON Web Token (JWT) ist ein JSON-basierter offener Standard für das Erstellen von Access Tokens. Mithilfe eines JWT kann ein Client sich ausweisen. Ein JWT wird vom Server entweder mit einem Secret oder seinem privaten Schlüssel signiert. Dadurch können Server und Client überprüfen, ob der Token legitim ist. Ein JWT besteht aus drei Teilen: dem Header, der Payload und der Signature. Im Header steht der fürs Verschlüsseln der Signatur benutzte Algorithmus z.B.:

`{"alg": "RS256", "typ": "JWT"}`. Im Payload stehen die Daten, die entweder den Client ausweisen oder ähnliche Information. Beispiel: `{"user": "cat", "iat": 1520875121, "exp": 1520911121}` *iat* bedeutet *issued at* und sagt aus, wann der Token generiert wurde. In der Signature stehen der Key, der unsignierte Token, das ist der Header und die Payload Base64 codiert, und die Signatur. Alle drei Teile werden Base64 codiert und mittels Punkt voneinander getrennt.

1.3.11 MongoDB

MongoDB ist eine schemenlose Datenbank. Schemenlos bedeutet, dass die Datenbank im Vergleich zu schemenbehafteten Datenbanken keine klare Strukturierung benötigt. Einer schemenlosen Datenbank kann man einfach Daten geben und wieder abfragen. Eine schemenbehaftete Datenbank ist in Zeilen und Spalten unterteilt. Diese müssen vorher feststehen. Da Raspbian, das Betriebssystem vom Raspberry Pi, nur 32 Bit ist und MongoDB ab Version 3 nur mehr in 64 Bit erhältlich ist, mussten wir auf eine ältere Version wechseln. Die Verbindung der MongoDB Datenbank erfolgt über einen Driver. Der Driver muss mit der Datenbankversion übereinstimmen. MongoDB ist ein Database Management System (DBMS). Das bedeutet, ein Server läuft auf dem Port 27017, über den alle Datenbanken im System erreichbar sind, beispielsweise ist die Datenbank `fuettr` über `localhost:27017/fuettr` erreichbar. Eine Gruppe von Daten nennt man Collection. Zugriff auf die Datenbank erfolgt serverseitig wie folgt:

```
1 const dbServer = await mongoDB.MongoClient.connect(url);
```

Listing 1.3: Verbinden mit dem DBMS

```
1 const dbFuettr = await dbServer.db('fuettr');
```

Listing 1.4: Auswählen der Datenbank

```
1 const collTimes = await dbFuettr.collection('data_times');
```

Listing 1.5: Auswählen der Collection

```
1 const Times = await this._times.find({ identifier: 'Times' }).toArray();
```

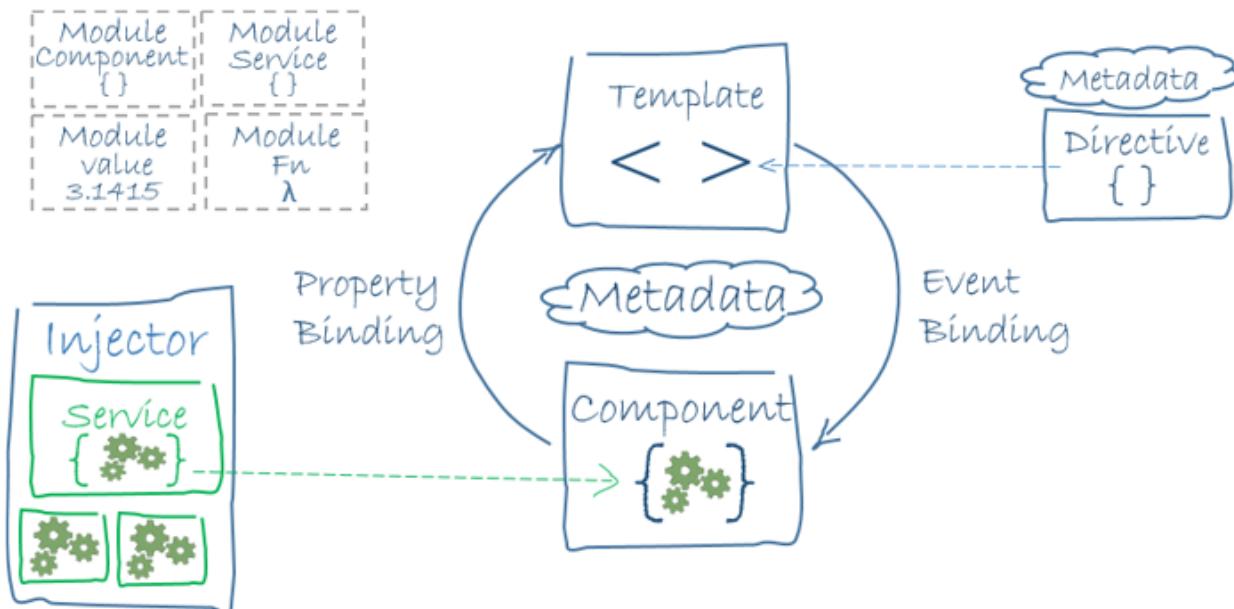
Listing 1.6: Auslesen aller Datensätze mit einem Identifier

```
1 this._times.updateOne({ identifier: 'Times' }, { $set: times });
```

Listing 1.7: Überschreiben eines Datensatzes mit einem Identifier

1.3.12 Angular 2/4

Angular ist ein TypeScript Framework, das aus dem Javascript-Framework AngularJS weiterentwickelt wurde. Es wird von Google entwickelt. Angular ist gegliedert in Module. Die grobe Struktur wird in der Abbildung 1.2 dargestellt⁵.

Abbildung 1.2: Angular Struktur⁶

1.3.12.1 Modules

Jede Angular App ist in *Modules* gegliedert. *Modules* fassen meist ähnliche Funktionen zusammen. Jede App muss mindestens ein *Module* enthalten. Dies heißt standardmäßig *AppModule*. Ein *Module* ist die größte Einheit einer Angular App. Ein *Module* kann folgende Komponenten beinhalten:

⁵Genauere Informationen und Tutorials unter <https://angular.io/>

⁶Abbildung 1.2 Quelle: <https://angular.io/guide/architecture#whats-next> (besucht am 10.3.2018)

- Services
- Andere Module
- View Classes
 - Components
 - Directives
 - Pipes

1.3.12.2 Libraries

Eine Angular Library ist ein Modul, das *Modules* exportiert. Diese können von *Components* und *Modules* importiert werden. Der Name jeder Angular library beginnt mit `@angular`. Angular libraries können mit dem npm installiert werden.

1.3.12.3 Components

Ein *Component* kontrolliert einen Teil des Bildschirms, den sogenannten *view*. Die Logik des *Components* wird in einer Klasse definiert. Die Klasse interagiert mit dem *view* durch eine Application Programming Interface (API) von Eigenschaften und Methoden.

Lifecycle Hooks

Ein Lebenszyklus in Angular ist immer der Lebenszyklus des jeweiligen *Components*. Ein *Component* kann erstellt, verändert und zerstört werden. Zu jedem Stadium gibt es eine Methode, die automatisch je nach Stadium von Angular aufgerufen wird. Die am häufigsten verwendeten sind:

- **OnInit()**: Wird aufgerufen, wenn der *Component* erstellt wird. Die dazugehörige Methode ist `ngOnInit()`. Zeitaufwendige Operationen sollten anstatt im constructor hier aufgerufen werden, da das den Erzeugungsvorgang verlangsamen würde.
- **OnDestroy()**: Wird aufgerufen, wenn der *Component* zerstört wird. Die dazugehörige Methode ist `ngOnDestroy()`. Hier sollten alle laufenden Prozesse wie Intervals und Callbacks beendet werden.
- **OnChanges()/DoCheck()**: Werden aufgerufen, wenn sich irgend etwas an der *Component* ändert. Die dazugehörige Methoden sind `ngOnChanges()` und `ngDoCheck()`.

Alle Lifecycle Hooks müssen von der Klasse mit `implements` implementiert und von `@angular/core` importiert werden.

1.3.12.4 Templates

Das Aussehen des *views* wird in einem *Template* definiert. Ein *Template* ist eine HTML Datei, mit Angular's Template Syntax. Das bedeutet, dass einige Zusatzbefehle vorkommen können. Beispiele hierfür sind:

- *ngFor
- *ngIf
- {{variable}}
- (click)
- [variable]
- <app-route>

Die meisten Zusatzbefehle werden für *Data Binding* verwendet.

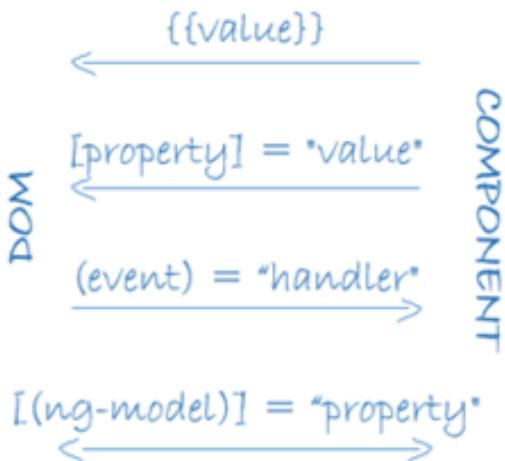


Abbildung 1.3: Angular Databinding⁷

von *data binding* dargestellt. Jede Art hat eine Richtung, vom DOM zum DOM oder in beide Richtungen.

- `{{variable}}` Der Wert der Variable in den Klammern wird anstelle des Klammerausdrucks dargestellt. Wenn sich der Wert ändert, ändert sich auch der Wert im DOM
- `` übergibt den Wert in der angegebenen Variable in das angegebene Attribut
- `` ruft die angegebene Methode auf, wenn das angegebene Event auftritt, in diesem Fall ein click-Event

1.3.12.5 Data binding

Ohne Framework wäre der Programmierer dafür verantwortlich sicherzustellen, dass Daten in HTML Elemente geschrieben werden und dass auf Benutzerinteraktionen reagiert wird. Das ist aufwendig, fehleranfällig und schwer zu lesen. Angulars Lösung dafür nennt sich *data binding*. Der Programmierer muss nur mehr im HTML Template Angular mitteilen, wie die beiden Seiten verbunden werden sollen. In der Abbildung 1.3 sind die vier möglichen Arten

⁷Abbildung 1.3 Quelle: <https://angular.io/guide/architecture-components#data-binding> (besucht am 10.3.2018)

- `<input [(ngModel)] = "variable" >` der Wert im Eingabefeld und in der Variable sind voneinander abhängig, ändert sich der eine, so ändert sich zugleich der andere zum gleichen Wert

1.3.12.6 Services

Services sind Klassen, die eine Aufgabe erfüllen, unabhängig von allem anderen. Sie werden vor allem verwendet, wenn eine Aufgabe etwas Zeit erfordert wie z.B. eine Serveranfrage. Services bieten auch die Möglichkeit, dass *Components* Zugriff auf eine gemeinsame Variable haben, da Angular keine globalen Variablen hat. Services müssen vom *Dependency Injector* injiziert werden. Der *Dependency Injector* indiziert alle Abhängigkeiten in einer *Component*.

1.3.13 Angular CLI

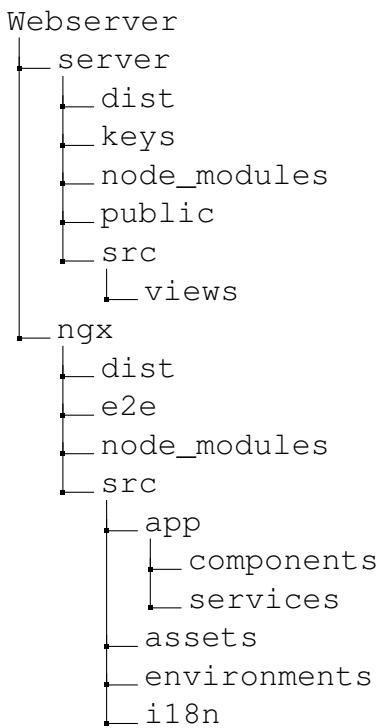
Das Angular Command Line Interface (CLI) ist ein JavaScript Modul, das über den npm installiert werden kann. Mithilfe des CLI ist das Erstellen und Übersetzen von Angular Anwendungen um vieles erleichtert. Ein neues Projekt erstellt man mit `ng new <projektnname>` und übersetzt wird mit `ng build`. Mit der Option `--prod` wird die Anwendung kompakter und für den Einsatz im produktiven Umfeld übersetzt.

1.3.14 Bootstrap

Bootstrap ist eine CSS Bibliothek, die die Möglichkeit bietet, bereits vorgefertigte Komponenten auf unserer Website zu verwenden. An erster Stelle stehen bei Bootstrap responsive Design und Mobilgeräte. Responsive bedeutet, dass die Elemente sich an die Breite des Bildschirms anpassen. Dadurch erspart man sich als nicht sehr designaffiner Programmierer viel Arbeit. Auf der offiziellen Bootstrap Website kann man außerdem verschiedene Themes auswählen. Bootstrap wurde von Twitter entwickelt und ist entweder über den npm oder über Bootstraps eigenes Content Delivery Network (CDN) erhältlich.

1.4 Umsetzung

1.4.1 Projektstruktur



Das Projekt ist so strukturiert, dass eine klare Trennung zwischen Client und Server herrscht.

Server Im *keys* Ordner befinden sich der öffentliche und private Schlüssel, die vom install Script erstellt werden. Im *public* Ordner befinden sich die Ressourcen, die direkt vom Server gesendet werden, z.B. Fehlermeldungsseiten. Im *src* Ordner sind die TypeScript Arbeitsdateien. Diese werden in den *dist* Ordner übersetzt.

Client Der Client Ordner hat den Namen *ngx*, das sagt aus, es ist ein Angular Projekt, die Angular Version ist aber nicht weiter angegeben. Im *src* Ordner befinden sich alle Dateien, die das Angular CLI benötigt, um die Anwendung zu bauen. Im *assets* Ordner befinden sich alle Ressourcen, die die Anwendung benötigt wie z.B. Bilder. Im Ordner *i18n* befinden sich die Dateien, die das CLI braucht, um die Anwendung in mehrere Sprachen zu übersetzen. Im *app* Ordner befindet sich die eigentliche Anwendung. Die Anwendung ist klar getrennt in den Ordner *components* und *services*.

1.4.2 Client

1.4.2.1 Übersicht

Das Design sollte übersichtlich und einfach gestaltet werden. Der Benutzer soll auf den ersten Blick die wichtigsten Funktionen und Informationen erkennen können.

Auf der Startseite sind alle wichtigen Informationen übersichtlich dargestellt. Auf der linken Seite werden die Uhrzeit der letzten erfolgreichen Fütterung, die Zeit der nächsten Fütterung und die Zeit bis zur nächsten Fütterung dargestellt. Darunter werden Fehler und Warnungen, falls welche auftreten sollten, angezeigt. Da unbekannt ist, wie viele Fehler und Warnungen auftreten, werden diese in einem *ngFor aufgelistet.

Auf der Fütterungszeiten-Seite kann der Benutzer die Katzenfütterungsanlage ein und ausschalten. Dies wurde mit einer Checkbox realisiert, die durch Styles wie ein Schalter gestaltet wurde. Darunter können die Fütterungszeiten geändert und deaktiviert werden. Siehe Abbildung 1.5. Der Button 'Speichern' wird deaktiviert, sobald eine Zeit ungültig eingegeben wurde oder wenn die Zeiten nicht in aufsteigender Reihenfolge sortiert sind.

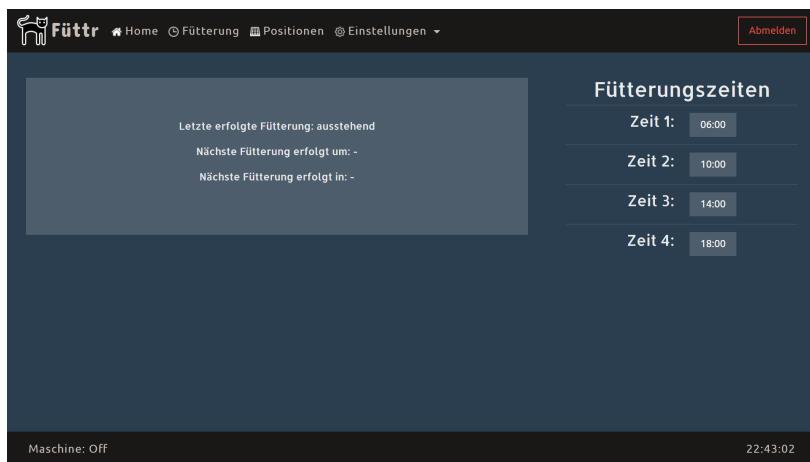


Abbildung 1.4: Startseite

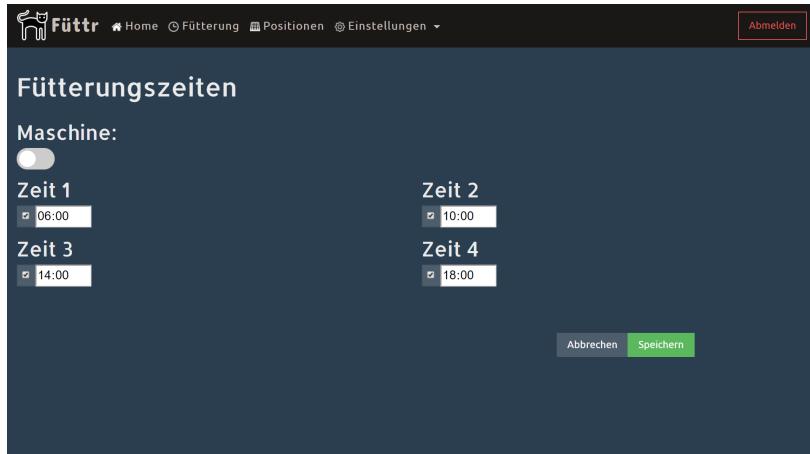


Abbildung 1.5: Fütterungszeiten

Auf der Geräteinformations-Seite werden die wichtigsten Daten über das Gerät angezeigt. diese Daten sind:

- Seriennummer
- Interner Rechner
- WLAN Status
- IP Adresse
- Softwareversion

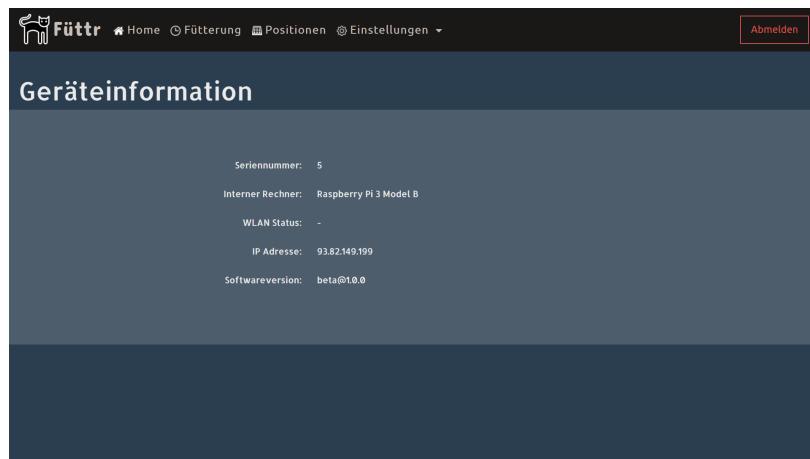


Abbildung 1.6: Geräteinformationen

Die IP-Adresse ist die aktuelle **externe** IP-Adresse.

Auf der Update-Seite kann der Benutzer nach Updates suchen, Updates starten oder die Maschine herunterfahren. Wenn der Benutzer auf den Herunterfahren-Button klickt, der sich auf der linken Seite ganz unten befindet, wird er gewarnt, dass die Maschine nur mehr über das Aus- und wieder Einsticken des Netzteils startbar ist.



Abbildung 1.7: Update

1.4.2.2 Design

Für das Design der Website wurde Bootstrap ausgewählt, da es bereits viele Positionierungsmöglichkeiten bietet und ohne viel Aufwand im Client verwendet werden kann. Bootstrap ist außerdem für Mobilgeräte und alle Bildschirmgrößen optimiert. Es bietet die Möglichkeit für Dropdownmenüs, Menüleisten und Popups. Diese Funktionen werden bei Bootstrap im Normalfall über JQuery⁸ gesteuert, aber da JQuery eine weitere externe JavaScript Bibliothek ist, wurde stattdessen das JavaScript Modul *ng-bootstrap* verwendet. Dieses ist eine Erweiterung für Angular, mit dem sich die Bootstrap Komponenten steuern lassen. Damit dies funktioniert, muss *ng-bootstrap* im *Module* unserer Applikation eingebunden werden.

⁸Mehr zu JQuery unter <https://jquery.com/>

```

1 ...
2 import { NgbModule, NgbDropdown } from '@ng-bootstrap/ng-bootstrap';
3 ...
4 imports: [NgbModule.forRoot(), ...]
5 ...

```

Listing 1.8: app.module (Auszug) Importieren von ng-bootstrap

Da Bootstrap ohne etwas zu verändern etwas langweilig aussieht, entschied ich mich für das *Superhero* Thema von der Seite <https://bootswatch.com/superhero/>. Da mir die orange Farbe nicht gefiel, änderte ich sie auf einen dunkleren Grauton, den ich aus einem Bild einer süßen Babykatze genommen habe. Bootstrap hat außerdem eine Standardschriftart, die ich geändert habe. Alle Schriftarten sind von der Seite <https://fonts.googleapis.com/> genommen worden, da diese in Google's CDN sind und somit ziemlich sicher die meiste Zeit verfügbar sind. Gewählt wurden die Schriftarten Ubuntu für alle Knöpfe und die Navigationsleiste, Allerta für alle Texte und Concert One für das Logo. Damit die Schrift auf allen Geräten gut lesbar ist und nicht zu groß wird, mussten die Schriftgrößen der Bildschirmbreite entsprechend angepasst werden.

```

1 @media only screen and (max-width: 576px) {
2   body {
3     font-size: 0.5rem !important;
4   }
5   .bigger-h1 {
6     font-size: 1.4rem !important;
7   }
8 }
9 @media only screen and (max-width: 768px) {
10  body {
11    font-size: 0.7rem !important;
12  }
13  .bigger-h1 {
14    font-size: 1.6rem !important;
15  }
16 }
17 @media only screen and (max-width: 992px) {
18  body {
19    font-size: 0.9rem !important;
20  }
21  .bigger-h1 {
22    font-size: 1.8rem !important;
23  }
24 }
25 @media only screen and (max-width: 1200px) {
26  body {
27    font-size: 1.1rem !important;
28  }
29  .bigger-h1 {
30    font-size: 2rem !important;

```

```
31      }
32 }
```

Durch `@media only screen and (max-width: ZAHLPx)` wird die Bildschirmbreite bestimmt und alle Styles innerhalb der Klammern werden nur dann angewandt, wenn der Ausdruck in der Klammer zutrifft. In der Klammer wird die Schriftgröße des `body` Elements und allen Elementen der Klasse `.bigger-h1` angepasst.

Login

Auf der Login Seite befindet sich rechts oben ein Dropdownmenü, in dem die Sprache geändert werden kann. Dies ist gelöst durch ein `*ngFor`. Wenn das Menü geschlossen ist, wird dort die Flagge der aktuellen Sprache angezeigt, Österreich bei Deutsch und Großbritannien bei Englisch. Wenn man auf das Menü klickt, klappen alle Sprachen herunter, die zur Auswahl stehen und der Benutzer kann auf eine klicken. Die verschiedenen Auswahlmöglichkeiten sind Links, die den Benutzer auf den jeweiligen Sprachcode umleiten und die Seite somit in der gewünschten Sprache anzeigen. In der Mitte der Seite befindet sich ein Bootstrap Jumbotron. Ein Jumbotron ist dafür gedacht, auf etwas Aufmerksamkeit zu ziehen. Damit verändert sich die Hintergrundfarbe und es entsteht eine klare Abhebung. Innerhalb des Jumbotrons ist ein HTML Formular, in das der Benutzername und das Passwort vom Benutzer hineingeschrieben werden. Beim Klick auf den Knopf *Anmelden* werden diese Daten dem Server übermittelt. Sollte das Anmelden gegliickt sein, wird automatisch die Startseite angezeigt. Auf allen weiteren Seiten ist rechts oben ein Knopf, mit dem sich der Benutzer abmelden kann.

Startseite

Auf der linken Seite befindet sich ein Jumbotron, in dem die letzte Fütterungszeit, die nächste Fütterungszeit und die Zeit bis zur nächsten Fütterung angezeigt werden. Auf der rechten Seite werden die aktiven Fütterungszeiten angezeigt. Sollte eine Zeit deaktiviert werden, so wird sie auch nicht in dieser Liste angezeigt. Der Jumbotron und die Liste sind mithilfe von Bootstrap in einem Verhältnis von 2:1 nebeneinander angeordnet. Sollte die Bildschirmbreite zu klein werden, weil die Seite auf einem Mobilgerät angezeigt wird, oder weil der Benutzer das Browserfenster klein darstellt, so werden die beiden Elemente untereinander dargestellt. Unterhalb des Jumbotrons werden auftretende Fehler und Warnungen angezeigt. Dies ist wiederum mit zwei `*ngFor` gelöst. Die Fehler und Warnungen sind Bootstrap Alerts. Sie sind dafür gedacht, dem Benutzer auf etwas aufmerksam zu machen. Sie können durch ein x auf der rechten Seite geschlossen werden. Am unteren Bildschirmrand ist eine Leiste, die die gleiche Farbe wie die Navigationsleiste hat. Auf ihr wird links angezeigt, ob die Maschine ein- oder ausgeschaltet ist.

Fütterungszeiten

Unter der Überschrift befindet sich ein Schalter, der es dem Benutzer ermöglicht, die Maschine

ein- und auszuschalten. Dieser Schalter ist eine normale HTML Checkbox, die durch CSS Styling wie ein Schalter designet wurde.

Listing 1.10: HTML Schalter

```
1 <label class="switch">
2   <input type="checkbox" [(ngModel)]="machine_state">
3   <span class="slider"></span>
4 </label>
```

```
1 .switch {
2   position: relative;
3   display: inline-block;
4   width: 60px;
5   height: 34px;
6 }
7
8 .switch input {
9   display: none;
10}
11
12 .slider {
13   border-radius: 34px;
14   position: absolute;
15   cursor: pointer;
16   top: 0;
17   left: 0;
18   right: 0;
19   bottom: 0;
20   background-color: #ccc;
21   -webkit-transition: 0.4s;
22   transition: 0.4s;
23 }
24
25 .slider :before {
26   border-radius: 50%;
27   position: absolute;
28   content: '';
29   height: 26px;
30   width: 26px;
31   left: 4px;
32   bottom: 4px;
33   background-color: white;
34   -webkit-transition: 0.4s;
35   transition: 0.4s;
36 }
37
38 input :checked + .slider {
```

```

39   background-color: #2196f3;
40 }
41
42 input :focus + .slider {
43   box-shadow: 0 0 1px #2196f3;
44 }
45
46 input :checked + .slider :before {
47   -webkit-transform: translateX(26px);
48   -ms-transform: translateX(26px);
49   transform: translateX(26px);
50 }
```

Mit dem `:before` Selector wird ein Element vor dem tatsächlichen Element eingefügt. Dies wird gemacht, da das `input` Element nicht so dargestellt werden kann, wie wir es gerne möchten. Mit `display: none;` blenden wir das `input` Element aus. Mit `transform:`, `-webkit-transform:` und `-ms-transform:` wird der Schalter animiert und es wird sichergestellt, dass alle herkömmlichen Browser es unterstützen. Darunter kann der Benutzer die Fütterungszeiten einstellen. Die `input` Elemente sind durch Bootstrap in zwei Spalten im Verhältnis 1:1 angeordnet. Darunter befinden sich zwei Knöpfe, mit denen man die Eingabe der Zeiten abbrechen und bestätigen kann. Bei erfolgreicher Speicherung wird neben dem Speichern ein Knopf eingeblendet, auf dem *Gespeichert* steht. Dies wird durch Angular Animationen ermöglicht, siehe 1.4.2.3. Sollte das Speichern fehlgeschlagen, sei es durch Verbindungsprobleme oder einem falschen bzw. abgelaufenen Tokens, so wird unter dem Speichern Knopf über die ganze Seite ein Alert angezeigt, der dem Benutzer mitteilt, dass das Speichern fehlgeschlagen ist. Alle Eingegebenen Daten werden laufend überprüft, siehe 1.4.2.3.

Positionen

Auf dieser Seite befinden sich vier Elemente, die mithilfe von Bootstrap in vier gleich großen Spalten angeordnet sind. Wenn die Bildschirmbreite zu klein wird, so werden sie untereinander angezeigt.

Geräteinformationen

In der Mitte der Seite befindet sich ein Jumbotron, der die ganze Seitenbreite umfasst. In ihm ist eine Tabelle, die mit der Bootstrap Klasse `.table-hover` gestyled wurde. Diese Klasse bewirkt, dass die Zeile, über der der Mauszeiger sich gerade befindet, hervorgehoben wird. Dadurch kann der Benutzer leichter erkennen, in welcher Zeile er gerade liest.

Aktualisierungen

Auf dieser Seite befindet sich nur ein Knopf in der Mitte, der ein Popup, ein sogenanntes Modal, anzeigt. Dies ist durch *ng-bootstrap* gelöst. Bei Aufruf der Seite wird überprüft, ob eine Software-

aktualisierung verfügbar ist. Im Modal wird dann angezeigt, ob eine Aktualisierung verfügbar ist und wie lange es gedauert hat, bis die Überprüfung abgeschlossen war. Sollte eine Aktualisierung verfügbar sein, so wird ein Knopf angezeigt, über den der Benutzer den Aktualisierungsvorgang starten kann. Wird der Aktualisierungsvorgang gestartet, so wird der Knopf deaktiviert, sodass er nicht mehr anklickbar ist. Ein Ladebalken zeigt dem Benutzer, dass der Vorgang unbestimmte Zeit dauert. In dem Ladebalken steht ein Text, der dem Benutzer mitteilt, in welchem Stadium das Raspberry sich befindet, siehe 1.4.2.3.

1.4.2.3 Funktion

Struktur

Der Haupt-*Component*, der geladen wird, wenn die Seite aufgerufen wird, ist *app.component*. In seinem Template ist die Navigationsleiste, die sich alle Seiten teilen. Alle weiteren Seiten werden im `<router-outlet></router-outlet>` angezeigt. Das ist ein HTML-Tag, der von Angulars Router zur Verfügung gestellt wird. Ein Router ist ein Service, welches das Navigieren der Seite ermöglicht. Damit das funktioniert, muss dem Router mitgeteilt werden, wie die Seite aufgebaut ist. Dies geschieht im *router.module*:

```

1 const routes: Routes = [
2   { path: 'home', component: HomeComponent,
3     data: { title: 'Füttr' }, canActivate: [AuthGuard] },
4   { path: '', pathMatch: 'full', redirectTo: 'login' },
5   { path: 'login', component: LoginComponent,
6     data: { title: 'Füttr - Login' } },
7   { path: 'position', component: PositionComponent,
8     data: { title: 'Füttr - Positions' }, canActivate: [AuthGuard] },
9   { path: 'feed', component: FeedComponent,
10    data: { title: 'Füttr - Feeding-Cycle' }, canActivate: [AuthGuard],
11   { path: 'info', component: InfoComponent,
12     data: { title: 'Füttr - Info' }, canActivate: [AuthGuard] },
13   { path: 'update', component: UpdateComponent,
14     data: { title: 'Füttr - Update' }, canActivate: [AuthGuard] },
15   { path: '**', component: Error404Component,
16     data: { title: 'Füttr - 404 (not found)' } }
17 ];

```

Listing 1.12: Routes Definition

Jede Route bekommt ein *data* Objekt übergeben, das bei Aufruf der *Component* dem @Input Decorator injiziert wird. Der jeweilige *Component* ändert damit den Titel der Seite. Alle Routes, außer *login* und *404* sind von einem AuthGuard geschützt, das bedeutet, nur autorisierte Benutzer können diese Routes aufrufen. Der Request auf *root path: ''* wird umgeleitet auf *login*. Für jeden Request, dessen Route nicht definiert ist, wird der 404 Component geladen: *path: '**'*. Um auf eine andere Seite innerhalb der Applikation zu gelangen, gibt es zwei Möglichkeiten. Einmal über ein HTML Element, auf das der Benutzer klickt mittels HTML Attribut *routerLink* oder auf der TypeScript Seite.

Listing 1.13: Weiterleitung mittels HTML Attributes

```
1 <a class="nav-link" routerLink="/feed" i18n>...</a>
1 this.router.navigateByUrl('/home');
```

Listing 1.14: Weiterleiten mittels TypeScript

HTTP Service

Da alle *Components* Daten vom Server holen und manche auch Daten zum Server schicken, war es die logische Schlussfolgerung, ein Service zu erstellen, das jede Art von HTTP Anfrage verarbeiten kann. Dieses Service ist *http.service*. Alle Anfragen, egal ob *GET* oder *PUT* werden von diesem Service getätig. Das Service importiert Angular's *HttpClient*, mit dem es sehr einfach ist, Anfragen zu erstellen und die Antworten zu verarbeiten. Die Antworten vom Server sind vom Typ *application/json*, das bedeutet, sie sind JSON-Dateien, die der *HttpClient* dann in Objekte umwandelt. Diese Objekte werden dem *Component* übergeben, sobald die Anfrage abgeschlossen wurde.

Fütterungszeiten

Die Eingabe wurde realisiert durch HTML input Elemente vom Typ *time*. Da *time* browserabhängig ist und ältere Browser es als *text* interpretieren könnten, und die Application auf allen Browsern funktionieren soll, werden alle Eingabefelder auf Richtigkeit überprüft. Außerdem sollen alle Zeiten in aufsteigender Reihenfolge eingegeben werden. Der *time* Typ ist eigentlich nur ein string, das bedeutet, um die Reihenfolge zu überprüfen, müssen diese zuerst in Zahlen umgewandelt werden. Das geht mit der Funktion `parseInt()`. Ein string von einem *time* input sieht meistens so aus: `'12:34'` Wenn die Zeit leer ist, sieht er so aus: `'--:--'` Auf diese beiden Arten muss unser Parser vorbereitet sein. Der Parser wandelt den string in Minuten um, danach werden sie verglichen. Sollte eine Zeit ungültig sein oder sollte eine Zeit außer Reihenfolge sein, so wird der Speicher-Button deaktiviert. Überprüft wird, sobald der User etwas ändert. Dies geschieht durch den `DoCheck()` Lifecycle Hook (siehe 1.3.12.3). Über den Fütterungszeiten kann man außerdem die Maschine ein und ausschalten.

Listing 1.15: Zeiten Parser

```
1 toMinutes(time: string): number {
2   let timeHours: number;
3   let timeMinutes: number;
4   if (time[0] === '-' && time[1] === '-')
5     && time[3] === '-' && time[4] === '-') {
6     return null;
7   }
8   timeHours = parseInt(time[0], 10) * 10;
9   timeHours = timeHours + parseInt(time[1], 10);
10  timeMinutes = parseInt(time[3], 10) * 10;
11  timeMinutes = timeMinutes + parseInt(time[4], 10);
```

```

12     return timeHours * 60 + timeMinutes;
13 }

```

Neben dem Speichern Knopf wird bei erfolgreicher Speicherung ein weiterer Knopf animiert. Dies geschieht durch Angular Animationen.

```

1  animations: [
2    trigger('SaveAnimation', [
3      state('false', style({ opacity: '0.0' })),
4      state('true', style({ opacity: '1.0' })),
5      transition('* => *', animate('300ms'))
6    ]),
7    trigger('FailAnimation', [
8      state('false', style({ opacity: '0.0' })),
9      state('true', style({ opacity: '1.0' })),
10     transition('* => *', animate('300ms'))
11   ])
12 ]

```

Listing 1.16: Animieren des Knops

Bei dem zu animierenden Element wird `[@NameDerAnimation] = "auslöserVariable"` in die Attribute geschrieben. Ändert sich diese Variable, so wird die Animation gestartet. Animiert werden kann alles, dass durch Styles bestimmt werden kann.

Updates

Auf der Updates Seite kann der Benutzer nach neuen Updates suchen und das Gerät updaten. Realisiert wurde dies durch eine Anfrage an Github, wo eine Datei namens *version.json* liegt. Diese wird verglichen mit der Datei, die auf dem System liegt. Dieser Vergleich geschieht bereits im OnInit() Lifecycle Hook (siehe 1.3.12.3). Der User kann nochmal nach Updates suchen, wenn er dies möchte. Sollte ein Update verfügbar sein, so wird der Button aktiv und der Benutzer kann sich dazu entscheiden, dass Gerät upzudaten. Wird aktualisiert, so wird eine Anfrage an den Server gesendet siehe 1.4.3.2. Der Client beginnt, dauerhaft Anfragen zu senden, bis eine fehlschlägt. Eine Nachricht teilt dem Benutzer mit, dass das System aktualisiert. Dann werden so lange Anfragen gesendet, bis wieder eine Antwort ankommt. Dann wird die Seite automatisch neu geladen. Der Updatevorgang ist damit abgeschlossen.

Login

Das Login funktioniert auf der Basis des Angular Auth Guards. Dieser schützt gewisse Routes gegen unbefugte Benutzer. Im Login bekommen wir, bei erfolgreicher Authentifikation einen JWT vom Server, den wir in einer Variable speichern und dann im `Authorization` Header bei Bedarf mitschicken. Dies geschieht durch einen sogenannten *Interceptor*, der den HTTP Request nimmt und ihn klont, da HTTP Requests in Angular nicht veränderbar sind, und dann zusätzlich den `Authorization` Header hinzufügt.

1.4.2.4 Internationalization

Um die Webseite in mehreren Sprachen anzeigen zu können, muss sie in allen Sprachen extra übersetzt sein. Das würde bedeuten, für jede Sprache wird ein eigenes Angular Projekt benötigt. Das wäre sehr viel Speicherplatz und Rechenleistung. Glücklicherweise bietet Angular eine Möglichkeit, die App in mehrere Sprachen zu übersetzen, ohne die ganze App neu zu schreiben. Diese Möglichkeit nennt sich Internationalization oder kurz i18n. Angular muss nur wissen, welche Teile wie übersetzt werden sollen. Dafür schreiben wir bei allen HTML Elementen, deren Text wir übersetzen wollen i18n zu den Attributen. `<h4 i18n>Zeit 1:</h4>`. Mit dem Kommando `ng xi18n --output-path src/i18n` werden alle Texte in eine Datei namens *messages.xlf* extrahiert. Diese Datei wird dann kopiert, für jede Sprache eine neue Datei. Jede Datei wird entsprechend des Sprachencodes umbenannt, für Deutsch wäre das *messages.de.xlf*. In dieser Datei steht jeder Text in einem `<source>` Tag. Diese Texte werden übersetzt und unterhalb in einen `<target>` Tag geschrieben. Im Falle dieser Arbeit wurden die Texte alle in Deutsch und Englisch übersetzt. Zum Bauen des mehrsprachigen Projekts muss Angular mitgeteilt werden, welche Sprachen und welche Dateien es verwenden soll. Unter Linux geht das mit dem Kommando

```
for lang in en de; do ng build --output-path  
=dist/$lang --aot --prod --bh $lang/ --i18n-file=src/i18n/messages.$lang.xlf  
--i18n-format=xlf --locale=$lang; done . Dieses Kommando baut unsere Applikation in  
den dist Ordner, jede Sprache bekommt dabei ihren eigenen Unterordner, der dem Sprachencode  
entspricht.
```

1.4.2.5 Kommunikation mit dem Server

Der Client benötigt viele Daten vom Server und sendet einige an den Server. Das Übertragungsprotokoll, Tabelle 1.1, gibt klare Auskunft über alle möglichen Anfragen und Antworten.

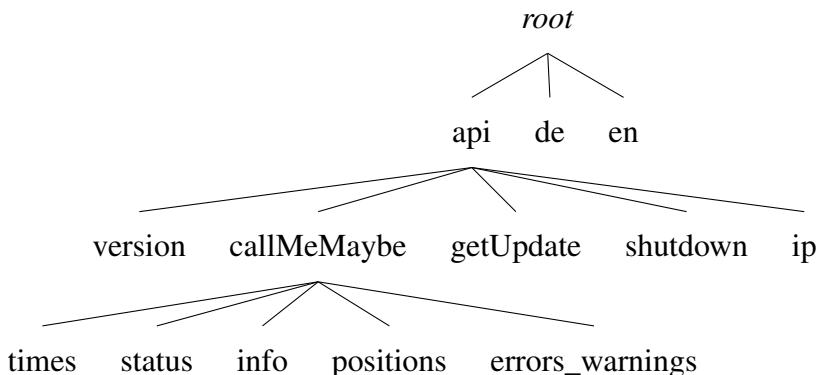
HTTP Request Typ	URI	Server Response	Beispiel
GET	/api/ip	IP	{"ip": "62.116.40.166"}
GET	/api/version	Version	{"version": "beta@1.0.0"}
GET	/api/getUpdate	200 OK	
GET	/api/shutdown	200 OK	
GET	/api/callMeMaybe/times	Zeiten	{"time1": "12:20", "time1_active": true, "time2": "13:20", "time2_active": true, "time3": "14:20", "time3_active": true, "time4": "15:20", "time4_active": true}
GET	/api/callMeMaybe/status	Status	{"nextFeeding": "", "lastFeeding": "", "ausstehend": "nextFeedingIn": "-", "machineState": false}
GET	/api/callMeMaybe/info	Info	{"serialnumber": "117", "internal": "Raspberry Pi 3 Model B", "wlanState": "not connected"}
GET	/api/callMeMaybe/positions	Positionen	{"motor1": "-", "motor2": "-", "sensor1": "-", "sensor2": "-"}
GET	/api/callMeMaybe/errors_warnings	Errors und Warnings	{"Errors": [{"message": "Error", "hidden": false}], "Warnings": [{"message": "Warning", "hidden": false}]}]
POST	/api/putMeHere/times	Zeiten	{"time1": "12:20", "time1_active": true, "time2": "13:20", "time2_active": true, "time3": "14:20", "time3_active": true, "time4": "15:20", "time4_active": true}
POST	/api/putMeHere/changeState	Status	{"state": "true"}
POST	/login	JWT	{"token": "...", "isLoggedIn": true}
POST	/logout	200 OK	

Tabelle 1.1: Übertragungsprotokoll Webserver - Webclient

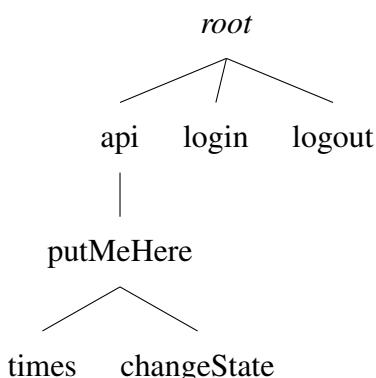
1.4.3 Server

1.4.3.1 Serverpfade

GET



POST



1.4.3.2 Funktion

Da auf einem Port nur ein Programm auf Anfragen warten kann und sichergestellt sein soll, dass der Server nur einmal gestartet wird, wird der Server als Singleton ausgeführt. Das bedeutet, der Konstruktor der Klasse ist nicht öffentlich. Stattdessen gibt es einen Getter, der überprüft, ob ein Objekt der Klasse bereits existiert. Sollte ein Objekt bereits erstellt sein, so wird dieses zurückgegeben. Existiert noch kein Objekt, so wird eines erstellt und danach zurückgegeben. Der Server wird gestartet, indem dem Node.js Modul *http* das Server Objekt übergeben wird:

```
const server = http.createServer(this._express).listen(port)
```

```

1 public static get Instance(): Server {
2   if (Server._instance === undefined) {
3     Server._instance = new Server();
4   }
5   return Server._instance;
6 }
```

Listing 1.17: Server Singleton

Der Server wurde mit dem JavaScript Modul *express* programmiert. Express ist aufgebaut mit sogenannten Middlewares. Diese werden bei Eintreffen einer Anfrage der Reihe nach durchgegangen, bis eine Middleware der Anfrage entspricht. Wenn z.B. die Anfrage `HTTP/1.1 GET /api/version` eintrifft, werden die Middlewares nach der Reihe durchgegangen. Jede Middleware mit `use`, so lange kein Pfad mit angegeben wurde, wird ausgeführt, alle anderen werden übersprungen. Die nächste Middleware, die ausgeführt wird, ist `get('/api/version', ...)`. Wird in dieser Middleware `next()` aufgerufen, so werden alle weiteren Middlewares durchgegangen. Sollte `next()` nicht aufgerufen werden, so werden keine weiteren Middlewares durchgegangen.

```

1  this._express.use(bodyParser.json());
2  this._express.use(bodyParser.urlencoded({ extended: true }));
3  this._express.use(
4    requestLanguage({
5      languages: ['en-GB', 'de-DE', 'de-AT']
6    })
7  );
8  this._express.set('views', path.join(__dirname, '/views'));
9  const pugEngine = this._express.set('view engine', 'pug');
10 pugEngine.locals.pretty = true;
11 this._express.use(this.logger);
12 this._express.use(express.static(path.join(__dirname, '../public')));
13 this._express.use('/assets', express.static(path.join(__dirname, ' ../../
    ngx/dist/assets')));
14 this._express.post('/login', (req, res, next) => this.login(req, res, next
    ));
15 this._express.post('/logout', (req, res, next) => this.logout(req, res,
    next));
16 this._express.use('/api', ApiRoutes.ApiRouter.Routes);
17 this._express.use('/', AppRoutes.AppRouter.Routes);

```

Listing 1.18: Middlewares

Die beiden Middlewares `bodyparser.json()` und `bodyparser.urlencoded()` bereiten den Request so auf, dass er vom Rest der Middlewares verstanden werden kann. Diese Middlewares kommen vom JavaScript Modul *body-parser*. Die Middleware `requestLanguage()` ist ein JavaScript Modul, dass aus dem Request-Header den Header `Accept-Language` ausliest und für die weitere Verarbeitung in `req.language` speichert. Pug Engine ist eine rendering Engine, das bedeutet, damit können HTML Seiten während der Serverlaufzeit generiert werden, somit können Variablen eingefügt werden. `express.static()` stellt ein Verzeichnis mit allen Unterverzeichnissen zur Verfügung. Das gleiche Prinzip wie bei einem Apache Webserver. Die beiden Seiten des Servers, die Client Seite, und die API sind in eigene Dateien ausgelagert. Somit herrscht ein klare Trennung zwischen dem, was der Benutzer zu sehen bekommt und dem, das nur die JavaScript Applikation sieht. Alle *get* und *post* Middlewares haben eine Callback Methode. In dieser bearbeiten wir alle Anfragen und beantworten sie.

```

1  public languageselector(req: express.Request, res: express.Response, next:
    express.NextFunction) {

```

```

2   if (req.language === 'de-DE' || req.language === 'de-AT') {
3     res.redirect('/de');
4     return;
5   } else {
6     res.redirect('/en');
7     return;
8   }
9 }
```

Listing 1.19: Sprachenauswahl

Wenn eine Anfrage auf `root (/)` eintrifft, entscheidet der Server, ausgehend von dem Wert, der in `req.language` steht, in welcher Sprache der Benutzer die Webseite zu sehen bekommt. Ist der Wert entweder `de-De` oder `de-AT`, so wird die deutsche Seite angezeigt, bei allen anderen Werten, wird die englische Seite angezeigt. Dies geschieht durch Weiterleiten des Requests auf entweder `/de` oder `/en`. Die Angular Applikation wird durch ein `use(express.static())` zur Verfügung gestellt.

```

1 public async login(req: express.Request, res: express.Response, next:
                     express.NextFunction) {
2   let User: Login = { token: '', isLoggedIn: false };
3   let done = false;
4   let token = '';
5   const username = req.body.user;
6   const userpass = SHA512(req.body.password).toString();
7   const Users = await FuettrDB.Instance.getUsers();
8   for (let i = 0; i < Users.length; i++) {
9     let name = JSON.parse(JSON.stringify(Users[i])).user_name;
10    let pass = JSON.parse(JSON.stringify(Users[i])).user_password;
11    if (userpass === pass && username === name) {
12      token = await jwt.sign({ user: username }, this._privkey, {
13        expiresIn: '10h', algorithm: 'RS256' });
14    }
15    if (token !== '') {
16      User.token = token;
17      User.isLoggedIn = true;
18      res.send(JSON.stringify(User));
19    } else {
20      res.status(401).send(JSON.stringify(User));
21    }
22 }
```

Listing 1.20: Login Methode

In der Login Methode wird zuerst das im Request befindliche Passwort gehashed. Nur hashen wird im Normalfall als nicht ausreichend betrachtet und weitere Faktoren wie z.B. der Benutzername oder eine gewisse Zeichenkette werden noch mit gehashed. Dann werden alle Benutzer aus der Datenbank ausgelesen und mit dem Benutzer im Request verglichen. Sind Benutzername und

Passworthash gleich, so wird mit der Methode `jwt.sign()` ein JWT erstellt und signiert. Dieser Token wird in ein Objekt gepackt und dem Client als Antwort gesendet. Sollte keiner der Benutzer übereinstimmen, so wird ein leerer Token und der Fehlercode 401 Unauthorized gesendet.

```

1 public update(req: express.Request, res: express.Response, next: express.
    NextFunction) {
2   fs.writeFileSync(path.join(__dirname, '../..../build'), 'true');
3   child.exec('git pull', (error, stdout, stderr) => {
4     if (stdout !== '') {
5       log.info(stdout);
6     }
7     if (error !== null) {
8       log.warn(error);
9     }
10    child.exec('sudo reboot', (error, stdout, stderr) => {
11      if (stdout !== '') {
12        log.info(stdout);
13      }
14      if (error !== null) {
15        log.warn(error);
16      }
17    });
18  });
19 }

```

Listing 1.21: Update Methode

Beim Request auf `/api/getUpdates` soll die Software aktualisiert und das Raspberry neu gestartet werden. Dafür wurde Node.Js' `child_process` Modul verwendet. Dieses Modul gibt uns die Möglichkeit, shells zu starten, in der wir dann Konsolenbefehle ausführen können. Zuerst wird das Repository aktualisiert mit `git pull`. Danach wird das System neugestartet mit `sudo reboot`. Vorher wird in eine Datei, die vom Raspberry Startup Script `rc.local` überprüft wird, `true` geschrieben. Wenn in dieser Datei `true` steht, werden beim Starten des Raspberry alle Komponenten der Software neu gebaut.

1.4.3.3 MongoDB

Alle Daten, die nicht direkt vom Java-Programm benötigt werden, werden in eine Datenbank gespeichert. Über diese Datenbank haben sowohl das Java-Programm als auch der Webserver Zugriff auf die gemeinsamen Daten wie die Fütterungszeiten und Geräteinformationen. Da MongoDB eine Schemenlose Datenbank ist, können wir die Daten einfach dem DBMS übergeben und sie werden gespeichert. Da ein Zugriff auf die Datenbank ebenfalls nur einmal erfolgen sollte, wurde die Klasse ebenfalls als Singleton ausgeführt.

```

1 const url = 'mongodb://localhost:27017/fuettr';
2 try {
3   const dbServer = await MongoClient.connect(url);

```

```

4  const dbFuettr = await dbServer.db('fuettr');
5  const collTimes = await dbFuettr.collection('data_times');
6  const collUsers = await dbFuettr.collection('data_user');
7  const collInfo = await dbFuettr.collection('data_info');
8  const collHardware = await dbFuettr.collection('data.hardware');
9 .
10 .
11 .
12 this._times = collTimes;
13 this._info = collInfo;
14 this._hardware = collHardware;
15 this._users = collUsers;
16 log.info('Database connected.');
17 } catch (err) {
18   throw err;
19 }

```

Listing 1.22: Verbindung mit der Datenbank

Die Verbindung mit dem DBMS geschieht über die Methode `MongoClient.connect()`. Da wir uns dazu entschieden haben, jeden Datensatz in eine eigene Collection zu speichern, müssen anfangs alle Collections den verschiedenen Variablen zugewiesen werden. Beim Verbinden wird außerdem überprüft, ob die Collections leer sind. Ist dies der Fall, so werden Standardwerte in die Collections geschrieben.

1.4.3.4 Kommunikation mit dem Java Programm

Da der Server und das Java-Programm miteinander kommunizieren müssen, war es notwendig, ein Kommunikationsprotokoll zu erstellen. Alle Daten werden im JSON Format ausgetauscht. Das Protokoll sieht wie folgt aus:

HTTP Request Typ	URI	Server Response
PUT	/ChangeMachineState	neuer Maschinenstatus
GET	/errors_warnings	JSON Objekt mit einem Error und einem Warning Array

Tabelle 1.2: Übertragungsprotokoll Webserver - Java

Die Kommunikation mit dem Java-Programm erfolgt über Node.js' eigenes `http` Modul. Das Modul hat Methoden, mit denen Server und Clients erstellt werden können. Die Kommunikation erfolgt über den lokalen Port 666. Auf diesem Port wartet der Server vom Java-Programm auf Anfragen. Der Webserver, der in diesem Falle zum Client wird, sendet einfache HTTP `GET` und `POST` Anfragen an den Java-Server. Dieser antwortet dann mit der gewünschten Resource.

1.4.4 Verbinden mit WLAN im Java Programm

Da das Verbinden mit dem Wireless Local Area Network (WLAN) mit Netzwerktechnik zu tun hat, fiel mir diese Aufgabe ebenfalls zu. Das Verbinden mit dem WLAN auf dem Raspberry ist ein einfacher Eintrag in eine Textdatei, die unter `/etc/wpa_supplicant/wpa_supplicant.conf` zu finden ist. Über den Befehl `sudo iwlist wlan0 scan` können alle WLAN Netzwerke, die in Reichweite sind, angezeigt werden. Dieser Befehl wird beim Initialisieren des Fensters ausgeführt und liefert einen String zurück, welcher in einer Variable gespeichert wird. In diesem String stehen alle verfügbaren Netzwerke. Die Namen der Netzwerke müssen aus dem String extrahiert und in einer Liste gespeichert werden. Diese Liste wird in einem Dropdownmenü angezeigt. Klickt der Benutzer auf einen Eintrag, so muss er noch zusätzlich das Passwort eingeben. Beim Klick auf den Knopf *Verbinden* wird das Netzwerk mit dem Passwort in der *wpa_supplicant.conf* Datei gespeichert. Bei erfolgreicher Verbindung wird in die Datenbank geschrieben, dass das Raspberry mit dem WLAN verbunden ist.

```
1 network={  
2     ssid="testing"  
3     psk="testingPassword"  
4 }
```

Listing 1.23: WLAN Konfigurationsbeispiel

1.5 Raspberry PI

Im Laufe des Arbeitens kam die Idee auf, ein Skript für das Raspberry zu schreiben, damit bei einem neuen, leeren System nicht alles per Hand installiert werden muss. Diese Skript bot auch die Möglichkeit, im Falle eines kommerziellen Gebrauchs der Maschine sehr schnell und sehr viele Geräte mit der Software zu versorgen. Das Skript installiert Node.js, MongoDB, Java, Git, Angular und alle Module, die von der App gebraucht werden. Es lädt das Repository herunter und baut alle Komponenten der Software. Es konfiguriert alles und startet alle Teile der Software. Das ganze Skript ist in der Standard Sprache der Linux Shell geschrieben. Das Skript kann über den Befehl `bash -c "$(curl -sL https://goo.gl/npRzS6)"` ausgeführt werden.

1.6 Verbesserungsmöglichkeiten und Zusammenfassung

1.6.0.1 Verbesserungsmöglichkeiten

Da die Sprache TypeScript erst im Rahmen dieser Diplomarbeit erlernt wurde, sind viele Unreinheiten im Code zu finden. Es wurde sehr viel experimentiert und ausprobiert. Viele Dinge können verbessert werden.

Eine umfangreichere Verwendung der Datenbank wäre besser gewesen. In der Datenbank selbst wäre es besser gewesen, die Benutzer in eine eigene Collection zu tun und alle anderen Daten in einer Collection zusammenzufassen. Außerdem wären andere Namen besser für die Collections gewesen. Ich war dabei allerdings an meinen Kollegen gebunden, der voreilig die Namen und Unterteilungen festlegte, ohne Rücksprache zu halten und im Nachhinein es nicht mehr ändern wollte.

Im Client Design kann die Seite für Mobilgeräte optimiert werden. Statt Bootstrap kann man Google Material verwenden, das ist für Angular optimiert.

Eine größere Auswahl an Sprachen können noch angeboten werden, aber da ich nur Deutsch und Englisch beherrsche, war dies nicht möglich.

Sicherheitstechnisch könnte man den Server als HTTPS Server ausführen. Dazu benötigt man allerdings eine fixe IP und ein Zertifikat.

1.6.0.2 Zusammenfassung

Viele Abschnitte und Funktionen wurden öfters programmiert und sehr oft überarbeitet, da erst im Laufe des Arbeitens die Sprache TypeScript erlernt wurde und viele Facetten dieser Sprache erst bei längerem Arbeiten zum Vorschein treten. Es gab keine großen Komplikationen. Der Terminplan wurde immer eingehalten. Die Zusammenarbeit mit den Kollegen war bis auf oben genanntes Problem meist aufschlussreich und konstruktiv.

KAPITEL 2

Java-Programm

2.1 Anforderungen

2.1.1 Programm

Die Hauptaufgabe des Programms, welches am Raspberry Pi laufen soll, ist es, dem Benutzer eine Möglichkeit zum Steuern der Katzenfütterungsanlage zur Verfügung zu stellen. Weiters soll das Programm die Motoren steuern und die Sensoren in der Anlage auswerten können. Für diese Aufgabe sollen die General Purpose Input/Output (GPIO) Pins am Raspberry verwendet werden.

2.1.2 Design - Benutzerinterface

Der Benutzer soll mit Hilfe eines kleinen Touchdisplays die Möglichkeit haben die Anlage zu steuern. Deswegen muss darauf geachtet werden, dass alle Graphical User Interface (GUI) Fenster sinnvoll per Touch-Gesten verwendbar sind. Das Design der GUI-Fenster soll einfach und übersichtlich gestaltet werden. Auf der Hauptseite, also der Seite die immer zu sehen ist, sollen Informationen dargestellt werden, die dem Benutzer einen schnellen Überblick über den Zustand der Anlage geben. Alle anderen nicht direkt ersichtlichen Funktionen sollen über sinnvoll benannte Menüpunkte erreichbar sein.

2.1.3 Externe Steuerung

Da die Katzenfütterungsanlage die Katze füttern soll, wenn die Familie der Katze auf Urlaub ist, sollte die Anlage auch über das Internet erreichbar sein. Dafür gibt es die Möglichkeit einen Benutzer auf der Anlage anzulegen, mit welchem man anschließend über eine Webseite auf die Anlage zugreifen kann. Weiters soll das Java Programm (als Server) mit der Web-Applikation (als Client) über Sockets kommunizieren, also Daten austauschen können.

2.2 Voruntersuchung

2.2.1 Wieso Java und nicht C?

Zu Beginn musste entschieden werden mit welcher Programmiersprache gearbeitet werden soll. Zur Auswahl standen Java und C, weil diese beiden Sprachen im Unterricht am stärksten behandelt wurden.

Vorteile von C:

- Programme sind ressourcenschonender
- Hardwarenahe Programmierung einfach durchführbar, direkter Zugriff auf die Pins

Nachteile von C:

- C ist nicht Multi-Threading fähig
- Die großen Freiheiten des Programmierers können zu einem unleserlichem Code führen

Vorteile von Java:

- Erstellen einer GUI mit Java Swing ist einfacher (weil es im Unterricht erlernt wurde)
- In Java kann nur sauber programmiert werden (aufgrund der Überprüfung des Compilers)

Nachteile von Java:

- Java-Programme haben eine schlechtere Performance als C-Programme

Nach dem Gegenüberstellen der Vor- und Nachteile wurde Java als Programmiersprache gewählt.

2.2.2 Wieso das Raspberry Pi 3 Model B?

Schon zu Beginn der Diplomarbeit war klar, dass mit einem Raspberry gearbeitet werden soll, weil bereits im Unterricht mit dem Raspberry gearbeitet wurde und weil an das Raspberry ein Display angeschlossen werden kann. Nun musste entschieden werden, welches Model verwendet werden soll.

Technische Daten des Raspberry Pi 3 Model B:

- Rechenleistung (CPU hat 4 Kerne und eine Taktfrequenz von 1,2GHz)
- Anzahl der GPIO-Pins (26)
- WLAN Fähigkeit

Technische Daten des Raspberry Pi 2 Model B:

- Rechenleistung (CPU hat 4 Kerne und eine Taktfrequenz von 0,9GHz)

- Anzahl der GPIO-Pins (26)
- keine WLAN-Fähigkeit

Wir haben das Raspberry Pi 3 Model B, aufgrund der oben angeführten technischen Daten, gewählt. Den größten Einfluss auf die Entscheidung hatte dabei die WLAN-Fähigkeit. Das liegt daran, dass dadurch die Anlage überall im Haus aufgestellt werden kann und keine Ethernetschnittstelle benötigt wird. Beim Raspberry Pi 2 ist es mit einem WLAN USB Device auch möglich eine WLAN-Verbindung herzustellen. Aber es ist uns wichtig, dass das Raspberry sich, ohne der Hilfe eines zusätzlichen Devices, mit dem WLAN verbinden kann. Ein weiterer positiver Aspekt des Raspberry Pi 3 ist die höhere Rechenleistung.

2.2.3 Auswahl eines Touchdisplays

Das Display muss folgende Anforderungen erfüllen:

- Es muss ein Touchscreen-Display sein
- Es muss einfach an das Raspberry anschließbar sein
- Es soll nicht zu klein sein
- Es soll nicht zu teuer sein

Aufgrund dieser Anforderungen wurde das Touchdisplay von Raspberry gewählt.

Folgende technische Daten des Display legten den Grundstein für die Entscheidung:

- Das Anschließen des Displays an das Raspberry erfolgt über ein Displayportkabel und zwei Kabel für die Stromversorgung
- Das Display hat eine Größe von 7 Zoll
- Der Preis beträgt 70€

2.2.4 Wieso pi4j?

Da Java als Programmiersprache gewählt wurde, musste eine Möglichkeit die GPIO-Pins anzusteuern gefunden werden. Weil bei der Recherche außer Pi for Java (pi4j) kaum etwas gefunden wurde, wurde pi4j gewählt. Weiters vorteilhaft ist, dass das Ansteuern der Pins über die Java Software nicht sehr kompliziert ist. Hierfür müssen nur die Bibliotheken von pi4j in das Projekt eingebunden werden.

2.2.5 Wieso Mongodb?

Eine Datenbank wurde gewählt, weil sie, gegenüber dem Speichern der Daten in eine Datei, mehrere Vorteile aufweist.

2.2.5.1 Vorteil gegenüber Daten in Datei speichern

Vorteile einer Datenbank:

- Professionellere Lösung als das Speichern in eine Datei
- Beim Speichern von großen Datenmengen effizienter
- Es kann vom Java Programm und vom Webserver zugegriffen werden, ohne dass es zu einem Fehler kommt, wenn ein DBMS mit Server verwendet wird
- Das Suchen von Daten ist einfacher

2.2.5.2 Vergleich mit anderen Datenbanken

Vergleich von Mongodb mit mySQL:

- Mongodb
 - Mongodb ist schemenlos und open source
 - Zugriffskonzept basiert auf JSON
 - Das Speichern von Daten mit einem stark strukturierten Datenmodell ist einfach
- mySQL
 - mySQL hat ein Datenschema und ist nur teilweise open source (PostgreSQL ist open source)
 - Zugriffskonzept auf mehrere Arten
 - Unflexibel bei Daten mit einem stark strukturierten Datenmodell

Aufgrund dieser oben angeführten Fakten wurde Mongodb als Datenbank gewählt. Ausschlaggebend ist, dass Mongodb open source ist. Weiters ist es für die Web-Applikation leichter Daten zu speichern, da bereits mit JSON formatierten Daten gearbeitet wird.

2.2.6 Kommunikation mit der Web-Applikation

Der Server mit dem die Web-Applikation kommunizieren kann, wird aufgrund der gewählten Programmiersprache, in Java geschrieben. Der Server und die Web-Applikation laufen am gleichen System und kommunizieren über Sockets. Der Server wird im Hintergrund aktiv sein und auf Anfragen der Web-Applikation warten. Je nach Anfrage wird der Server Daten zurück senden oder Methoden im Programm aufrufen. Die Daten, die bei der Kommunikation ausgetauscht werden, sind im JSON Format codiert.

2.3 Umsetzung

Bei der Umsetzung, also beim Schreiben des Programms, wurde wie folgt vorgegangen. Zuerst wurden alle GUI-Fenster per Hand grob entworfen. Anschließend wurden diese in der IDE Netbeans als `javax.swing.JFrame` Form erstellt. Genaueres über die GUI-Fenster folgt unter Punkt 2.3.4. Danach wurden grundlegende Funktionen, die das Programm zu erfüllen hat, implementiert. Weiters wurden noch die Klassen: Mongodb, pi4j, der Server und der ErrorAndWarningHandler als Singleton implementiert.

Nun folgen ausführlichere Beschreibungen über die oben angeführten Klassen.

2.3.1 Mongodb

2.3.1.1 Allgemeines

Mongodb ist ein schemenlose Datenbank. Schemenlos bedeutet, dass die Daten keine besondere Formatierung brauchen um gespeichert zu werden. Zusätzlich wird jedem gespeicherten Datensatz automatisch ein einmaliger Identifikator gegeben. Weiters ist Mongodb open source Software. Das bedeutet, dass der Quelltext öffentlich ist und auch genutzt sowie auch geändert werden darf. Zusätzlich ist Mongodb kostenlos nutzbar.

Bei einer schemenbehafteten Datenbank werden die Daten in einer Tabelle in Reihen und Spalten gegliedert. Um eine solche Datenbank effizient nutzen zu können, wird ein einmaliger Identifikator, der Primary Key, verwendet. Dieser ist erforderlich um Relationen zu realisieren. Eine Relation ist eine Verbindung zwischen zwei verschiedenen Datensätzen in einer Datenbank erstellt werden.

In Mongodb werden die Daten in Collections gespeichert. Die in den Collections gespeicherten Datensätze werden Documents genannt.

Die Datenbank kann in der Konsole mit dem Befehl **mongod** gestartet werden. In unserem konkreten Fall am Raspberry startet die Datenbank beim Starten des Raspberry automatisch. Weiters kann in der Konsole mit dem Befehl **mongo**, sofern die Datenbank gestartet ist, die Mongo-Shell geöffnet werden. In der Shell können alle angelegten Datenbanken verwaltet werden. Unter Verwalten wird das Ändern, Hinzufügen und Löschen von Daten verstanden. Es können auch neue Datenbanken angelegt oder alte Datenbanken gelöscht werden.

Befehle:

- `show dbs` ... anzeigen aller angelegten Datenbanken
- `show collections` ... anzeigen aller Collections in einer Datenbank
- `use` ... verwenden einer Datenbank, falls die Datenbank noch nicht existiert wird sie neu erstellt

(Beispiel: `use <Datenbankname>`)

- `drop()` ... löschen einer Datenbank oder einer Collection

(Beispiel: `<Datenbankname>.drop()` , `<Collectionname>.drop()`)

- `find()` ... suchen nach bestimmten Daten

(Beispiel: `db.data.find()`)

- `count()` ... zählen der Dokumente in einer Collection

(Beispiel: `db.<Collectionname>.count()`)

- `insert()` ... hinzufügen eines Dokumentes

(Beispiel: `db.data.insert({ "time1": "13:30" })`

- `updateOne()` ... updaten von Daten

(Beispiel: `db.data.updateOne(<ObjektID>, <Daten>)`)

- `deleteOne()` ... löschen eines Dokumentes

(Beispiel: `db.data.deleteOne(<ObjektID>)`)

2.3.1.2 Datenbankmanagementsystem

Ein DBMS verwaltet eine oder mehrere Datenbanken. Mehrere Datenbanken werden dabei benötigt, wenn mehrere Anwendungen oder Programme jeweils eine eigene Datenbank brauchen. Ein Beispiel für ein DBMS ist Mongodb.

2.3.1.3 Singleton

Der Datenbankzugriff wurde in einem Singleton implementiert. Dadurch wird nur ein Objekt der Datenbank erzeugt. Das bedeutet, dass nur von diesem Objekt aus auf die Datenbank zugegriffen und nur eine Verbindung geöffnet wird.

Ein weiterer Vorteil ist, dass wenn einmal eine andere Datenbank verwendet werden sollte, nur diese eine Klasse geändert werden muss, weil nur in dieser Klasse der Code für die spezifische Datenbank enthalten ist. Diese Art von Klasse wird Wrapper-Klasse genannt. Das bedeutet, dass diese Klasse den benötigten Code, für den Zugriff auf die Datenbank, umhüllt.

2.3.1.4 Code Beispiele

Da am Raspberry die neueste Version von Mongodb nicht funktioniert, wird die Version 2.14.2 verwendet. Weitere Details zu diesem Thema sind unter Punkt 2.4.1 zu finden.

Aus diesem Grund sind auch die Methoden, die als Beispiele angeführt sind, von der älteren Version.

Als Erstes muss eine Verbindung mit der Datenbank aufgebaut werden. Falls die Datenbank noch nicht existiert, wird sie automatisch erstellt. Dies ist in Java mit folgenden Methoden möglich:

Listing 2.1: Mit Mongodb verbinden

```
MongoClient mongoDB = new MongoClient();
DB database = mongoDB.getDB("<Datenbankname>");
```

Danach muss die Collection, in der gearbeitet werden soll, ausgewählt werden. Falls die Collection noch nicht existiert wird sie automatisch erstellt. Das ist mit der folgenden Methode möglich:

Listing 2.2: Collection auswählen

```
DBCollection coll = database.getCollection("<Collectionname>");
```

Anschließend kann mit der Datenbank gearbeitet werden.

- Die Dokumente in einer Collection zählen:

Listing 2.3: Anzahl der Collections zählen

```
<Collectionname>.count();
```

- Ein Dokument aus der Datenbank lesen:

Listing 2.4: Nach Dokument suchen

```
DBObject document = <Collectionname>.find(<Identifikator>).next();
```

- Ein Dokument zu einer Collection hinzufügen:

Listing 2.5: Ein Dokument hinzufügen

```
<Collectionname>.insert(document);
```

- Ein Dokument in einer Collection updaten:

Listing 2.6: Ein Dokument updaten

```
<Collectionname>.update(<Identifikator>, document);
```

Bevor das Java Programm beendet wird, sollte die Verbindung zur Datenbank wie folgt getrennt werden:

Listing 2.7: Verbindung zur Datenbank trennen

```
mongoDB.close();
```

Ein konkretes Beispiel des Codes aus dem Programm für die Katzenfütterungsanlage sieht so aus:

Listing 2.8: Konkretes Beispiel: Dokument suchen

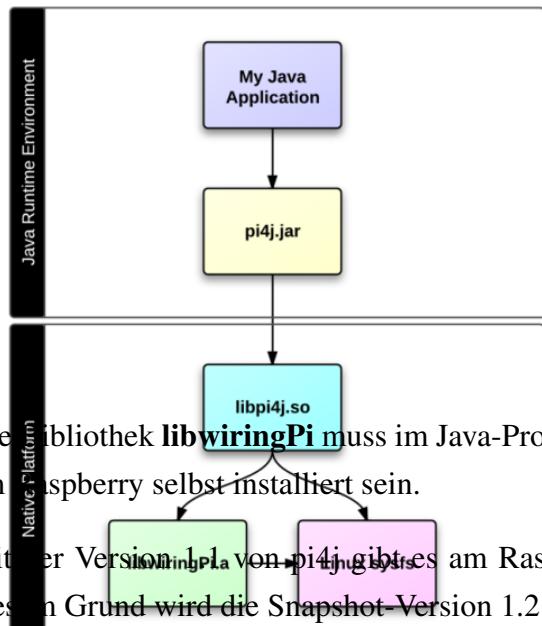
```
DBObject document = collUser.find(
    new BasicDBObject("identifier", "User")).next();
```

Im Code-Beispiel 2.8 wird in der Collection **collUser** nach dem Dokument mit dem Identifikator **new BasicDBObject("identifier", "User")** gesucht. Das gefundene Dokument wird dann dem **DBObject document** zugewiesen.

2.3.2 pi4j

2.3.2.1 Allgemeines

Es wird die open source Software pi4j, welche Bibliotheken zur Verfügung stellt, mit denen es möglich ist, von einem Java Programm, wenn es auf einem Raspberry ausgeführt wird, auf die GPIO-Pins des Raspberries zuzugreifen, verwendet. Dabei kann ein GPIO-Pin als In- oder Output definiert werden. Wenn ein Pin als Output definiert wird, kann er den Zustand High (+3,3V) oder Low (0V) haben. Mit einem Input Pin, können digitale Signale im Bereich von 0V bis +3,3V gemessen werden. Das Ergebnis der Messung ist entweder High oder Low. Weiters ist es auch möglich einem Pin einen Listener zuzuweisen. Dieser Listener wartet bis auf diesem Pin ein Event auftritt und führt dann zum Beispiel eine Methode aus.



Die Software pi4j stellt mithilfe des Java Native Interface (JNI) eine Verbindung von der Java Virtual Machine (JVM) zu dem nativen System des Raspberries her. Dadurch wird es möglich vom Java-Programm auf die Pins zuzugreifen.

Der konkrete Zugriff erfolgt über die Bibliothek **libwiringPi**.

Abbildung 2.1: Abhängigkeiten¹

¹ Abbildung 2.1 Quelle: <http://pi4j.com/dependency.html> (besucht am: 02/03/2018)

2.3.2.2 Pin Numbering Scheme

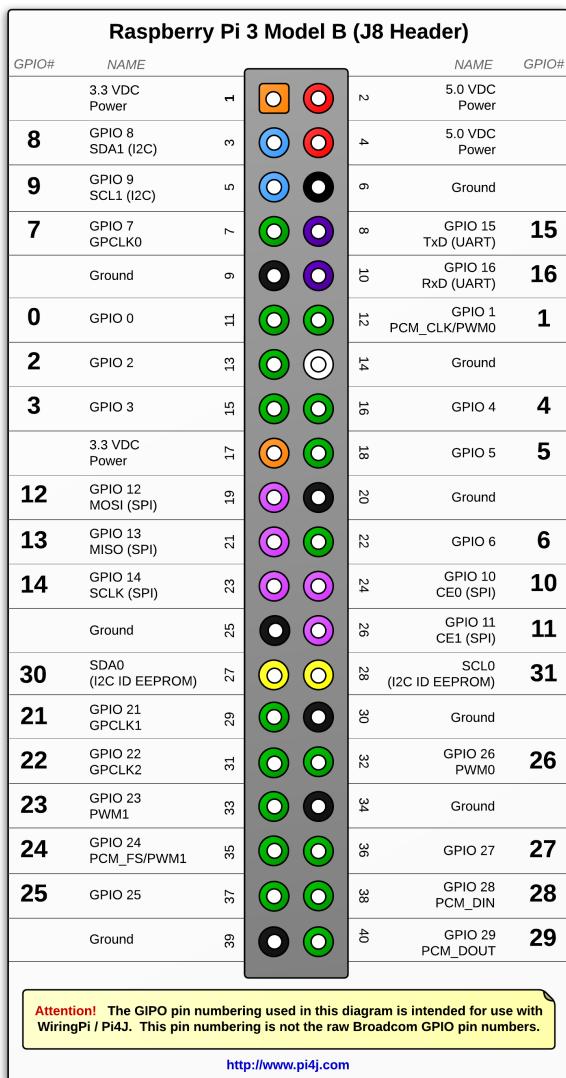


Abbildung 2.2: Pin Numbering Scheme²

2.3.2.3 Gewählte Pin Belegung

Zum Ansteuern der Motoren und Auswerten der Sensoren wurden nur GPIO-Pins verwendet. Die Motoren in der Anlage werden über eine Treiberstufe angesteuert. Für die Sensoren wird jeweils ein Pin zum Auswerten benötigt. In Summe werden acht GPIO-Pins benötigt.

In der Abbildung 2.2 ist zu sehen wie die Pins bei einem Raspberry Pi 3 Model B belegt sind.

Am Raspberry haben einige Pins schon fest zugewiesene Funktionen, aber bei den 26 GPIO-Pins kann der Programmierer selbst entscheiden, welche Funktion der Pin wahrnehmen soll.

Pins auf die mit pi4j nicht zugegriffen werden kann, sind:

- Power mit 3,3V oder 5V
- Ground

Auch unter den GPIO-Pins gibt es Pins mit einer zusätzlichen Funktion. Zum Beispiel können der GPIO-Pin 15 und 16 für eine Universal Asynchronous Receiver Transmitter (UART) Übertragung genutzt werden. Die GPIO-Pins 12, 13 und 14 dienen als Serial Peripheral Interface (SPI) Schnittstelle.

²Abbildung 2.2 Quelle: <http://pi4j.com/pins/model-3b-rev1.html> (besucht am: 02/03/2018)

Diese GPIO-Pins werden, wie in der folgenden Tabelle 2.1 veranschaulicht, verwendet:

Pin	Verwendungszweck
GPIO_00	Sensor Schüsselplatte
GPIO_01	Sensor Förderband (Futtersackerl)
GPIO_02	Motor Schüsselplatte: Aktivieren
GPIO_03	Motor Schüsselplatte: Drehen im Uhrzeigersinn
GPIO_04	Motor Schüsselplatte: Drehen gegen den Uhrzeigersinn
GPIO_06	Motor Förderband: Aktivieren
GPIO_10	Motor Förderband: Drehen im Uhrzeigersinn
GPIO_08	Motor Förderband: Drehen gegen den Uhrzeigersinn

Tabelle 2.1: Belegung der GPIO-Pins

Wenn einer der beiden Sensoren betätigt wird, liefert dieser das Signal High (+3,3V). Wenn der Sensor nicht betätigt ist, liefert dieser Low (0V).

Um einen Motor einzuschalten, müssen jeweils zwei GPIO-Pins auf High geschalten werden. Zuerst muss der Motor aktiviert werden. Wenn der Motor nicht aktiviert ist, ist er ausgeschalten und vom Stromnetz getrennt. Weiters muss dann noch ein weiterer GPIO-Pin auf High geschalten werden. Dieser weitere Pin gibt die Drehrichtung des Motors an.

Tabelle 2.2: Signalverlauf

Motor/Sensor	Signal am Motor/Sensor	Zustand des Motors/Sensors	GPIO-Pin Nummer	Signal am GPIO-Pin
Sensor Schüsselplatte	High	betätigt	GPIO_00	High
	Low	nicht betätigt	GPIO_00	Low
Sensor Förderband	High	betätigt	GPIO_01	High
	Low	nicht betätigt	GPIO_01	Low
Motor Schüsselplatte	Low	steht still, Stromversorgung deaktiviert	GPIO_02 GPIO_03 GPIO_04	Low Low Low
	Low	steht still, Stromversorgung aktiviert	GPIO_02 GPIO_03 GPIO_04	High Low Low
	High	Motor dreht im Uhrzeigersinn	GPIO_02 GPIO_03 GPIO_04	High High Low
	High	Motor dreht gegen Uhrzeigersinn	GPIO_02 GPIO_03 GPIO_04	High Low High
Motor Förderband	Low	steht still, Stromversorgung deaktiviert	GPIO_06 GPIO_10 GPIO_08	Low Low Low
	Low	steht still, Stromversorgung aktiviert	GPIO_06 GPIO_10 GPIO_08	High Low Low
	High	Motor dreht im Uhrzeigersinn	GPIO_06 GPIO_10 GPIO_08	High High Low
	High	Motor dreht gegen Uhrzeigersinn	GPIO_06 GPIO_10 GPIO_08	High Low High

Die GPIO-Pins des Raspberrys werden nicht direkt mit den Motoren und Sensoren verbunden. Die GPIO-Pins werden mit einer Treiberstufe verbunden. Diese Treiberstufe steuert direkt die Motoren und holt die Signale von den Sensoren und gibt sie an den Raspberry weiter.

2.3.2.4 Singleton

Die benötigten Methoden von pi4j wurden in einer Wrapper-Klasse als Singleton implementiert. Es wird ein Singleton verwendet, weil der benötigte Controller für die Pins nur einmal erzeugt werden kann. Wenn der Controller trotzdem öfters erzeugt wird, wird eine Exception geworfen. Der Singleton wird nun dazu verwendet, dass auf die Pins von unterschiedlichen Klassen zugegriffen werden kann.

2.3.2.5 Code Beispiele

Um mit den GPIO-Pins arbeiten zu können, muss zu Beginn ein Controller erstellt werden. Dies ist wie folgt möglich:

Listing 2.9: GPIO-Controller erstellen

```
GpioController controller = com.pi4j.io.gpio.GpioFactory.getInstance();
```

Wenn der Controller erstellt ist, kann auf die Pins, mit denen gearbeitet werden soll, zugegriffen werden. Bei einem Pin kann auch eine **ShutdownOption** angeben werden. Diese gibt an in welchen Zustand der Pin vor dem Herunterfahren gesetzt wird. Dieser Zugriff erfolgt wie folgt:

- Zugreifen auf einen Pin als digitalen Input-Pin:

Listing 2.10: Zugriff auf einen Pin als Input

```
GpioPinDigitalInput pin = controller.provisionDigitalInputPin(
    RaspiPin.GPIO_00, PinPullResistance.PULL_DOWN);
pin.setShutdownOptions(true);
```

- Zugreifen auf einen Pin als digitalen Output-Pin:

Listing 2.11: Zugriff auf einen Pin als Output

```
GpioPinDigitalOutput pin = controller.provisionDigitalOutputPin(
    RaspiPin.GPIO_02, PinState.LOW);
pin.setShutdownOptions(true, PinState.LOW);
```

Wenn das erledigt ist kann mit dem Pin gearbeitet werden. Dies kann wie in den folgenden Beispielen erfolgen:

- Zustand eines Input-Pins auswerten:

Listing 2.12: Pinzustand abfragen

```
pin.getState()
```

Diese Methode liefert den Zustand des Pins zurück. Dieser kann High oder Low sein.

- Zustand eines Output-Pins setzen:

Listing 2.13: Pinzustand verändern

```
pin.low();
pin.high();
```

Mit **pin.low()** kann der Zustand eines Pins auf Low (0V) und mit **pin.high()** auf High (+5V) gesetzt werden.

Vor dem Beenden des Java Programmes sollte der Controller wie folgt heruntergefahren werden:

Listing 2.14: Controller herunterfahren

```
controller.shutdown();
```

2.3.3 Server-Client-Kommunikation

2.3.3.1 Server

Der Server wird beim Starten des Java Programmes mithilfe des SwingWorkers in einem eigenen Hintergrund-Thread gestartet. In diesem Hintergrund-Thread wartet der Server bis ein Client versucht Kontakt mit ihm aufzunehmen. Wenn die Verbindung akzeptiert wird, wird ein neuer Thread geöffnet in dem die Kommunikation mit diesem Client abläuft. Sobald die Kommunikation beendet ist, wird der zuvor geöffnete Thread wieder geschlossen.

Der Server wurde auch als Singleton implementiert, damit von anderen Klassen einfach auf ihn zugegriffen werden kann.

2.3.3.2 Übertragungsprotokoll

Im Übertragungsprotokoll wird festgelegt wie die Kommunikation zwischen Server und Client abläuft. Hier wird festgelegt was eine gültige Request (Anfrage) ist und wie die Response (Antwort) auf den jeweiligen Request aussieht. Weiters wird festgelegt auf welchen Technologien die Kommunikation basiert. Zusätzlich wird noch der Verbindungsauflauf sowie der Abbau beschrieben.

Die Kommunikation zwischen Server und Client basiert bei der Katzenfütterungsanlage auf Sockets. Diese Sockets basieren auf Transmission Control Protocol Internet Protocol (TCP/IP). Dies ist ein verbindungsorientiertes Protokoll. Das bedeutet, dass Pakete automatisch erneut geschickt werden, wenn bei der Übertragung ein Fehler passiert ist oder das Paket verloren gegangen ist. Eine Kommunikation über einen Socket kann wie folgt aussehen. Bei beiden Auflistungen kann der Socket am Ende jeweils noch geschlossen werden.

- Clientseitig

- 1 Socket erstellen
- 2 Datenpaket an eine Adresse verschicken

- Serverseitig
 - 1 Socket erstellen
 - 2 Den Socket eine Adresse geben
 - 3 Auf Datenpakete warten

Als Timeout wurden 5 Sekunden gewählt, weil das sinnvoll zu sein scheint.

Wenn der Request des Clients mit einem **GET** beginnt, bedeutet das, dass der Client Daten vom Server fordert. Beginnt der Request mit einem **PUT**, bedeutet das, dass der Client dem Server Daten schicken will.

Nach jedem **GET** oder **PUT** folgt die URI welche angibt, welche Daten der Client fordert oder welche Daten der Client dem Server schicken will.

Die folgende Tabelle zeigt alle gültigen Requests und die jeweilige Response darauf:

Request		Response
Aktion	URI	
GET	/errors_warnings	Der Server schickt dem Client die aktuell anzugezeigenden Errors und Warnungen in einer Json-Array
PUT	/ChangeMachineState	Der Server ruft eine Methode auf um den Maschinenzustand zu ändern

Tabelle 2.3: Übertragungsprotokoll

2.3.3.3 Response-GET

Wenn der Server eine Request mit dem Beginn **GET** bekommt, muss er dem Client die Daten schicken.

Bei der Katzenfütterungsanlage werden dem Client die ganzen auftretenden Errors und Warnungen, vom Server geschickt. Die Errors und Warnungen werden dem Client als JSON-Object gesendet. Dieses JSON-Object wird in der Klasse, welche alle Errors and Warnungen verarbeitet, erstellt. Das JSON-Object ist unter Punkt 2.3.4.3 dargestellt.

2.3.3.4 Response-PUT

Wenn der Server einen Request mit dem Beginn **PUT** bekommt, muss er Daten vom Client entgegen nehmen.

Bei der Katzenfütterungsanlage bekommt der Server die Daten nicht direkt. Da der Maschinenzustand nur **Ein** oder **Aus** sein kann, wird dieser nicht übermittelt. Stattdessen wird, wenn der Request zum Ändern des Maschinenzustands erhalten wird, die Methode **machineStateChanger()** aufgerufen, welche den Maschinenzustand ändert. Diese Methode aktualisiert auch die GUI-Elemente am Raspberry, abhängig vom Maschinenzustand.

2.3.3.5 Verarbeiten des Requests

Um den Request zu verarbeiten wurde die Klasse **ConnectionThread** geschrieben. Der Request wird wie folgt verarbeitet:

- 1 Der Request wird eingelesen.
- 2 Danach wird überprüft, ob der Request "GET" oder "PUT" enthält. Wenn keine dieser beiden Funktionen enthalten ist, wird eine Fehlermeldung gesendet.
- 3 Wenn dies überprüft wurde und kein Fehler aufgetreten ist, wird der URI überprüft. Über den URI erfährt der Server was er machen soll. Der Inhalt des URI ist ein String und wird im Übertragungsprotokoll festgelegt.

2.3.4 Errors und Warnungsverarbeitung

2.3.4.1 Allgemeines

Der Benutzer der Katzenfütterungsanlage muss über Fehler, die während des Programmablaufs auftreten, informiert werden. Über folgende Fehler wird der Benutzer der Anlage informiert: fehlgeschlagene Fütterungen und Futtermagazin ist leer. Weiters wird der Benutzer über folgende Warnung informiert: es wurde noch kein Benutzer angelegt.

Wenn zu Beispiel das Futtermagazin leer ist und anschließend nachgefüllt wird, wird die nächste Fütterung normal ausgeführt. Die Warnung das kein Benutzer angelegt ist, wird nicht mehr angezeigt sobald einer angelegt wurde.

Dazu dient der **ErrorAndWarningHandler** welcher in einer Wrapper-Klasse als Singleton implementiert wurde. In dieser Klasse ist für jeden möglichen Fehler eine Boolean Variable angelegt. Beim Auftreten eines Fehlers wird die zum Fehler gehörende Boolean Variable auf **true** gesetzt. Beim Erstellen der Liste, die die Errors und Warnungen enthält, werden die jeweiligen Boolean-Variablen abgefragt. Wenn die Boolean-Variable **true** ist, wird der Error oder die Warnung hinzugefügt.

Fehler auf die der Benutzer nicht reagieren kann, also Fehler die im Programm auftreten, werden dem Benutzer nicht angezeigt. Diese Fehler werden mit Hilfe eines Loggigs verarbeitet und anschließend mit Zeitstempel in einer Logging-Datei gespeichert.

2.3.4.2 Errors und Warnungen aktivieren/deaktivieren

Um die Boolean-Variable, die die Errors and Warnungen aktiviert, auf **true** zu setzen, muss die jeweils zum Error oder zur Warnung gehörende Methode aufgerufen werden. Diese Methode kann wie folgt aussehen:

Listing 2.15: Error setzen

```

public void setFeedingHasFailedError (Boolean errorOn)
{
    error_hasFeedingFailed = errorOn;
    if (errorOn == true)
    {
        failedFeedingTime = new SimpleDateFormat ("yyyy.MM.dd HH:mm:ss")
            .format (new java.util.Date ());
    }
}

```

In diesem konkreten Beispiel wird der Error, der dem Benutzer mitteilt, dass eine Fütterung fehlgeschlagen hat, aktiviert. Mit dem Parameter **errorOn** kann bestimmt werden, ob der Error aktiv oder inaktiv sein soll. Wenn der Parameter **true** ist, ist der Error aktiv. Daraus folgt, dass der Error inaktiv ist, wenn **errorOn = false** ist.

In dieser Methode wird zusätzlich ein Zeitstempel erstellt. Dieser Zeitstempel wird nur erstellt, wenn der Error aktiviert wird.

2.3.4.3 JSON-Object

Diese Klasse stellt auch das JSON-Object zur Verfügung, welches dann dem Client (Web-Applikation) geschickt wird. Dieses JSON-Object beinhaltet zwei JSON-Arrays.

Das JSON-Object kann wie folgt aussehen, wenn Errors sowie auch Warnings enthalten sind:

Listing 2.16: JSON-Object mit Errors und Warnings

```

{
    "Errors": [
        {
            "message" : "Error1", "hidden" : false},
        {
            "message" : "Error2", "hidden" : false}
    ]
    "Warnings": [
        {
            "message" : "Warning1", "hidden" : false},
        {
            "message" : "Warning2", "hidden" : false}
    ]
}

```

2.3.5 SwingWorker

Der SwingWorker in Java.Swing wird benötigt, wenn eine Methode mehr Zeit benötigt und somit die GUI blockieren würde. Mithilfe des SwingWorkers, aus der Klasse javax.swing, kann diese Aktion in einem Hintergrund-Thread ausgeführt werden. Der Vorteil davon ist, dass die GUI bedienbar bleibt und nicht blockiert. Dafür wird dann mit Hilfe des SwingWorkers ein Hintergrund-Thread geöffnet in dem die Methode, die mehr Zeit benötigt, abgearbeitet wird.

Das oben geführte Beispiel könnte auch mit der Klasse Thread gelöst werden. Der Hauptgrund für die Verwendung von SwingWorker ist, dass der SwingWorker für die Verwendung in Applikationen mit GUI optimiert ist. Der SwingWorker verfügt über Methoden mit denen nach oder während des Ablaufens des Hintergrundprozesses auf die GUI-Elemente zugegriffen werden kann.

Um den SwingWorker verwenden zu können, muss die Klasse wie folgt erstellt werden:

Listing 2.17: SwingWorker Klasse erstellen

```
private class Worker extends SwingWorker<Object, String> { }
```

Im Generic steht als Erstes der Datentyp des Rückgabewerts von **get()** und als Zweites der Datentyp des Übergabewertes von **publish()**. Diese beiden Methoden sind für die Übermittlung eines Zwischenergebnisses.

2.3.5.1 EDT

Im Java Swing Event Dispatch Thread (EDT) wird der Quellcode sequentiell abgearbeitet. Dies bedeutet, dass alle Methoden ihre benötigte Zeit in Anspruch nehmen und danach die jeweils Nächste ausgeführt wird. Im Grunde wird dann ein SwingWorker verwendet, wenn die GUI für den Benutzer merklich blockiert. In der Regel wird gesagt, dass dann ein SwingWorker verwendet werden soll, wenn die Möglichkeit besteht, dass die GUI für 1 Millisekunden oder mehr blockiert.

2.3.5.2 TimeUnit

Wenn ein Hintergrund-Thread mit einer **while()** Schleife implementiert ist, wiederholen sich die Methoden in der Schleife nach jedem Durchgang wieder. Wenn dies ohne ein Warten passiert, wird sehr viel Prozessorleistung benötigt.

Deswegen wird die Methode `TimeUnit.MILLISECONDS.sleep(500);` verwendet. Mit dieser Methode wartet das Programm für 500 Millisekunden, ohne Rechenleistung zu konsumieren. Dadurch wird der Prozessor nicht so stark ausgelastet.

2.3.5.3 Methoden des SwingWorkers

- `doInBackground()`

Mit **doInBackground()** wird dem SwingWorker mitgeteilt welche Methoden im Hintergrund auszuführen sind. Wenn der Swingworker anschließend mit `.execute()` ausgeführt wird, wird ein neuer Hintergrund-Thread erstellt. Dieser Thread läuft parallel zum EDT. Der Hintergrund-Thread wird geschlossen, wenn mit `.cancel()` ein Flag, welches dem SwingWorker mitteilt, dass er sich demnächst beenden soll, gesetzt wird. Weiters kann sich der Hintergrund-Thread beenden, wenn alle Methoden abgearbeitet wurden oder im Thread eine Exception geworfen wird. In einem Hintergrund-Thread darf nicht auf die GUI-Elemente von Java Swing zugegriffen werden.

Im folgenden Beispiel wird ein Hintergrund-Thread gezeigt, welcher durch das Beenden des SwingWorkers mit `.cancel(true)` beendbar ist.

Listing 2.18: SwingWorker `doInBackground()` mit while Schleife

```
@Override
protected Object doInBackground() throws Exception
{
    while (!isCancelled())
    {
        timeOfDay = String.format("%1$th:%1$tM", new
Date(System.currentTimeMillis()));

        publish( timeOfDay);

        TimeUnit.MILLISECONDS.sleep(500);
    }
}
```

```

    return 1;
}

```

Wenn der SwingWorker beendet wird, während er aufgrund der Methode

`TimeUnit.MILLISECONDS.sleep(500);` wartet, wird eine **InterruptedException** geworfen. Dies ist nur der Fall, wenn der SwingWorker mit `.cancel(true)` beendet wird.

Folgender Hintergrund-Thread arbeitet die Methoden nur einmal ab und wird dann beendet.

Listing 2.19: SwingWorker `doInBackground()` ohne while Schleife

```

@Override
protected String doInBackground() throws Exception
{
    timeOfDay = String.format("%1$tH:%1$tM", new
Date(System.currentTimeMillis()));

    return timeOfDay;
}

```

- **done()**

Wenn ein Hintergrund-Thread beendet wird, wird **done()** im EDT aufgerufen. Im **done()** können dann GUI-Elemente bearbeitet werden. Im Code-Beispiel ?? ist ein Beispiel wie das **done()** aussehen könnte:

Listing 2.20: SwingWorker done()

```

@Override
protected void done()
{
    try
    {
        String time = get();
    }
    catch (Exception ex)
    {
        Logger.getLogger(ex.getMessage()).log(Level.SEVERE, null, ex);
    }
}

```

- **process()**

Wenn in einem Hintergrund-Thread `publish();` aufgerufen wird, wird im EDT **process()** aufgerufen. Hier können wieder GUI-Elemente bearbeitet werden.

Listing 2.21: SwingWorker process()

```

@Override
protected void process(List<String> chunks)

```

```
{  
    String chunk = chunks.get((chunks.size() - 1));  
  
    lbDate.setText(chunk);  
}
```

- `cancel()`

Mit `.cancel(true)` kann ein SwingWorker beendet werden. Das **true** in den Klammern bedeutet, dass ein Thread-Interrupt an den Thread gesendet. Wenn statt **true false** verwendet wird, wird das Flag erst beim nächsten Schleifendurchlauf überprüft.

Dafür muss folgender Befehl aufgerufen werden:

Listing 2.22: SwingWorker abbrechen

```
worker.cancel(true);
```

Bei diesem Methodenaufruf ist **worker** die Objektvariable des SwingWorkers.

2.3.6 GUI-Fenster

Die GUI-Fenster stellen die grafische Oberfläche für den Benutzer zur Verfügung. Da ein Touchscreen-Display verwendet wird, ist es möglich, alle GUI-Fenster durch das Berühren des Touchscreens, mit einem Finger, zu bedienen. Das Bedienen mit mehreren Fingern und die Verwendung von Touch-Gesten sind nicht möglich.

Beim Starten des Raspberrys wird das Programm automatisch gestartet. Dabei wird das Main-Window, also das Hauptfenster des Programms, als Vollbild am Display angezeigt. Somit ist es dem Benutzer nur möglich das Programm zu bedienen und nicht zur grafischen Oberfläche des Raspberrybetriebssystems zu gelangen. Genaueres zum Autostart unter Kapitel ??.

2.3.6.1 Java-Programm im Autostart

Um auf einem Betriebssystem, das auf Linux basiert, ein Java-Programm zu starten, gibt es zwei Möglichkeiten.

1 rc.local

Eine Möglichkeit ist die **rc.local**. Mit der **rc.local** können aber nur Programme ohne grafische Oberfläche gestartet werden. In diese muss folgender Befehl geschrieben werden:

```
java -jar /home/pi/main.jar &
```

2 autostart

Die zweite Möglichkeit ist **autostart**. Hier ist der X-Server, welcher für eine grafische Ausgabe benötigt wird, bereits aktiv. Somit können nun Java-Programme mit einer grafischen Oberfläche gestartet werden. Die Datei **autostart** ist unter folgendem Pfad zu finden :

/home/<user>.config/lxsession/LXDE-<user>/autostart.

In die Datei muss dann noch folgender Befehl eingegeben werden:

@java -jar /home/pi/main.jar &

Das @ am Beginn der Zeile sagt aus, dass das Programm, wenn es nicht ordnungsgemäß geschlossen wird, automatisch neu geöffnet wird.

Bei beiden Möglichkeiten wird jeweils **main.jar** als Programm angegeben. Dies stellt im Fall eines Programms mit GUI das Hauptfenster dar. Bei der Katzenfütterungsanlage ist es das **Main-Window**.

Die Abkürzung **.jar** steht für "Java Archive".

2.3.6.2 MainWindow

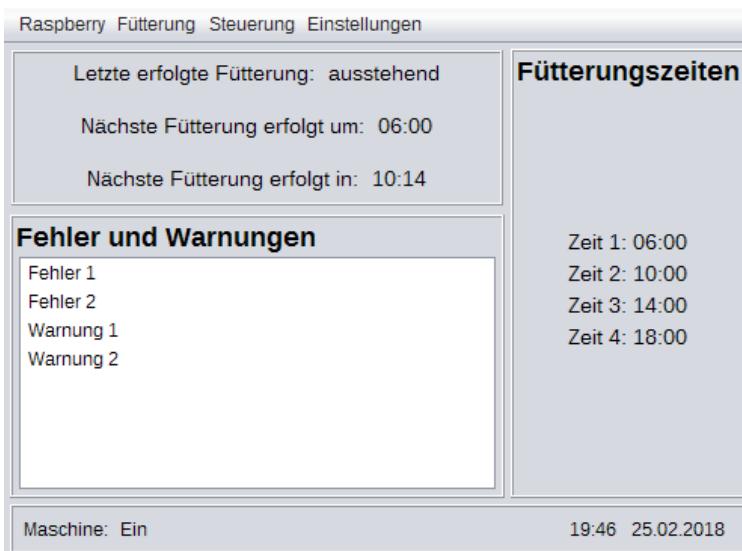


Abbildung 2.3: MainWindow

Das **MainWindow** besteht aus einer **javax.swing.JFrame Form**. In diesem Fenster wird dem Benutzer ein allgemeiner Überblick über die Anlage zur Verfügung gestellt. So mit sind alle wichtigen Informationen schnell und leicht ersichtlich. Weiters ist das **MainWindow** das Hauptfenster des Programms. Über dieses sind alle Dialog-, Steuerungs- und Informationsfenster erreichbar. Diese Fenster werden von Kapitel ?? bis Kapitel 1.7 genauer beschrieben. Das Aufrufen von anderen Fenstern ist über die Menüleiste, welche sich oben in der GUI befindet, möglich.

Unter den Menüs der Menüleiste sind folgende Menüpunkte zu finden:

1 Raspberry

- Neustarten
- Herunterfahren

2 Fütterung

- Einschalten/Ausschalten
- Fütterungszeiten verwalten

3 Steuerung

- manuelle Steuerung
- Positionsinformation

4 Einstellungen

- Update
- Benutzer anlegen
- WLAN
- Geräteinformation

Das **MainWindow** ist als Singleton implementiert, weil dadurch nicht jedes Mal, wenn eine andere Klasse auf ein Datenelement des **MainWindow** zugreift, ein neues Objekt von **MainWindow** erzeugt wird. Somit wird nur ein Objekt des **MainWindow** erstellt. Das Singleton-Objekt wurde wie folgt erstellt:

Listing 2.23: MainWindow createInstance()

```
public static MainWindow createInstance()
{
    if (instance != null)
    {
        throw new RuntimeException("intance already created");
    }

    if (!SwingUtilities.isEventDispatchThread())
    {
        throw new RuntimeException("not in EDT");
    }
    instance = new MainWindow();

    return instance;
}
```

Diese Methode kann nur im EDT aufgerufen werden. Wenn diese Methode nicht im EDT aufgerufen wird oder das Objekt bereits besteht, wird jeweils eine eigene Exception geworfen.

Diese Methode wird in der **main** Methode der Klasse wie folgt aufgerufen:

Listing 2.24: MainWindow createInstance() Aufruf

```
java.awt.EventQueue.invokeLater(new Runnable()
{
    @Override
    public void run()
    {
        MainWindow frame = MainWindow.createInstance();
    }
});
```

Bis jetzt wurde der Singleton nur implementiert. Nun muss noch eine Methode implementiert werden um die **Instance** des Objekts abfragen zu können. Diese Methode sieht wie folgt aus:

Listing 2.25: MainWindow.getInstance()

```
public static MainWindow getInstance()
{
    if (instance == null)
    {
        throw new RuntimeException("instance not created");
    }

    return instance;
}
```

In der GUI werden je nach Maschinenstatus gewisse Funktionen blockiert. Wenn die Fütterung aktiviert ist, sind folgende Funktionen blockiert:

- 1 manuelle Steuerung
- 2 Update

Das Blockieren der Funktionen wird mit folgender Methode gemacht:

Listing 2.26: GUI Elemente blockieren

```
private void updateGUIElements()
{
    // control GUI elements depending on the machine state
    // update and manualControl not available while machine state = on
    if (machineStateOn == true)
    {
        menuUpdate.setEnabled(false);
        menuManualControl.setEnabled(false);
    }
    else
    {
        menuUpdate.setEnabled(true);
        menuManualControl.setEnabled(true);
    }
}
```

2.3.6.3 TimeManagement



Abbildung 2.4: TimeManagement

In der Klasse **TimeManagement** ist es dem Benutzer möglich alle Zeiten zu verwalten. Das GUI-Fenster dieser Klasse ist ein Dialogfenster. Bei Dialogfenstern ist es üblich, dass sie mit **Ok** und **Abbrechen** bedienbar sind.

Es können bis zu vier verschiedene Zeiten gewählt werden. Dabei müssen die Zeiten, bei

Zeit 1 beginnend, aufsteigend angeordnet werden. Zusätzlich können Zeiten auch deaktiviert werden. Um eine Zeit zu deaktivieren, muss das Häkchen vor der Zeit weggenommen werden. Wenn die Zeiten ausgewählt wurden, kann mit **Ok** bestätigt werden. Somit werden die Daten in das Programm übernommen und in der Datenbank gespeichert. Weiters schließt sich noch das Fenster.

Falls **Abbrechen** gedrückt wird, werden die Änderungen nicht übernommen und das Fenster schließt sich.

Wenn das Dialogfenster im EDT aufgerufen wird, blockiert der EDT an dieser Stelle so lange, bis das Dialogfenster wieder geschlossen wird.

Es sind nur vier Fütterungen pro Tag möglich, weil die Katzenfütterungsanlage ein kleines Futtermagazin hat. In diesem Fall würde sich das Magazin bei mehreren Fütterungen sehr schnell leeren. Wenn das nun passiert würde die Katzenfütterungsanlage ihren Zweck nicht mehr erfüllen, weil die Maschine fast täglich nachgefüllt werden müsste.

Da die Fütterungen am Tag begrenzt sind, wurde die Anzahl der Zeiten nicht variabel sondern mit einem Maximum realisiert.

Um in den Spinnern Uhrzeiten anzeigen zu lassen, müssen diese davor konfiguriert werden. Dazu muss in Netbeans, im Designmodus der GUI, ein Rechtsklick auf den Spinner gemacht werden. Weiters muss **Customize Code** ausgewählt werden. Anschließend öffnet sich folgendes Fenster:

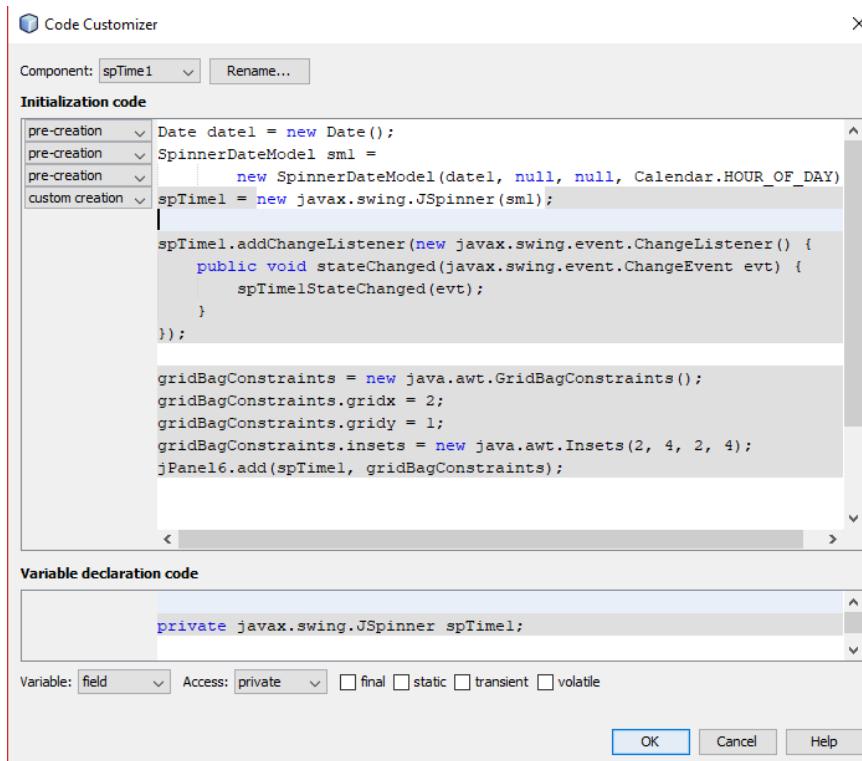


Abbildung 2.5: Spinner Konfiguration

In diesem Fenster ist der Code, welcher in den ersten vier Zeilen steht, hinzuzufügen.

Hier wird das Modell für den Spinner gesetzt. Dieses Model gibt an, dass der Spinner ein Datum darstellen soll. Weiters wird noch festgelegt wie das Datum angezeigt werden soll. Hier wird die Uhrzeit dargestellt.

Diese Schritte müssen für alle der vier verwendeten Spinner angewendet werden.

Zusätzlich muss noch für jeden Spinner folgende Methode im Constructor nach **initComponents()**; aufgerufen werden:

Listing 2.27: Spinner Zeitzone

```
JSpinner.DateEditor at1 = new JSpinner.DateEditor(spTime1, "HH:mm");
spTime1.setEditor(at1);
```

Im **Constructor** der **TimeManagement** Klasse werden, wenn das Fenster geöffnet wird, alle Spinnner mit den aktuellen Zeiten gefüllt. Zusätzlich werden noch die Häkchen, die festlegen ob eine Zeit aktiv ist, gesetzt.

Bevor das Fenster geschlossen wird, wird beim Drücken von **Abbrechen** überprüft, ob sich die Inhalte der Spinner verändert haben. Um dies zu überprüfen werden Events verwendet. Dafür wird für jeden Spinner das folgende Event verwendet:

Listing 2.28: Spinner Event

```
private void spTime1StateChanged(javax.swing.event.ChangeEvent evt)
{
    saved = false;
}
```

Dieses Event wird aufgerufen, wenn der Inhalt der Spinners verwendet wird. Dabei wird dann eine Boolean-Variable, die bekannt gibt, ob etwas verändert wurde, auf **false** gesetzt. Wenn diese Boolean-Variable auf **false** ist, wird vor dem Beenden dem Benutzer eine Warnung ausgegeben, dass noch nicht gespeichert wurde. Siehe Code-Beispiel ?? Zusätzlich kann dann ausgewählt werden, ob wirklich ohne zu speichern beendet werden soll.

Listing 2.29: TimeManagement Fenster schließen

```
private void onCancel(java.awt.event.ActionEvent evt)
{
    if (saved == false)
    {
        if (JOptionPane.showConfirmDialog(this, "Nicht gespeicherte Inhalte  
gehen verloren! Wollen Sie speichern bevor Sie das  
Fenster schließen?", "Hinweis",  
JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
        {
            getValues();

            dispose();
        }
    }
    else
    {
        dispose();
    }
}
```

Wenn das Fenster mit **Ok** geschlossen wird, werden die Inhalte der Spinner, wenn sie geändert wurden, aus den Spinnern geholt. Zusätzlich wird das Gleiche mit den ComboBoxes gemacht. Die Inhalte der Spinner und ComboBoxen werden anschließend in ein Dokument, also ein Format das für die Datenbank verständlich ist, gespeichert. Das Dokument sieht wie folgt aus:

Listing 2.30: Zeitendokument

```
// ... Code ...

newTimeDoc = new BasicDBObject("identifier", "Times")
    .append("time1", time1)
    .append("time1_active", timel_active)
    .append("time2", time2)
    .append("time2_active", time2_active)
    .append("time3", time3)
    .append("time3_active", time3_active)
    .append("time4", time4)
    .append("time4_active", time4_active);
```

Dieses Dokument wird anschließend mit folgender **Getter-Methode** für andere Klassen, konkret für das **MainWindow**, zugänglich gemacht:

Listing 2.31: Zeitendokument Getter-Methode

```
public BasicDBObject getNewTimeDoc()
{
    return newTimeDoc;
}
```

2.3.6.4 CreateAndRegisterUser



Abbildung 2.6: CreateAndRegisterUser

Die Klasse **CreateAndRegisterUser** bietet dem Benutzer die Möglichkeit einen Benutzer anzulegen. Es ist deshalb nur ein Benutzer anlegbar, weil es bei der Anlage keinen Sinn machen würde, wenn mehrere Benutzer angelegt werden. Der angelegte Benutzer wird nur benötigt, um sich auf der Webseite der Anlage anzumelden. Deshalb ist es ausreichend nur einen Benutzer anlegen zu können. Die GUI ist in Abbildung ?? zu sehen.

Die GUI der Klasse ist ein Dialogfenster. Bei Dialogfenstern ist es üblich, dass sie mit **Ok** und **Abbrechen** bedienbar sind.

Es ist nur möglich einen Benutzer anzulegen. Wenn bereits ein Benutzer angelegt ist und dann ein neuer angelegt wird, wird der alte Benutzer überschrieben. Um einen neuen Benutzer anzulegen müssen folgende Felder ausgefüllt werden: Benutzername, Passwort und Passwort wiederholen. Wenn eines dieser Felder leer gelassen wird, wird eine Fehlermeldung ausgegeben. Weiters wird eine Fehlermeldung ausgegeben, wenn die beiden Passwörter nicht übereinstimmen. Wenn alle drei Eingabefelder korrekt ausgefüllt wurden, kann der Benutzer angelegt werden indem **Ok** gedrückt wird. Dadurch werden der Benutzer und das Passwort in das Programm übernommen und in der Datenbank gespeichert. Anschließend erscheint ein Informationsfenster, welches den Benutzer darüber informiert, dass der Benutzer erfolgreich angelegt wurde. Danach schließt sich das Dialogfenster. Wenn anstatt von **Ok Abbrechen** gedrückt wird, schließt sich das Dialogfenster und die Änderungen werden verworfen.

Während das Dialogfenster geöffnet ist blockiert der EDT an der Stelle, an der das Dialogfenster geöffnet wurde.

Um das Passwort einzugeben wird ein **PasswordField** verwendet. Dieses Eingabefeld wird speziell dazu verwendet Passwörter einzugeben. Eine Besonderheit davon ist, dass die Passwörter nicht als Text sondern als Sternchen dargestellt werden. Eine weitere Besonderheit ist, dass der Inhalt eine Zeichenkette ist, also ein Feld von Characters: **char[]**. Weiters unterscheidet sich auch das Abfragen des Inhaltes im Vergleich mit einem normalen Textfeld. Beim **PasswordField** kann der Inhalt mit `passwordField.getPassword()` abgefragt werden.

Um das Passwort im Programm besser verarbeiten zu können wird es wie folgt in einen String umformatiert:

Listing 2.32: char zu String

```
String string_password = valueOf(char_password);
```

Beim Umformatieren des Passworts in einen String könnte ein Fehler auftreten wenn Sonderzeichen verwendet werden. Deshalb sollte verhindert werden, dass der Benutzer Sonderzeichen eingibt oder das Programm angepasst werden, so dass Sonderzeichen nicht verwendet werden können.

Um dem Benutzer Sicherheit zu gewährleisten, wird das Passwort, bevor es in der Datenbank gespeichert wird, wie folgt gehasht:

Listing 2.33: Hash Passwort

```
public static String hash(String passwordToHash, String salt)
{
    String generatedPassword = null;
    try
    {
        MessageDigest md = MessageDigest.getInstance("SHA-512");
        md.update(salt.getBytes("UTF-8"));
        byte[] bytes = md.digest(passwordToHash.getBytes(
            StandardCharsets.UTF_8));
        StringBuilder sb = new StringBuilder();
        for (int i = 0;
             i < bytes.length;
             i++)
        {
            sb.append(Integer.toString(
                (bytes[i] & 0xff) + 0x100, 16).substring(1));
        }
        generatedPassword = sb.toString();
    }
    catch (NoSuchAlgorithmException e)
    {
        e.printStackTrace();
    }
    catch (UnsupportedEncodingException ex)
    {
        Logger.getLogger(HashPassword.class.getName()).log(
            Level.SEVERE, null, ex);
    }
    return generatedPassword;
}
```

Um das Passwort zu hashen wird der Hashalgorithmus **SHA-512** verwendet. Im Programm wird

das Passwort nur einmal gehasht. Falls die Katzenfütterungsanlage in Serie gehen sollte, sollte das Passwort öfters gehasht werden um die Sicherheit zu erhöhen.

Für das Hashen des Passworts wurde auf Stackoverflow³ recherchiert.

Mit dem Parameter **String salt** kann noch ein String übergeben werden, welcher vor dem Haschen des Passworts am Passwort angehängt wird. Somit wird die Entschlüsselung des Passworts nochmals erschwert. Der **salt** wird normalerweise als Byte-Array und nicht als String deklariert.

Beim Drücken von **Ok**, wird, bevor bevor das Fenster geschlossen wird überprüft, ob sich die Inhalte der Text- und Passwortfelder verändert haben. Dabei wird vom Event überwacht, ob in dem Feld ein neues Zeichen eingegeben wird. Danach wird eine Boolean-Variable, die angibt ob gespeichert wurde, auf **false** gesetzt. Das Event für das Textfeld sieht wie folgt aus:

Listing 2.34: Textfeld Event

```
private void tfUserNameKeyPressed(java.awt.event.KeyEvent evt)
{
    saved = false;
}
```

Wenn die Variable **false** ist wird dem Benutzer die Warnung angezeigt, dass die veränderten Daten noch nicht gespeichert wurden.

Wenn das Dialogfenster mit **Ok** bestätigt wird, wird der Benutzername und das Passwort in ein Dokument, welches für die Datenbank verständlich ist, gespeichert. Dieses Dokument sieht wie folgt aus:

Listing 2.35: Benutzerdokument

```
newUserDoc = new BasicDBObject("identifier", "User").append("user_name",
    user_name).append("user_password", hashedPassword);
```

Dieses Dokument wird anschließend noch mit einer **Getter-Methode** für das **MainWindow** zugänglich gemacht:

Listing 2.36: Benutzerdokument Getter-Methode

```
public BasicDBObject getNewUserDoc()
{
    return newUserDoc;
}
```

³A. Sinha. *Stackoverflow*. Okt. 2015.

<https://stackoverflow.com/questions/33085493/hash-a-password-with-sha-512-in-java>.

2.3.6.5 ManualControl



Abbildung 2.7: ManualControl

In der Klasse **ManualControl** wird eine manuelle Steuerung für die Motoren implementiert. Dies erleichtert dem Benutzer das Reinigen der Anlage, weil er zum Beispiel die Drehplatte, mit den Futtergeschüsseln, immer nachdrehen kann. Das gleiche gilt für das Förderband.

Das GUI Fenster ist in Abbildung ?? zu sehen.

Die GUI der Klasse ist ein Steuerfenster. Da dieses Fenster ein Steuer- und kein Dialogfenster ist, werden kein **Ok** und **Abbrechen** benötigt. Aus diesem Grund ist nur ein Knopf **Schließen** vorhanden.

Wie beim Dialogfenster blockiert das Steuerfenster den EDT an der Stelle, wo es geöffnet wurde, bis es wieder geschlossen wird.

Dieses Fenster ist aus Sicherheitsgründen nur dann aufrufbar, wenn der Maschinenzustand auf **Aus** ist. Der Grund dafür ist, dass es zu Fehlern kommen kann, wenn während einer Fütterung ein Motor ein- oder ausgeschalten wird.

Weiters wird der Benutzer bevor er dieses Steuerfenster öffnet darüber informiert, dass er nun direkt die Motoren steuert. Zusätzlich wird der Benutzer gefragt, ob er die Steuerung wirklich öffnen will.

Der Benutzer kann nun beide Motoren unabhängig voneinander steuern. Er kann die Motoren in und gegen den Uhrzeigersinn drehen. Dazu muss er den Knopf mit der jeweiligen Drehrichtung im Steuerungsfenster klicken. Die Motoren sind mit dem jeweiligen Stopp-Knopf wieder stoppbar. Falls die Motoren vom Benutzer vor dem Schließen des Fensters nicht gestoppt werden, werden diese automatisch vom Programm gestoppt. Dies dient dazu, dass sichergestellt ist, dass die Motoren immer gestoppt werden.

Die Methode zum Schließen des Fensters, welche auch die Motoren stoppt, sieht wie folgt aus:

Listing 2.37: Motoren stoppen und Fenster schließen

```
private void onClose(java.awt.event.ActionEvent evt)
{
    // stop engines when closing the control dialog
    // security measurement

    pi4jInstance.stopEngine1();
    pi4jInstance.stopEngine2();

    dispose();
}
```

Neben dem Bereich für die Steuerung befindet sich noch ein Bereich in dem die Echtzeit-Zustände aller Motoren und Sensoren angezeigt werden. So kann der Benutzer, ohne direkt in die Anlage sehen zu müssen, feststellen, ob sich ein Motor dreht oder ein Sensor betätigt ist.

Die Motoren werden gesteuert, indem beim Drücken eines Knopfes, eine Methode aus dem **pi4j Singleton** (Kapitel 2.3.2) aufgerufen wird. Das Singleton-Objekt steuert in weiterer Folge die Output-Pins.

Wenn ein Knopf gedrückt wird, wird die jeweilige zum Knopf gehörende Methode aufgerufen. Dies kann wie folgt aussehen:

Listing 2.38: Motoren drehen

```
private void onMoveEngine1Counterclockwise(
    java.awt.event.ActionEvent evt)
{
    pi4jInstance.moveEngine1Counterclockwise();
}
```

Um die aktuellen Positionen der Motoren und Zustände der Sensoren zu ermitteln und auszugeben wird ein SwingWorker verwendet. Dies ist unter Kapitel ?? genauer beschrieben.

2.3.6.6 Positionsinformation



Abbildung 2.8: Positionsinformation

In der Klasse **Positionsinformation** wird dem Benutzer eine Übersicht über die Zustände der Motoren und Sensoren geboten. So kann er ohne, direkt in die Anlage zu sehen, feststellen, ob sich ein Motor dreht oder ein Sensor betätigt ist.

Die GUI ist in Abbildung ?? zu sehen.

Die GUI der Klasse ist ein Informationsfenster. Da dieses Fenster kein Dialogfenster ist, werden **Ok** und **Abrühen** nicht benötigt. Aus diesem Grund reicht ein Knopf **Schließen** aus.

Wie beim Dialogfenster blockiert das Informationsfenster den EDT an der Stelle, wo es geöffnet wurde, bis es wieder geschlossen wird.

Dieses Fenster ist, im Gegensatz zu **ManualControl**, auch aufrufbar, wenn die Fütterung aktiviert, also der Maschinenzustand auf **Ein**, ist.

Bei der Anzeige für die Drehrichtung der Motoren und die Zustände der Sensoren kann Folgendes angezeigt werden:

1 Motoren

- Motor steht still
- Motor dreht links
- Motor dreht rechts

2 Sensoren

- Unbetätigt
- Betätigt

Die Drehrichtungen und die Zustände werden mithilfe des **pi4j Singleton** (Kapitel 2.3.2) erfasst. In dieser Klasse sind Methoden implementiert, die diese Messungen durchführen können.

Um dies annähernd echtzeitfähig zu realisieren wird ein SwingWorker verwendet. Im **doInBackground()** dieses SwingWorkers, siehe Code-Beispiel ??, werden die Zustände der Motoren und Sensoren abgefragt. Diese Zustände werden mit einem **publish()** an ein **process()** übergeben. Im **process()** werden die Zustände in die Label der GUI geschrieben.

Das **process()** sieht wie folgt aus:

Listing 2.39: Label aktualisieren

```

@Override
protected void process(List<String[]> chunks)
{
    String state[] = chunks.get(chunks.size() - 1);

    lbSensor1.setText(state[0]);
    lbSensor2.setText(state[1]);
    lbEngine1.setText(state[2]);
    lbEngine2.setText(state[3]);
}

```

Das **doInBackground()** ist wie folgt implementiert:

Listing 2.40: Motor- und Sensorzustände

```

@Override
protected Object doInBackground() throws Exception
{
    while (!isCancelled())
    {
        strSensor1 = pi4j_instance.statusSensor1();
        strSensor2 = pi4j_instance.statusSensor2();
        strEngine1 = pi4j_instance.statusEngine1();
        strEngine2 = pi4j_instance.statusEngine2();

        String state[] = null;
        state[0] = strSensor1;
        state[1] = strSensor2;
        state[2] = strEngine1;
        state[3] = strEngine2;

        publish(state);

        TimeUnit.MILLISECONDS.sleep(100);
    }
    return 1;
}

```

In diesem **doInBackground()** werde alle 100 Millisekunden die Motordrehrichtungen und die Sensorzustände abgefragt. Somit ist das nur eine annähernd echtzeitfähige Abfrage der Zustände. Dies wird so realisiert um die Prozessorauslastung zu verringern.

2.3.6.7 SystemInfo



Abbildung 2.9: SystemInfo

In der Klasse **SystemInfo** werden dem Benutzer wichtige Informationen über seine Anlage angezeigt. Das GUI Fenster ist ein Informationsfenster. Das Fenster blockiert den EDT, an der Stelle an der es geöffnet wird, bis es geschlossen wird. Da dies nur ein Informationsfenster ist reicht ein Knopf **Schließen** für die Bedienung aus.

Der Name, des in der Anlage verbauten Rechners, wird von der Datenbank eingelesen. Dies funktioniert indem die aus der Datenbank gelesenen Dateien von einem String in ein JsonObject umgewandelt werden. Anschließend kann aus dem JsonObject der Name des verbauten Rechners ausgelesen werden.

Der WLAN-Status wird, wie der Name des internen Rechners aus der Datenbank eingelesen. Der WLAN-Status wird in die Datenbank geschrieben, wenn sich der Raspberry mit einem WLAN verbindet. Genaueres dazu ist unter Kapitel 1.4.4 zu finden.

Die IP-Adresse der Anlage wird vom Web-Server geholt, welcher diese ermittelt. Dieser Server läuft ebenfalls am Raspberry und ist immer unter einem festgelegtem Port aktiv. Die Internet Protocol (IP)-Adresse wird wie folgt, im **MainWindow**, abgefragt:

Listing 2.41: Abfragen der IP-Adresse

```
URL urlIp = new URL("http://localhost:17325/api/ip");

URLConnection conIp = urlIp.openConnection();

BufferedReader bReaderIp = new BufferedReader(
    new InputStreamReader(conIp.getInputStream()));

final StringBuilder sbIp = new StringBuilder();

while ((line = bReaderIp.readLine()) != null)
{
    sbIp.append(line);
}

ip = sbIp.toString();
```

In diesem Code-Beispiel wird zuerst die Uniform Resource Locator (URL), unter der der Server

zu erreichen ist, festgelegt. Anschließend wird eine Verbindung mit der Server aufgebaut. Wenn die Verbindung besteht wird ein **BufferedReader** und ein **StringBuilder** erstellt. Der **BufferedReader** dient dazu, um die IP-Adresse vom Server zu lesen. Der **StringBuilder** wird benötigt um den String zusammenzufügen, wenn mehrere Zeilen eingelesen werden müssen. Zum Schluss wird mit **sbIp.toString()** der String erstellt, welcher mit einer GETTER-Methode für die Klasse **SystemInfo** zugänglich gemacht wird.

Die Version wird sowie die IP-Adresse von einem Server abgefragt. Das Abfragen wird nach dem gleichen Schema wie bei der IP-Adresse im **MainWindow** gemacht. Der erstellte String wird auch hier mit einer GETTER-Methode für die Klasse **SystemInfo** zugänglich gemacht.

2.3.6.8 Update

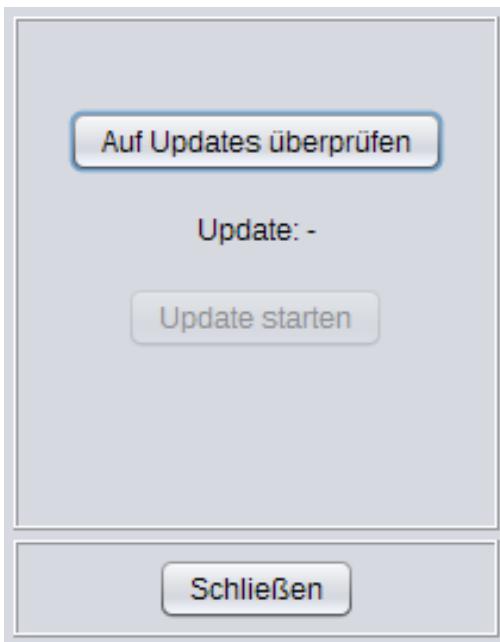


Abbildung 2.10: Update suchen

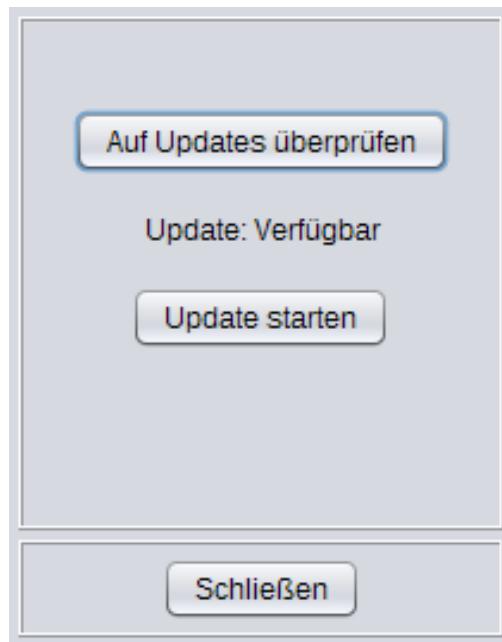


Abbildung 2.11: Update verfügbar

Mit der **Update** Klasse ist es dem Benutzer möglich die Software seiner Anlage aktuell zu halten. Die GUI der Klasse ist auch ein Steuerfenster. Deswegen werden kein **Ok** und **Abbrechen** benötigt und es kann ein Knopf **Schließen** verwendet werden. Während eines Updates ist der **Schließen** Knopf nicht verfügbar.

Wie beim Dialogfenster blockiert das Steuerfenster den EDT an der Stelle, wo es geöffnet wurde, bis es wieder geschlossen wird.

Wenn das Fenster aufgerufen wird, hat es das Aussehen aus der Abbildung 1.7. Nun kann der Benutzer das Fenster schließen oder den Knopf **Auf Updates überprüfen** drücken. Sofern ein Update gefunden wird, wird der Knopf **Update starten** aktiviert und das Label ändert sich von **-** auf **Verfügbar**. Danach sieht das Fenster aus wie in Abbildung ??.

Beim Überprüfen auf Updates wird die lokale Versionsnummer am Raspberry mit der globalen Versionsnummer am Git-Repository verglichen. Wenn die Versionen nicht übereinstimmen ist ein Update verfügbar. Obwohl die Versionsnummer immer chronologisch nach oben gezählt wird, wird auch ein Update gemacht wenn die Versionsnummer niedriger wird. Der Grund dafür ist, wenn eine Version einmal einen schwerwiegenden Fehler beinhaltet, ist es ohne Probleme möglich, auf die vorherige Version zurück zu gehen.

Die Daten für das Update werden von einem Git-Repository heruntergeladen. Dies wird mit einem **git pull** Befehl gemacht. Anschließend wird am Raspberry eine Datei bearbeitet, welche dem Raspberry beim Starten mitteilt, dass die Programme neu zu bauen sind. Nach jedem Update wird das Raspberry automatisch neu gestartet.

Wenn nun der Knopf **Update starten** gedrückt wird, wird folgende Methode aufgerufen:

Listing 2.42: Update

```
private void onUpdate(java.awt.event.ActionEvent evt)
{
    try
    {
        btClose.setEnabled(false);

        Process process = Runtime.getRuntime().exec("git pull");
        process.waitFor();

        write();

        pTextUpdateErfolgreich.setVisible(true);
        btClose.setEnabled(true);

        Runtime.getRuntime().exec("sudo reboot");

    }
    catch (IOException ex)
    {
        Logger.getLogger(Update.class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (InterruptedException ex)
    {
        Logger.getLogger(Update.class.getName()).log(Level.SEVERE, null,
            "WaitFor ist interrupted");
    }
}
```

Listing 2.43: Writer

```
private void write()
{
    String path = String.format(
        "/home/" + System.getProperty("user.name") + "/git/fuettr/build");

    try (final BufferedWriter writer
        = new BufferedWriter(
            new OutputStreamWriter(
                new FileOutputStream(path), "utf8")));
    {
        writer.write(String.format("true"));
    }
    catch (Exception ex)
    {
        ex.printStackTrace();
    }
}
```

Der Pfad, der zum Speicherort der Datei führt, beinhaltet den Benutzernamen des Raspberrys. Deswegen wird die Methode `System.getProperty("user.name")` verwendet, um den aktuellen Benutzernamen herauszufinden.

Wenn das Update, zum Beispiel durch einen Stromausfall, unterbrochen wird, sollte das kein Problem darstellen. Sofern der Stromausfall während dem Downloaden eintritt, wird am Raspberry nichts verändert, weil das Neubauen der Programme erst beim Neustarten des Raspberrys gemacht wird. Wenn nun der Download unterbrochen wird, wird die Datei, die dem Raspberry mitteilt, dass die Programme neu zu bauen sind, nicht bearbeitet. Nach dem Stromausfall kann das Raspberry normal gestartet und somit das Update erneut ausgeführt werden.

2.4 Zusammenfassung - Verbesserungsmöglichkeiten - Probleme

2.4.1 Probleme

2.4.1.1 Problem mit Mongodb am Raspberry

Mit Mongodb gab es ein Problem am Raspberry. Die neueren Versionen werden nämlich nicht unterstützt. Aus diesem Grund musste auf die älteste erhältliche Version 2.14.2 zurückgegriffen werden.

Es gibt auch Möglichkeiten die neuen Versionen am Raspberry verwenden zu können. Diese Lösungen sind im Rahmen einer Diplomarbeit zu zeitintensiv.

Wenn das Projekt weiter geführt werden sollte, sollte überprüft werden, ob die neueste Version von Mongodb mit dem Raspberry kompatibel ist.

2.4.1.2 Probleme mit pi4j

Mit der pi4j Version 1.1 ist das Problem aufgetreten, dass am Raspberry Pi 3 die GPIO-Pins nicht gefunden wurden. Um sicherzugehen, dass der Fehler nicht im Programm liegt, wurde das Code-Beispiel von pi4j verwendet. Auch beim Testen mit diesem Code trat der Fehler, dass die GPIO-Pins nicht gefunden werden können, auf.

Dieses Problem wurde mit dem Verwenden von der pi4j Version 1.2 Snapshot gelöst. Mit der Snapshotversion werden nun alle Pins gefunden

Wenn dieses Projekt weiter fortgeführt wird, sollte überprüft werden, ob von der Snapshotversion auf eine Betaversion gewechselt werden kann.

2.4.2 Verbesserungsmöglichkeiten

2.4.2.1 GUI auf "Touchscreen-Design“ abändern

Die GUI, die momentan bei der Anlage in Verwendung ist, ist nicht optimal um über einen Touchscreen-Display bedient zu werden.

Dabei sollte vor allem die Navigation überarbeitet werden:

Für die Navigation sollte keine Menüleiste verwendet werden. Es wäre besser, wenn Pfeile verwendet werden, um zwischen den Fenstern zu wechseln. Eine andere Möglichkeit wäre, die Fenster über eigene Icons im Hauptfenster aufzurufen.

2.4.2.2 Bessere Benutzerverwaltung - Mehrere Benutzer anlegen

Es könnte eine Möglichkeit geschaffen werden, mehrere Benutzer anzulegen. Dadurch wäre es möglich, Benutzer zu speichern, obwohl momentan ein anderer Benutzer eingeloggt ist. Dazu muss die GUI zum Verwalten der Benutzer auch abgeändert werden. Zuerst muss unterschieden

werden, ob ein Benutzer angelegt werden soll oder ob man sich mit einem Benutzer auf der Anlage anmelden will. Weiters könnte noch eine Übersicht über alle, an der Anlage angelegten Benutzer, hinzugefügt werden.

2.4.2.3 Selbst erstellbare Vorlagen in denen Zeiten gespeichert werden

Die Bedienung könnte dem Benutzer erleichtert werden, wenn er bereits bestehende oder selbst erstellte Vorlagen verwenden könnte. In diesen Vorlagen könnten die Fütterungszeiten und ob diese aktiv sind gespeichert werden. Diese Profile werden zusammen mit dem Benutzernamen auf einem Server abgespeichert. Daraus folgt, dass dem Benutzer, wenn er die Anlage wechselt und er sich erneut anmeldet, wieder alle Vorlagen zur Verfügung stehen. Dann kann er eine Vorlage auswählen und muss nicht alle Zeiten manuell einstellen.

2.4.3 Zusammenfassung

Bei der Umsetzung hat es, abgesehen von den oben angeführten, keine weiteren Probleme gegeben. Der Zeitplan wurde nicht ganz eingehalten. Dabei wurde öfters die Reihenfolge der Arbeitsschritte verändert. Terminlich gibt es keine große Verzögerung.

KAPITEL 3

Mechanik

3.1 Einleitung

Die Grundidee dieses Projektes entstand bei der Überlegung, wie man Haustierbesitzern das Leben erleichtern könne. Man könnte beispielsweise einem vielbeschäftigt Menschen am Morgen Zeit ersparen, indem die Katze durch die Maschine gefüttert wird. Man müsste sich des Weiteren zum Beispiel bei einem Kurzurlaub über das Wochenende keine Gedanken mehr um das Füttern des tierischen Mitbewohners machen. Aufgabe diesen Teiles, war die Umsetzung des mechanischen Aspekts der Arbeit. Des Weiteren galt es den effizientesten Weg des Fütterungsprozesses zu ermitteln.

3.2 Aufgabenstellung und Zielsetzung

Ziel ist es, eine Katzenfütterungsanlage zu entwickeln. Ausgangslage sind Katzenfutterbeutel aus Aluminium-Kunststoff-Folie. Die Fütterung mit Dosen wird hier nicht beachtet. Das Futter soll zeitlich gesteuert in den Behälter bzw. Napf gefüllt und so der Katze zugänglich gemacht werden. Die leeren Beutel werden geruchsisoliert entsorgt. Die Anlage soll zum Beispiel während Kurzurlauben oder zur täglichen Fütterung der Katze (oder des Hundes) Verwendung finden. Nach Aktivierung der Anlage wird in davor gewählten Zeiten die Katze gefüttert.

3.3 Problematik

Jede Variante in den unten beschrieben Maschinen weisen Schwächen bzw. Probleme auf. Diese werden in den folgenden Punkten erläutert.

3.3.1 Problematik des automatisierten Aufschneidens

Das Problem des automatischen Schneiden ist, dass das Material der Verpackung sehr zäh ist und eine hohe Zugfestigkeit besitzt. Deshalb ist ein hoher Anpressdruck zwischen den Klingen erforderlich.

derlich. Zudem darf sich die Klinge, auch wenn sie lang ist, nicht verbiegen, damit die Aluminium-Kunststoff-Folie nicht zwischen die Klinge gelangt und diese auseinander presst. Das hätte Zufolge, dass die die Packung zerknittert und noch schwerer zu schneiden wäre.

3.3.2 Problematik der Dichtheit bei Klemmen

Bei der zweiten Variante wird erwartet, dass der Besitzer die Packung aufschneidet und eine Klemme befestigt. Diese muss mindestens fünf Tage lang dicht halten damit die Maschine nicht verschmutzt und die Katze etwas zu fressen hat. Wenn die Katzenfutterpackung nicht dicht ist gelangt Luft hinein und das gesamte Futter trocknet ein, somit lässt sich das ganze Futter noch schwerer aus der Verpackung pressen. Des weiteren kann das Futter verderben wenn Luft eindringt. Schon bei geringer Veränderung des Futters kann es sein, dass die Katze, das Futter nicht frisst.

3.3.3 Problematik bei Entleerung der Verpackung

Beim Entleeren der Verpackung bei geleeartiger Füllung treten einige Probleme auf die sich meist nur mit dem Pressen der Verpackung lösen lassen. Das Futter kann auf der inneren Verpackungsfolie haften bleiben und somit durch die Schwerkraft nicht vollständig entleert werden. Ebenso muss Luft von der Öffnung bis zur geschlossenen Seite gelangen, damit der Luftdruck nicht das Entleeren verhindert ähnlich wie beim Entleeren einer vollen Ketchupflasche.

3.3.4 Problematik des Geruch

Bei automatischer Fütterung einer Katze ist der Geruch ein großes Problem, da Katzenfutter schon am ersten Tag einen strengen Geruch entwickelt, der sich über Tage steigern kann. Eine Gegenmaßnahme wäre, die ganze Maschine luftdicht zu gestalten. (Eine Ausnahme wäre es, wenn es den Benutzer nicht stört und ihm das Wohl des Tieres am Wichtigsten ist.

3.3.5 Problematik der Reinigung

Durch das Auspressen der Packungen kann das Gelee an der Walze haften bleiben und diese verschmutzen. Die gedachte Lösung ist, dass das Gehäuse aufklappbar ist. Das bedeutet, dass der Benutzer mit viel Freiraum in die Maschine greifen und somit die beiden Walzen reinigen kann. Die Futterschüsseln lassen sich durch die Konstruktion der unten beschriebenen Variante (Drehplatte) leicht entfernen. Die Futterplatte kann, bei Verschmutzung, durch ihre wasserfeste Beschichtung gereinigt werden.

3.3.6 Problematik beim Einfrieren des Futters

Wenn man das Futter einfriert, braucht man durchgehend Energie zum Betreiben der Gefriertruhe. Außerdem wird viel mehr Platz benötigt und der Kompressor macht Lärm. Des weiteren ist das

Dichthalten der Truhe ein großes Problem, da der Greifer an einen bestimmten Punkt in die Maschine eindringen muss, um die gefrorene Portion in den Behälter zu befördern. Wenn die Truhe nicht dicht hält, schmilzt der Inhalt und das Futter verdirbt.

3.4 Konzepte

In den folgenden Punkten werden die verschiedenen Varianten vorgestellt. Des weiteren werden durch Schemenskizzen die einzelnen Varianten verdeutlicht, um so einen Eindruck der zu realisierenden Maschine zu erhalten.

3.4.1 Variante 1: Automatisiertes Aufschneiden

Diese Variante wurde durchdacht um einen Eindruck zu erhalten, wie das automatische Schneiden funktionieren könnte. Welche Probleme es aufwirft und welche Vorteile es gibt, soll analysiert werden.

3.4.1.1 Übersicht der Prozessschritte

In den folgenden aufgelisteten Schritten werden die einzelnen Punkte erläutert und anhand von Fotos der mit Lego gebauten Variante in verschiedenen Ansichten gezeigt.

- 1 Füllen des Futtermagazins
- 2 Führen zur Schneidplatte
- 3 Schnitt
- 4 Pressen
- 5 Entsorgen
- 6 Füttern
- 7 Abtransport der Tasse vor dem nächsten Füttern.

3.4.1.2 Füllen des Futtermagazins

In den folgenden Bildern wird das Magazin anhand eines Aufbaues aus Lego in verschiedenen Blickwinkeln gezeigt. Hier muss man beachten, dass die vom Futterhersteller für die Öffnung vorgesehene Seite, in Richtung des Schneidewerks zeigt (die schmale Seite mit der Einkerbung). Siehe Abbildung: 3.1. Das ganze Förderband besteht grundsätzlich aus der Halterung, die das Magazin in einer bestimmten Höhe hält, damit die einzelnen Trennwände nicht mit dem Boden kollidieren. Der Oberteil des Bandes schließt eben mit der Schneideplattenhöhe ab, um ein leichtes Gleiten der Packung durch den Greifer zu ermöglichen, ohne dass er Höhenunterschiede überwinden muss. Des weiteren wird über zwei Räder ein Band gespannt an denen die Trennwände in Abstand der Dicke der Packung festgemacht werden. Das Band wird mithilfe eines Motors in Bewegung gebracht und kann somit von Abteil zu Abteil bewegt werden, um immer nach dem Füttern eine neue Packung bereit zu stellen. Auf diesem Band kann je nach Länge eine gewisse Anzahl an Futterpackungen abgelegt werden. Natürlich nur auf der Oberseite, da die Packungen ansonsten aus den Abteilungen fallen würden. Die Trennwände sind zum Startzeitpunkt alle auf der Oberseite. In diesem Prototyp hätten 7 Packungen platz.



Abbildung 3.1: Magazin Modellaufbau von Vorne

In Abbildung: 3.2 wird das Magazin von der Seite gezeigt.

In Abbildung: 3.3 wird das Magazin von Oben gezeigt.

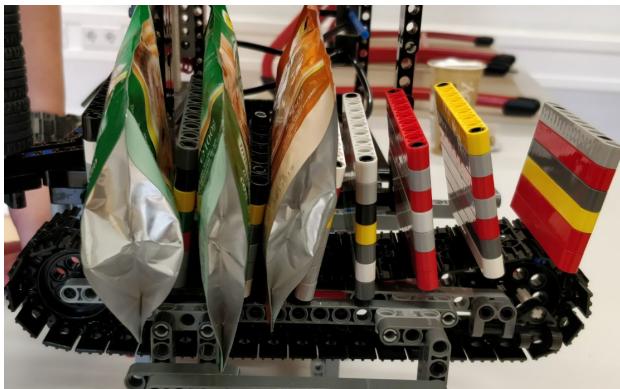


Abbildung 3.2: Magazin Modellaufbau von der Seite

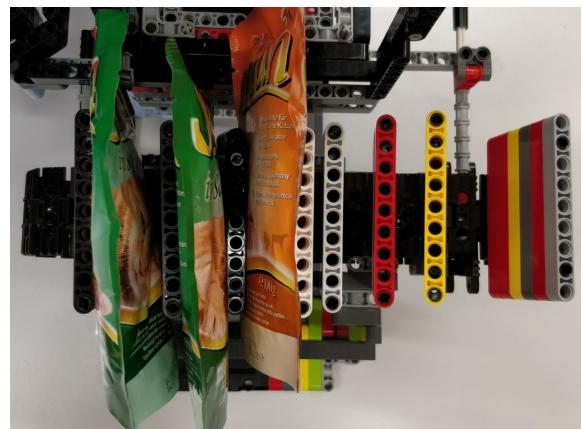


Abbildung 3.3: Magazin Modellaufbau von Oben

3.4.1.3 Führen zur Schneidplatte

In diesem Schritt wird mithilfe eines Greifers, dargestellt durch eine Hand, die Packung in richtiger Position gebracht. Durch die richtige Höhe des Förderbandes muss der Greifer keine hohe Kraft aufwenden, um die Packung an ihrem Zielort zu bringen. Der Greifer dient auch noch dazu, während des Schneidens neben den Magnetzylinern die Packung stabil zu positionieren und zu fixieren, ohne dass der Schnittpunkt verrutscht und die Schneide nicht mehr die Einkerbung, die leichter zu Schneiden ist, trifft. Eine schlechte Fixierung kann zu dem Problem führen, dass man die dickere Kunststoffumhüllung schneidet und der Schnitt nicht ordnungsgemäß durchgeführt wird. Dadurch kann der Kunststoff zwischen die Schneiden gelangen und diese damit auseinander drücken. Dadurch würde dann die Packung nicht aufgeschnitten werden können. Nur bei sehr guter Einspannung tritt beim Schnitt eine Schubbelastung auf, die einen relativ geringen Kraftaufwand für die Rissfortpflanzung erfordert. Die ertragbare Schubspannung muss immer nur punktuell in der Risskerbe überschnitten werden. Bei schlechter Einspannung besteht die Gefahr, dass die Zugspannung über einen größeren Bereich auftreten würde, was enorme Kräfte erfordern würde. Siehe Abbildungen: 3.4, 3.5



Abbildung 3.4: Magazin Auszug



Abbildung 3.5: Magazin Auszug Mitte

Wie im Bild 3.6 gezeigt, liegt die Katzenfutterpackung in der richtigen Position und wird mit zwei Magnetzylinern an der Schneidefläche festgehalten. Die Magnetzyliner haben genügend Kraft, um die Packung auch während des Schnittes und der Walzphase in Position zu halten. Wenn die Packung verrutschen würde, könnte im schlimmsten Fall die Funktion der Maschine beeinflusst werden, indem sie den Greifer oder das Förderband blockiert. Daraufhin müsste die Maschine manuell geöffnet und der Beutel per Hand raus geholt werden.



Abbildung 3.6: Schneidebereit

3.4.1.4 Schnitt

In der richtigen Position muss man mit zwei scharfen Klingen mit viel Kraft die Packung aufschneiden. Eine davon wird an der Schnittfläche angebracht und die andere macht die Schneidbewegung, wobei die beiden aneinander reibenden Kanten den Schnitt verursachen. Die Packung kann mit einem Schnitt vollständig geöffnet werden. Mit zu wenig Druck zwischen den Klingen gelangt zu viel Kunststoffmaterial zwischen die Schneideflächen und durch die Länge der Schneiden biegen sie sich auseinander und somit würde kein ordentlicher Schnitt entstehen (Zugspannung statt örtliche Schubspannung). Bei Mehrmaligen Auftreten dieses beschriebenen Problems bei der selben Packung kann es zufolge haben, dass sich die Packung nicht mehr mit der Maschine schneiden lässt, weil sie sich durch die vielen Versuche verformt hat. Da die Maschine nicht erkennt ob der Schnitt erfolgreich war, würde die Katze bei misslungenem Schnitt nicht gefüttert werden. Siehe Abbildung: 3.7



Abbildung 3.7: Schnitt

Eine Alternative wäre ein feingezähntes Schneiderad mit hoher Drehzahl. Versuche mit einem gezähnten Messer haben gezeigt, dass etwa neun Schnitte mit einem Messer erforderlich waren, um eine Verpackung vollständig aufzuschneiden. Für genauere Informationen siehe 3.5.3

3.4.1.5 Pressen

Nach dem Aufschneiden wird mit einer Rolle die Packung ausgepresst. Dazu werden zuerst die ersten zwei Magnetzyylinder gelöst, bis sich die Rolle vorbei bewegt hat. Danach werden sie wieder in Position gebracht. Daraufhin werden die anderen beiden gelöst und die Rolle fährt ans Ende. Die Rolle ist auf einer Welle platziert. Diese wird mit zwei Sicherungsringen an einer vorgegebenen Position befestigt. Die Rolle ist 10 cm breit, damit ohne Probleme die 9,4cm breite Futterpackung ausgepresst werden kann. Sie wird in einer Vorrichtung an der Maschine angehängt und steht mit einem bestimmten Winkel auf die Schneidfläche damit ein großer Anpressdruck entsteht. Durch die schmierige Konsistenz gleitet das Futter aus der Verpa-



Abbildung 3.8: Ausquetschen Beginn

ckung und wird nicht von der Walze zerquetscht. Nach der Beseitigung der Verpackung werden zuerst die beiden Magnetzyliner von der Maschine entfernt und in Anfangsposition gebracht, damit die Walze ohne Probleme in Startposition zurückkehren kann. Siehe Abbildungen: 3.8, 3.9, 3.10



Abbildung 3.9: Ausquetschen Mitte



Abbildung 3.10: Ausquetschen Ende

Eine Alternative wäre ein senkrechttes Ausquetschen nach unten, unterstützt von der Schwerkraft. Dies ist bei dieser Variante schwer zu realisieren.

3.4.1.6 Entsorgen

Nach dem Auspressen wird die leere Packung durch die Rückklappe in einen luftdichten Container geworfen. Die Klappe wird durch zwei Stifte gehalten und lässt sich durch ein Scharnier nach hinten klappen. Die zwei Stifte sind mit Kosten verbunden, da zwei Magnetzyliner benötigt werden und diese auch, in einem Schaltplan zu berücksichtigen sind. Außerdem benötigen sie zusätzlichen Platz. Die Klappe befindet sich hinter der Futterverpackung. Siehe Abbildung: 3.11.



Abbildung 3.11: Auswurf Beginn

In Abbildung: 3.12 sieht man den Stift (oranger Kreis) der ein vorzeitiges nach Hinten klappen verhindert. Die Stifte müssen so dimensioniert sein, dass sie die Kräfte der Walze aushalten.

In Abbildung: 3.13 wurde der Bolzen entfernt (oranger Kreis) und somit lässt sich die Klappe nach hinten klappen.

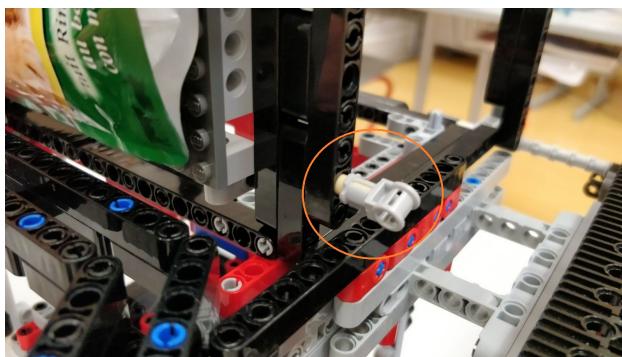


Abbildung 3.12: Bolzen drinnen



Abbildung 3.13: Bolzen entfernen

In Abbildung: 3.14 wird demonstriert wie die Magnetzyliner die leere Packung gegen die Klappe drücken, wodurch die Klappe sich öffnet und die leere Packung hinunterfällt.

In Abbildung: 3.15 sieht man sehr gut, wie die Klappe aussieht und wie sie sich nach hinten öffnet und der Futtersack in der luftdichten Box entsorgt wird.



Abbildung 3.14: Klappe öffnen

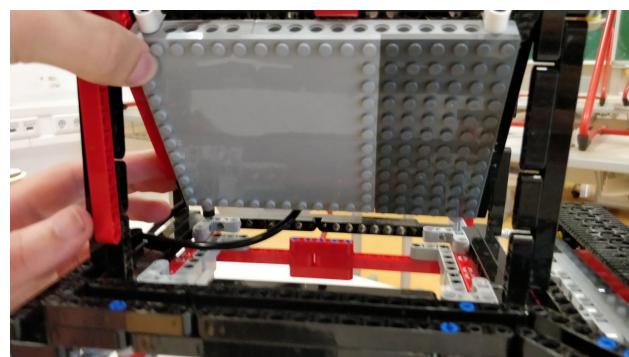


Abbildung 3.15: Fertiger Auswurf

3.4.1.7 Füttern

Die Maschine besitzt fünf Futterschüsseln die auf einer drehbaren Platte stehen. Vor dem Füttern wird eine saubere Platte unter der Stelle, wo später die Packung aufgeschnitten wird, positioniert. Während sie ausgepresst wird, fällt das Futter in die Futterschüssel. Wenn der Auspressvorgang beendet ist, wird die Futterschüssel an eine Position bewegt, wo die Katze Zugang zum Fressen hat. Die Schüsseln lassen sich einfach aus der Halterung nehmen, da sie nur in einem Loch in der Platte liegen. Das hat den Vorteil gegenüber anderer Schüsseln, die auf der Platte montiert sind, dass die Schüssel rasch und eindeutig positioniert werden kann und die Katze nicht soweit zur Futterschüssel hat. Die Platte ist mit einer Gummischicht überzogen damit die Schüssel, durch der Kopf der Katze wenn sie

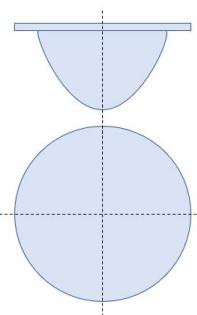


Abbildung 3.16: Einhänge Futterschüssel
Die Zeichnung zeigt einen vertikalen Querschnitt eines Futterschüssels. Oben befindet sich eine halbkreisförmige Aussparung, die in die Platte integriert ist. Unten befindet sich ein Kreis, der die Form der Schüssel darstellt. Beide Teile sind durch einen vertikalen Strich miteinander verbunden, was die Montage am Loch in der Platte ermöglicht.

frisst, nicht verrutscht. Sie lässt sich aus der Platte entnehmen, indem der Benutzer mit der Hand die Futtergeschüsse von unten durch das Loch drückt und mit der anderen Hand entnimmt. Danach werden die Schüsseln gewaschen, getrocknet und danach in das Loch fallen gelassen. Siehe Abbildung: 3.16

3.4.2 Variante 2: Vor aufgeschnittene Packung

Diese Variante wurde entwickelt um den Schneidemechanismus zu umgehen, da das Schneiden kein leichtes Unterfangen ist. Hierbei wird jedoch dem Benutzer zugemutet, mehr Zeit in die Maschine zu investieren als bei den anderen Varianten. Er muss sämtliche Packungen aufschneiden, mit einer Klemme versehen und in das Förderband einhängen. Mit dieser Bauweise wird viel auf die Hilfe der physikalischen Kräfte vertraut. Die Schwerkraft wird maßgeblich genutzt um die Packung zu entleeren.

3.4.2.1 Förderband und Kettenglieder

Für das Förderband wird über die zwei Kettenräder eine Kette gespannt. Auf diese Kette werden die Futterpackungen gehängt, dass funktioniert aber nur weil die Kettenglieder einen rechten Winkel auf jeder Seite haben (siehe Abbildung: 3.18). Auf diesen Winkel wird eine Aluplatte geschraubt und mit einer anderen Platte festgeklemmt. Die Kette wird mithilfe eines Kettenrades und eines Motors in Bewegung gebracht, damit bewegt sich die Packung immer näher Richtung Walze. Da die Packung senkrecht auf das Förderband gehängt wird, spielt die Schwerkraft eine große Rolle und unterstützt den Entleerungsprozess der Katzenfutterpackung.

Die zwei dunkelblauen Kreise symbolisieren die zwei Kettenräder, die die Kette antreiben. Um die Kreise liegen die einzelnen Kettenglieder, verbunden zu einer Kette. Die Pfeile sollen die Futterpackungen darstellen, die am Schaft an der Kette befestigt sind und mit den Öffnungen(Pfeilspitzen) nach unten zeigen. Siehe Abbildung: 3.17.

Die dunklere von den blauen Flächen ist die Oberseite des rechten Winkels mit zwei Löchern zur Befestigung der Aluplatte. Die hellere Fläche ist die Unterseite des Winkels. Die Bolzen des Kettengliedes sind Orange eingezeichnet. Siehe Abbildung: 3.18.

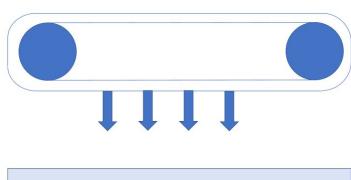


Abbildung 3.17: Foerderband

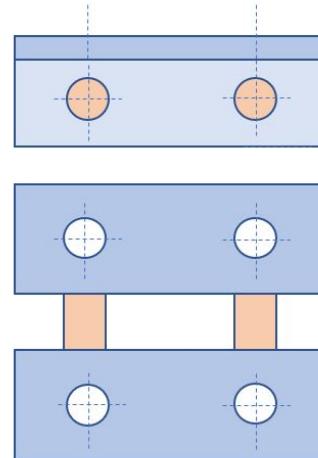


Abbildung 3.18: Kettenglied

3.4.2.2 Walze

Die Walze dient zum Ausquetschen der Futterpackung. Nachdem die Futterpackung in Bewegung ist, wird bei einer gewissen Position die Klemme entfernt und durch die Walze gepresst. Die Walze ist innen hohl und wird auf der Welle platziert. Die erste Walze auf der Antriebsseite und die zweite Walze auf einer eigenen gefertigten Welle. Beide Walzen werden durch eine Feder aneinander gepresst, nur so stark, dass die Halterung, an der die Packung festgemacht ist, durchkommt. Dennoch stark genug, dass sich die Packung entleert. Die Walze an der eigen gefertigten Welle wird mit zwei Aluplatten und einem Scharnier in Stellung gehalten.

In Abbildung: 3.19 ist auf der Walze ein Pfeil gezeichnet. Der Pfeil stellt eine Futterpackung dar und die Pfeilrichtung zeigt die Richtung, in der die Packung ausgepresst wird.

In Abbildung: 3.20 ist der dunkelblaue Teil das eigentliche Scharnier, die hellblauen Flächen sind die Verlängerungen.



Abbildung 3.19: Walze

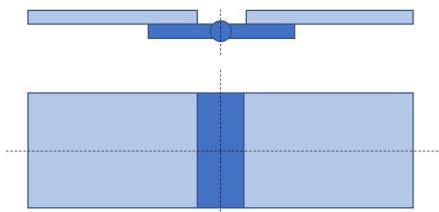


Abbildung 3.20: Scharnier

3.4.2.3 Futterplatte

Nach dem Pressen wird das Futter in die Schüssel gequetscht. Die Platte hat je nach Bedarf eine gewisse Anzahl an Schüsseln, aber maximal Fünf. Diese sind auf einer Platte platziert. Durch den Plattenmittelpunkt geht eine vertikale Welle, die die Platte nach links oder rechts drehen kann. Siehe Abbildung: 3.21

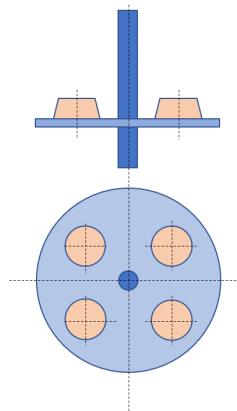


Abbildung 3.21: Futterplatte

3.4.3 Variante 3: Gefrorenes Futter

In dieser Variante wurde überlegt, das Futter einzufrieren, dieses danach aus der Gefriertruhe zu holen und zu erwärmen. Der Vorteil hierbei ist, dass keine Bakterien in das Futter gelangen können da es tiefgefroren ist und es kann die Portionsgröße beliebig gewählt werden. Des weiteren müsste man nicht über das Schneide-Problem nachdenken, da es eine knifflige Angelegenheit ist, die Packung bei jeden Schnitt perfekt zu positionieren und zu schneiden. Der große Nachteil ist der Platzbedarf und der hohe Energieverbrauch der Kühltruhe. Auch die Entnahme des Futters aus der Kühltruhe ist kein leichtes Unterfangen. Erstens kann mit Magnetzylinern gearbeitet werden, zur Verschiebung der Abdeckung. Zweitens könnte ein Loch in die Gefriertruhe geschnitten werden aus dem der Greifer das Futter entnimmt und darum wieder dicht halten muss. Falls es undicht ist, wird es darin zu warm, das Futter schmilzt und verdirbt schlussendlich. Hinzuzufügen ist auch noch, dass Katzen, wenn es um Futter geht, sehr wählerisch sind. Wenn das Futter gefroren ist, hat man zum Einen das Kondenswasser des aufgetauten Futters und zum Anderen schmeckt eingefrorenes Essen anders, als es der Katze vielleicht gewöhnt ist.

3.5 Aufbauten und Tests

In diesem Abschnitt der Diplomarbeit wurden Teile der vorne beschriebenen Varianten aufgebaut und verschiedene Tests durchgeführt, um die Funktionalität der Varianten zu gewährleisten.

3.5.1 Fütterungsexperiment

In diesem Experiment wurde getestet wie lange es dauert bis eine Packung nur mit Hilfe der Schwerkraft ausläuft. Der Beutel wurde nicht extra erwärmt und nur an den beiden unteren Ecken gehalten. Siehe Abbildungen: 3.22, 3.23

In Abbildung: 3.24 wird gezeigt wie viel nach 5 Minuten von der Packung in den Futterbehälter geflossen ist.

In Abbildung: 3.25 sieht man, dass nach 10 Minuten der ganze Inhalt in der Futterschüssel entleert wurde, dennoch tropft es nach.



Abbildung 3.22: Halterung



Abbildung 3.23: Entleerung Anfang



Abbildung 3.24: Entleerung
nach 5min



Abbildung 3.25: Entleerung
nach 10min

3.5.1.1 Schlussfolgerung des Fütterungsexperimentes

Durch die Beobachtung lässt sich durch das Experiment folgende Schlussfolgerungen treffen:

- Durch die leicht Gelee-artige aber eher dünnflüssige Konsistenz des Futters rinnt es kontinuierlich innerhalb von 10 min aus der Packung.
- Da der ganze Inhalt leicht gleitet, kann ohne Probleme durch eine Walze das restliche Futter ausgepresst werden.
- Durch das Eigengewicht wird der Entleerungsprozess erleichtert.

3.5.2 Schneideversuch 1. Art der 1. Variante

Schnitt anhand einer praxisnahen Anwendung dargestellt. Der Beutel wird mithilfe einer Papier-schneidemaschine geschnitten. Siehe Abbildungen: 3.26, 3.27, 3.28



Abbildung 3.26: Einlegen



Abbildung 3.27: Anfangsschnitt



Abbildung 3.28: Endschnitt

3.5.2.1 Schlussfolgerung des Schneideversuchs 1. Art der 1. Variante

Diese ist von den beiden Schneidevarianten die bevorzugte und wurde durch Abwiegen der Pro und Kontras ausgewählt.

In den folgenden Punkten sind die Ergebnisse von der ersten Variante aufgelistet:

- Durch den hohen Anpressdruck der Schneide an der Schneidplatte ließ sich die Packung vollständig öffnen.
- Es gelang beim 1. Versuch mit nur einem durchgehenden Schnitt.
- Die lange Klinge machte auch keine Probleme. Die Packung konnte nicht zwischen die Schneideflächen kommen und somit wurden die Klingen nicht auseinander gedrückt.
- Die vom Futterhersteller vorgegebene Einkerbung kann mit relativ wenig Kraft eine Rissfortpflanzung hervorrufen (genauere Beschreibung unter Variante 1, Führen zur Schneidplatte).
- Ein exaktes Positionieren und Schneiden war möglich.

3.5.3 Schneideversuch 2.Art der 1.Variante

Mit einem Metallwerkzeug mit wellenschliffartiger Kante wird der Futterbeutel entlang der Oberseite aufgeschnitten. Um die Packung vollständig zu öffnen, mussten mehrere Schnitte durchgeführt werden. Siehe Abbildung: 3.29

In Abbildung: 3.30 erkennt man wie offen die Packung nach 3 Schnitten war.

In Abbildung: 3.31 erkennt man wie offen die Packung nach 6 Schnitten war.

In Abbildung: 3.32 wurde die Packung nach 9 Schnitten vollständig geöffnet.



Abbildung 3.29: Schneide-
mittel



Abbildung 3.30: Anfangs-
schnitt
2.Art



Abbildung 3.31: Mittelschnitt 2.Art



Abbildung 3.32: Endeschnitt 2.Art

3.5.3.1 Schlussfolgerung des Schneideversuchs 2.Art der 1.Variante

In den folgenden Punkten werden die Ergebnisse und auch Alternativen der zweiten Variante aufgelistet:

- Falls die Packung nicht genau eingespannt ist, kann durch die zackige Schneide kein guter Schnitt vollbracht werden, da die Klinge nicht schneidet sondern reißt.

- Wenn der Schneidemechanismus auf diese Weise gelöst werden würde, müsste man das Messer durch ein Sägezahnrad ersetzen, das mit einem Elektromotor angetrieben wird.
- Es entsteht durch das Reißen der Verpackung, hervorgerufen durch die gezackte Klinge, kein gerader Schnitt.

3.5.4 Dichtheitsexperiment der Hebelklemme

Die für unsere Maschine entscheidende Klemme wurde 3D gedruckt. Die damit verschlossene Verpackung wurde fünf Tage getestet. Von der Klemme wurde nur die Umrandung der einzelnen Teile gedruckt das bedeutet die Teile sind in der Mitte hohl und dehnbarer bzw. biegsamer als massive Teile. Im Bild: 3.33 erkennt man den ersten Prototypen der Klemme. Die gedruckten Teile werden mit 1,5mm dicke Nägel verbunden. Die Klemme wird im richtigen Modell mit Metall gebaut und die Nägel werden durch passende Stifte ersetzt.

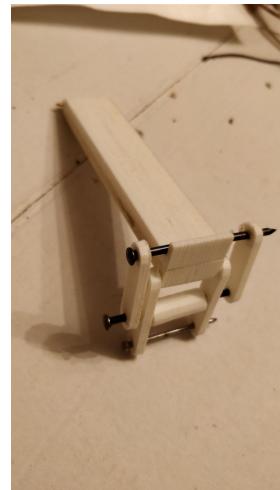


Abbildung 3.33: 3D-Klemme

Die tatsächliche Klemme sollte dann statt aus Kunststoff z.B. aus Aluminium hergestellt werden. Durch den höheren Elastizitätsmodul von Aluminium erhöht sich die Sicherheit der Dichtheit noch weiter.

Die Aufgabe der Klemme war, die Packung fünf Tage senkrecht aufzuhängen und die Klemme auf stärken und schwächen zu Prüfen.

Tag 1: Die Klemme wurde samt der Packung senkrecht aufgehängt und an den von Futterhersteller vorgegebene Schneidepunkt aufgeschnitten. Siehe Abbildung: 3.35. Der erste Eindruck war, dass die Klemme dichthält ohne menschlichen Einfluss. Sobald die Klemme leicht gepresst wurde, ran die Flüssigkeit aus der Klemme, weil diese nicht massiv bzw. stabil genug ist. Die beiden längeren Flächen die über die Packung gelegt wurden, bogen sich durch die menschliche Einwirkung (Erhöhung der Druckes in der Packung) auseinander. Dennoch war die Klemme, wenn sie sich in Ruhelage befand dicht.



Abbildung 3.34: Klemmen Probe



Abbildung 3.35: Klemme
Tag 1

Tag 2: Zur Überprüfung ob die Packung dicht hält wurde eine weiße Unterlage in diesem Fall ein Blatt-Küchenrolle darunter geschoben. Auch hier sieht man durch einen bildlichen Beweis, dass unter der Packung nichts verschmutzt ist. Siehe Abbildung: 3.36.

Tag 3: Das gleiche Resultat wie am Tag 1 und 2. Siehe Abbildung: 3.37.



Abbildung 3.36: Klemme
Tag 2



Abbildung 3.37: Klemme
Tag 3

Tag 4: Wie am Bild 3.38 zu erkennen ist, war am vierten Tag auch keine Flüssigkeit am Tuch. Siehe Abbildung: 3.38.

Tag 5: Der Beweß ist vollbracht. Die ganzen fünf Tage die der Benutzer sie verwendet hätte, hält auch die instabilere Klemme dicht. Siehe Abbildung: 3.39.



Abbildung 3.38: Klemme
Tag 4



Abbildung 3.39: Klemme
Tag 5

3.5.5 Zustand der Futterpackung nach 5 Tagen

Es wurde die Beschaffenheit und die Konsistenz des Futters nach fünf Tagen getestet. Dazu wurde eine Verpackung mit der Hebel-Klemme versehen und hängend in einen Raum platziert. Um eine Sicherheit einzuplanen hing der Futterbeutel 3 Wochen. Nach Ablauf dieser Zeit kamen wir zu folgenden Schlussfolgerungen:

- Aus der Futterpackung ist keine Flüssigkeit ausgetreten. Dies erkennt man daran, dass der Rand nicht verschmutzt bzw. verkrustet und die weiße Unterlage fleckenfrei war. Siehe Abbildung: 3.40.
- Des Weiteren war das Futter weder von Schimmel befallen noch roch es schlecht. Siehe Abbildung: 3.41.
- Zu guter Letzt die Geschmacksprobe. Dies wurde bewiesen, indem die Katze die ganze Schüssel fraß. Siehe Abbildung: 3.42.



Abbildung 3.40: Fütterungsexperiment
Ende



Abbildung 3.41: Futterkonsistenz



Abbildung 3.42: Futter in der Schüssel

3.6 Vergleich der Varianten

In diesem Kapitel der Diplomarbeit werden sämtliche Ideen schematisch dargestellt und erklärt.

3.6.1 Klemmen

Es wurden verschiedene Varianten durchdacht, auf welche Wege die Packung luftdicht verschlossen werden kann. Dabei sind die folgenden Mechanismen entstanden.

3.6.1.1 Einfache Klemme

Die einfache Klemme ist für gewöhnliche Verpackungen gut zu nutzen, jedoch ist sie für unsere Variante nicht zu gebrauchen, weil Kunststoff nicht so stabil wie Metall ist. Drückt die Kunststoffklemme die Packung an manchen Stellen zu wenig zusammen, kann Flüssigkeit austreten. Außerdem hält sie bei Zugbelastung schlechter als Metall. Siehe Abbildung: 3.43

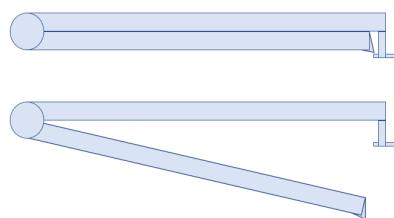


Abbildung 3.43: Einfache Klemme

3.6.1.2 Hebel-Klemme

Die Hebel-Klemme ist für diese Diplomarbeit die bevorzugte Methode, sie kann viel Druck auf die Packung ausüben, sodass keine Flüssigkeit entrinnen kann. Außerdem lässt sich durch den Hebel mit wenig Kraft die Klemme öffnen. Des weiteren können die Klemmen auf einer Stange aufgesammelt werden und liegen nicht an unerwünschten Positionen. Siehe Abbildung: 3.44

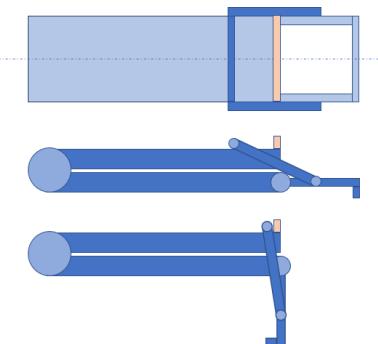


Abbildung 3.44: Hebel Klemme

3.6.1.3 Gummiband-Klemme

Die Gummiband-Klemme hat eine starke Klemmkraft, dies schützt vor dem Aufplatzen der Verpackung. Das Problem dieser Variante ist, dass das Gummiband spröder werden und irgendwann reißen kann, also besteht ein hoher Wartungsaufwand. Die Klemmen können auch nicht nach der Entfernung kontrolliert gesammelt werden. Somit könnten sie an unzugänglichen Stellen fallen.

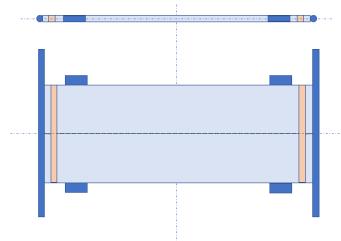


Abbildung 3.45: Gummiband Klemme

3.6.2 Klemmen Wahlvariante

Jede Klemme hat gewisse Vor- und Nachteile. Deshalb wurden alle Varianten miteinander verglichen und die am besten geeignete Klemme ausgewählt.

Die folgenden Punkte zeigen, warum die Hebelklemme für unsere Maschine geeignet ist.

- Durch den Hebel lässt sich die Klemme leicht öffnen, indem sich das Förderband bewegt, die Lasche durch eine Stange eingefädelt wird und diese durch die Bewegung in Richtung Walze geöffnet werden soll.
- Weil die beiden Metallplatten aufeinander pressen, hält die Verpackung durch den besonderen Verschluss luftdicht zusammen (siehe Abbildung: 3.44).

3.6.3 Futterschüsseln

Bei den Futtergeschüsseln mussten bestimmte Faktoren erfüllt bzw. auf wälderische Katzen abgestimmt werden. Deshalb konnte nicht die einfachste Variante genommen werden, die die am wenigsten Aufwand bedeutet hätte.

3.6.3.1 Drehfutterplatte

Die Drehplatte besteht aus fünf Schüsseln. Man kann pro Schüssel die Katze 2-mal pro Tag füttern z.B. morgens und abends. Dadurch hat die Katze jeden Tag eine neue Schüssel und falls sie nicht frisst muss sie nicht Hunger leiden (Da Katzen wälderisch sein können, wenn Futter länger herum steht). Auf einer Welle wird eine Platte befestigt. Darin werden fünf Löcher geschnitten und die Schüssel hinein gelegt. Die Drehplatte wird mit einem Schneckengewinde in die gewünschten Position gebracht. Siehe Abbildung: 3.46

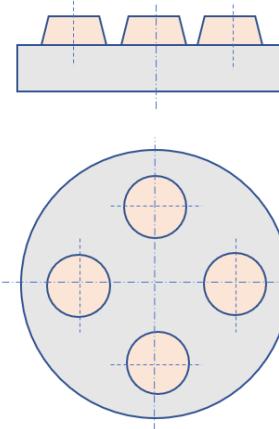


Abbildung 3.46: Drehplatte

3.6.3.2 Futterplatte Zylinder

Die Futterplatte mit Zylinder ist die umständlichste Variante. Es ist eine viereckige Platte auf der Schienen für das Schieben der Futtergeschüsseln platziert sind. Diese werden von Magnetzylinern angeschoben. Der Nachteil hierbei ist, dass viele Bauteile benötigt werden und alle Zylinder müssen aufeinander abgestimmt arbeiten um die Futtergeschüssel zur richtigen Position zu führen. Siehe Abbildung: 3.47

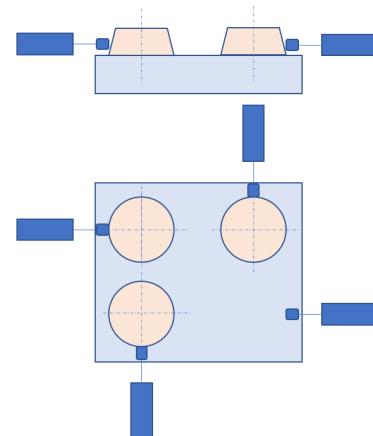


Abbildung 3.47: Platte Zylinder

3.6.3.3 Platte mit einer Schüssel

Die Platte mit nur einer Schüssel ist leicht zu realisieren, da sie nur wenige Bauteile benötigt. Da wäre zum Einen die Platte inklusive Futterschüssel und Schiene und zum Anderen auch die zwei Magnetzyliner, die die Futterschüssel in die Anfangs und Endposition bringen. Ein großer Nachteil weswegen diese Methode nicht in Frage kommt ist folgendes: Wenn die Katze nach dem Füttern nicht frisst dann bleibt der Inhalt in der Schale und trocknet ein oder Ungeziefer kommt hinein. Das hat zur Folge, dass die Schüssel jeden Tag gefüllt wird und eventuell übergeht. Siehe Abbildung: 3.48

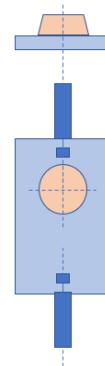


Abbildung 3.48: Einschüsselplatte

3.6.4 Futterschüssel Wahlvariante

Die verschiedenen Futterschüssel-Varianten haben große Vor- bzw. Nachteile bezüglich Platzbedarf und Futterschlüsselanzahl. Diese wurden gründlich durchdacht. Die Wahlvariante ist die Drehplatte, die Gründe dafür werden in folgenden Punkten erörtert:

- Ein großes Thema war die Anzahl der Futterschüsseln. Hierbei war es wichtig, dass die Katze jeden Tag eine saubere Schüssel zur Verfügung hat.
- Die Schüsseln sollten leicht zu entfernen und die Oberfläche der Platte leicht zu reinigen sein.
- Da die Platte auf einer Welle sitzt, lässt sie sich durch den Motor in die Befüll- bzw. Fütterungsposition bringen.

3.6.5 Futtermagazine

Diese Futtermagazine waren für die 1. Variante relevant. Bei diesen war es wichtig die Packungen so einfach wie möglich in die Schneideposition zu bringen.

3.6.5.1 Futtermagazin Horizontal

Das Futtermagazin Horizontal wäre für die erste Variante optimal. Da man den gewünschten Vorrat an Futterpackungen in die abgetrennten Räume platziert. Somit ist es einfach die gewünschte Position anzufahren und die Packungen mit einem Greifer in die Schneideposition

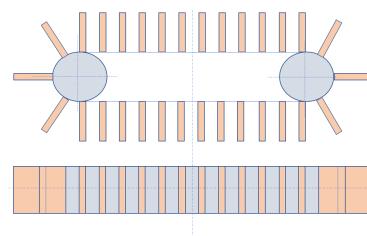


Abbildung 3.49: Futtermagazin Horizontal

zu bringen. Der Aufbau ist wie ein Förderband, nämlich zwei Räder, ein Band mit oben platzierten Trennwänden und ein Motor der dieses Futtermagazin in Bewegung bringt. Zu beachten wäre, wie die Futterpackungen ins Magazin eingelegt werden, nämlich mit der dünneren Fläche mit der Einkerbung, die der Hersteller angegeben hat, in Richtung Futterplatte. Siehe Abbildung: 3.49

3.6.5.2 Futtermagazin Vertikal

Das Futtermagazin Vertikal ist ein rechteckiges Gehäuse an denen 4 Magnetzyliner platziert werden (Die mit Kreise versehenen Blöcke). In diese Box kommen die 10 Futterpackungen. Der Ablauf funktioniert in folgender Reihenfolge. Zuerst öffnet sich der erste linke Magnetzyliner(roter Kreis) danach der gegenüberliegende zweite Magnetzyliner (gelber Kreis). Daraufhin gelangt die erste Futterpackung auf die unteren Zylinder (beide grünen Kreise). Nach diesem Schritt schließen sich die beiden Magnetzyliner wieder, damit die anderen Packungen nach dem Öffnen der unteren Magnetzyliner nicht durch die Maschine fallen. Daraufhin, wenn der Fütterungsbefehl kommt, öffnen sich die unteren Zylinder (beiden grünen Kreise)und die Packung gleitet über ein Blech zur Schnittfläche. Der große Nachteil dieser Methode ist, dass immer wieder Fehler auftreten können. Die Futterpackung kann falsch an der Schneidfläche ankommen bzw. sich an einem bestimmten Ort verkeilen. Siehe Abbildung: 3.50

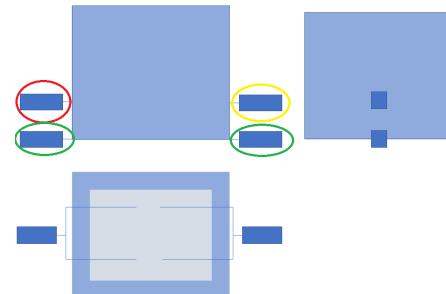


Abbildung 3.50: Futtermagazin Vertikal

3.7 Konstruktion der Wahlvariante und Details

3.7.1 Drehplatte

Die Drehplatte besteht aus Aluminium und ist 10 mm dick. Es werden 5 Löcher für die Schüsseln ausgeschnitten und in der Mitte ist ein Loch, durch das eine Stahlwelle führt. Die Stahlwelle wurde deshalb ausgewählt, da sie erstens stabiler ist und somit kleiner bzw. mit kleinerem Durchmesser gewählt werden kann. Zweitens ist der Vierkant der Motorwelle 4x4x20, dies ist für Aluminium sehr schmal da große Kräfte auftreten können und der Stift abreißen könnte. Die Drehplatte wird auf der Welle mit einem Flansch befestigt damit, das Verrutschen nach oben, unten und zur Seite verhindert wird. Siehe Abbildung: 3.51

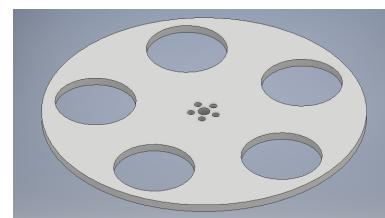


Abbildung 3.51: Drehplatte Inventor

Die Futterschüssel hat deshalb einen Rand damit sie in ein Loch gelegt werden kann ohne dass sie durchfliegt. Dennoch lässt sie sich leicht raus nehmen und reinigen. Durch eine rutschfeste Unterlage verrutscht die Schüssel nicht, auch wenn die Katze daraus frisst und sie mit Kraft die Schüssel zu verschieben versucht. Siehe Abbildung: 3.52.

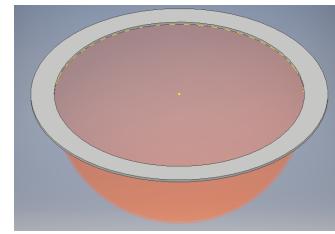


Abbildung 3.52: Futterschüssel Inventor

Die Welle wird aus Stahl konstruiert, damit der Vierkant (grüner Kreis) durch das Drehmoment des Motors nicht abreißt. Die Welle wird senkrecht auf die Grundplatte montiert. Ein Lager wird in den Boden gefräst und die Welle auf dem Lager platziert. Auf der oberen Seite der Welle wird der Motor angesteckt.

Siehe Abbildung: 3.53



Abbildung 3.53: Drehplattenwelle Inventor

3.7.2 Förderband, Kettenglied, Kettenrad und Welle

1

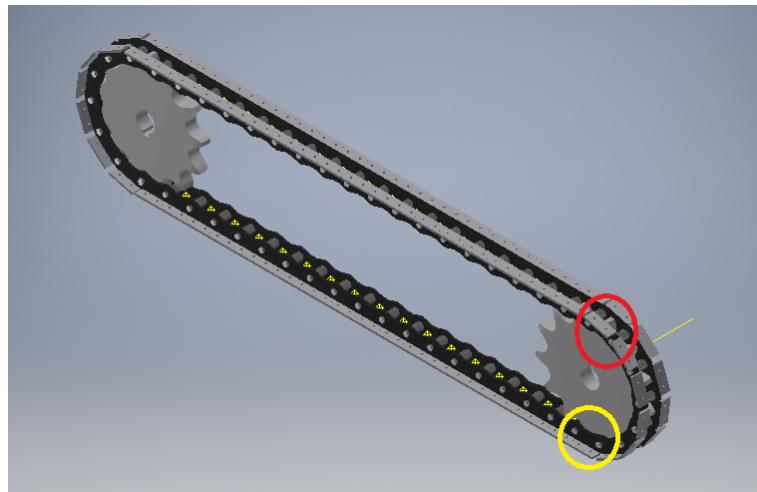


Abbildung 3.54: Kette Inventor

Das Förderband Abbildung: 3.54 weist die z-Achse senkrecht nach oben, die x-Achse nach links und die y-Achse nach rechts. Die Konstruktion wird in den folgenden Punkten beschrieben:

¹Kettenrad.

- 1 Als Erstes wurde die Sicht in Inventor nach Vorne gedreht um die Grundlage des Förderbandes zu zeichnen.
- 2 Zweitens wurde die Ebene YZ sichtbar gemacht und eine Ebene mit Versatz erstellt, die 234,95mm entfernt ist. In diesen beiden Ebenen kommen später die Kettenräder.
- 3 Drittens wird auf der XY Ebene ein Kreis mit einem Durchmesser von 53,068mm gezeichnet. Der zweite Kreis, in der gleiche Größe, wird auf der erstellten Ebene platziert(234,95mm von der ersten Ebene entfernt).
- 4 Viertens werden die beiden Kreise an den obersten und untersten Punkten verbunden. Siehe gelber und roter Kreis.
- 5 Fünftens wird an den beiden untersten Positionen der Kreise ein Punkt platziert. In Inventor existiert eine Methode namens Muster. Diese Funktion wird auf den Punkten platziert und die Richtung des Pfeiles nach rechts gedreht. Nun wird die Anzahl der Kettenglieder angegeben, in unseren Fall 50. Die werden gleichmäßig über die Skizze verteilt.
- 6 Sechstens wird eine neue Datei erstellt und das Kettenrad aus dem Inhaltscenter platziert. Siehe Abbildung 3.56.
- 7 Siebtens wird das Kettenglied konstruiert und alle 3 Komponenten in ein gemeinsames Projekt eingefügt. Siehe Abbildung: 3.55.
- 8 Achtens wird das Kettenglied am untersten Punkt des rechten Kreises abhängig gemacht.
- 9 Neuntens kann durch die Funktion "Muster" das Kettenglied gleichmäßig auf der Zeichnung verteilt werden und dadurch entsteht die fertige Kette.
- 10 Zehntens werden die Kettenräder auf den zuvor erstellten Ebenen abhängig gemacht. Danach ist die Kette fertig erstellt.

Das Kettenglied ist im Handel erhältlich. Die Anfertigung hat auf der Seite einen rechten Winkel auf denen Aluminiumplatten platziert sind. Auf diese Aluminiumplatten wird der Futterbeutel platziert. Siehe Abbildung: 3.55.

Das Kettenrad wurde aus dem Inhaltscenter platziert. Danach wurde eine Skizze angelegt und das Loch in der die Welle durch führen soll gezeichnet. Die Rundung für die Passfeder wurde darauf hin auch konstruiert, damit das Kettenrad nicht auf der Welle durchrutscht und das Drehmoment übertragen kann. Siehe Abbildung: 3.56.

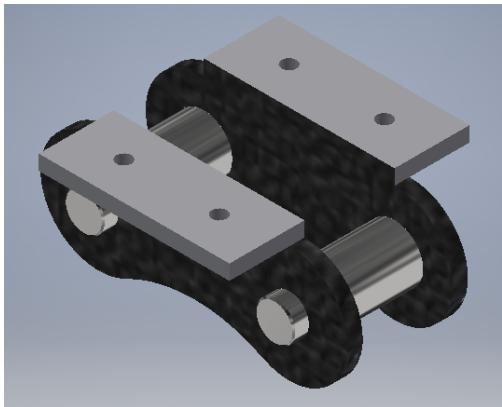


Abbildung 3.55: Kettenglied Inventor

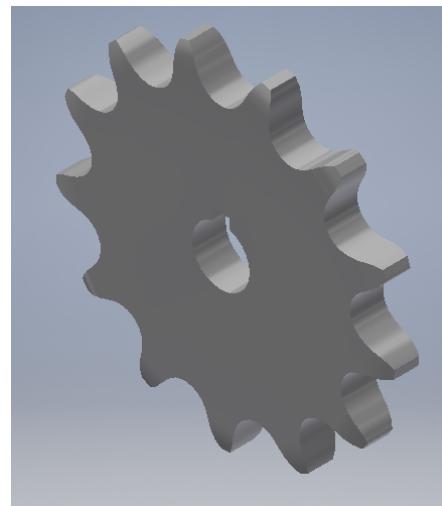


Abbildung 3.56: Kettenrad Inventor

Die Antriebswelle wurde aus Stahl konstruiert um den Vierkant(grüner Kreis) nicht während des Benutzens der Förderbandwelle durch den Motor abzureißen. Die Welle wird an den beiden Stützen gelagert. Siehe Abbildung: 3.57.



Abbildung 3.57: Förderbandwelle Inventor

3.7.3 Walze

Die beiden Walzen sind aus Kunststoff und innen hohl. Diese sind auch nicht auf der Welle befestigt, sondern können sich frei drehen. Es ist auf jeder Seite ein Sicherungsring, damit die beiden Walzen ihre Position axial nicht verlassen. Siehe Abbildung: 3.58.

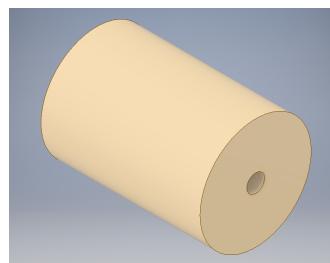


Abbildung 3.58: Walze Inventor

Die Feder drückt die beiden Walzen aneinander um die Futterpackung die durchgezogen wird auszuquetschen. Ohne die Feder an der Walze könnten zu viele Reste in der Packung bleiben und so das ganze Futter verschwendet werden. Die Feder wurde von uns konstruiert und auf eine eigene Konstruktion gehängt, unabhängig von der Welle, damit die Feder nicht mitgedreht wird, sondern fix an einem Platz positioniert ist, an den sie ihre Arbeit verrichtet. Siehe Abbildung: 3.59.

Auf die Aluminiumwelle wird die frei-händige Walze befestigt. Die Welle wird mit zwei Sicherungsringen gesichert, damit sie in Position bleibt. Siehe Abbildung: 3.60.

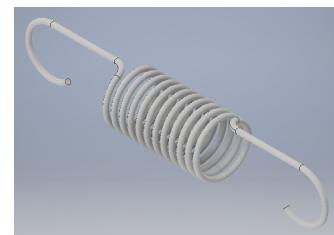


Abbildung 3.59: Feder Inventor



Abbildung 3.60: Welle für die Walze Inventor

3.7.4 Hebelklemme

Die Hebelklemme dient zum Optimalen klemmen der Verpackung. Es ähnelt einen Marmeladenglasverschluss von früher. Die Packung wird zwischen den zwei Platte eingelegt, danach wird der Schließmechanismus auf der oberen Platten eingehakt (siehe roter Kreis). Daraufhin wird die Unterseite des Schließmechanismus nach unten gedrückt (siehe blauer Kreis). Der Hebel wird soweit nach unten gedrückt bis er den Stopper berührt (siehe gelber Kreis). Der Stopper hat die Aufgabe, dass der Hebel nicht zu weit geschlossen werden kann. Das hat zur Folge, dass die Klemme leichter aufspringt. Siehe Abbildung: 3.61.

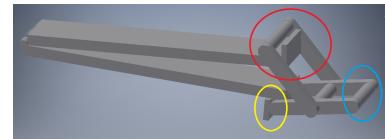


Abbildung 3.61: Hebel Klemme

3.7.5 Gesamtansicht der Konstruktion

Bei diesem Abschnitt werden von der Gesamtkonstruktion einzelne Teile, welche Probleme verursachen könnten, heraus genommen, gezeigt und beschrieben.

3.7.5.1 Ansicht des Pressvorganges und Positionierung der Walze

Es wird beachtet, dass der Spalt der beiden Walzen ungefähr in der Mitte der Futterschüssel ist, wenn man die Maschine von oben betrachtet. Wäre das nicht der Fall, würde das Futter daneben

fallen und am Boden der Maschine landen. Siehe Abbildung: 3.62.

Die Walze, die nur durch die Federn an die Maschine gedrückt wird, muss zusätzlich am Grundgerüst gelagert werden, damit sie auf gleicher Höhe wie die an der Maschine befestigten Walze hängt. Siehe Abbildung: 3.63.

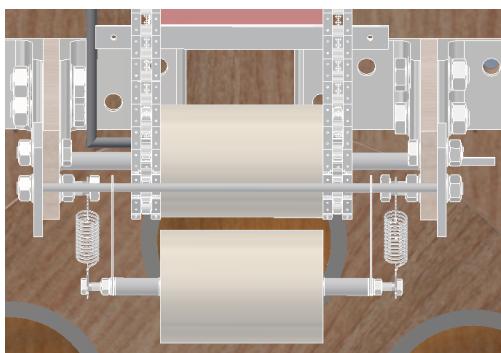


Abbildung 3.62: Pressvorgang

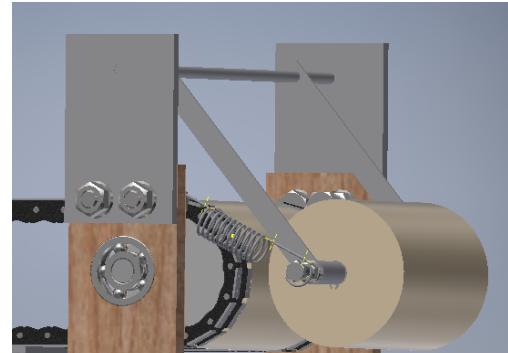


Abbildung 3.63: Walzenhalterung

3.7.5.2 Öffnen und befestigen der Futterpackung

Die Futterpackung wird am Förderband mit zwei Aluminiumplatten aufgehängt. Es wird die Rückseite der Futterpackung per Hand zusammengepresst und zwischen die beiden Platten gelegt. Danach werden die Flügelmuttern angezogen. Auf der Oberseite der Futterpackung wird nach dem hoch quetschen des Futters per Hand die Klemme über die vom Hersteller markierte Schneidelinie befestigt. Diese Linie wird beispielsweise mit einer Schere durchtrennt. Siehe Abbildung: 3.64.

Die Klemme wird durch einen 1 cm dicken gebogenen Draht, ähnelt einem Blitzableiter, geöffnet, indem dieser am Ende einen leichten Haken besitzt in dem der Hebel einhaken kann. Der Draht wird mit zwei Halterungen an der Grundplatte befestigt. Siehe Abbildung: 3.65.

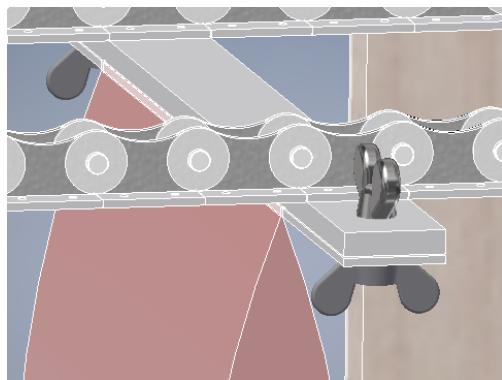


Abbildung 3.64: Aufhängung der Futterpackung

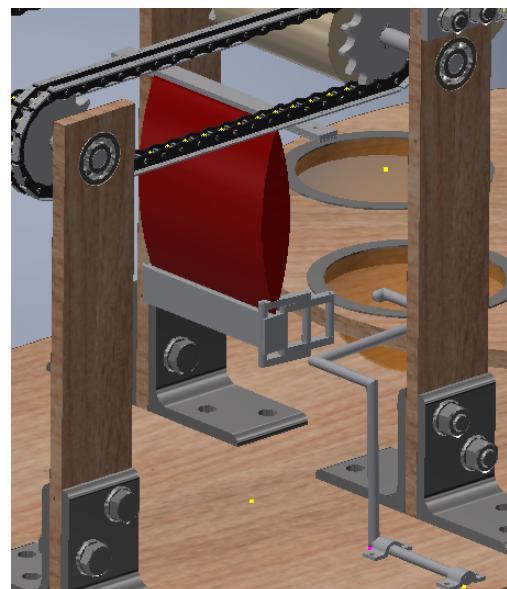


Abbildung 3.65: Öffnen der Hebel-Klemme

3.7.5.3 Gesamtkonstruktion

Hier wird die gesamte Konstruktion gezeigt. Das Förderband verpflegt die Katze zwei Tage und kann je nach Bedarf erweitert werden. Das Gehäuse wurde weg gelassen, um die innere Konstruktion besser sehen zu können. Das Gehäuse wäre nur ein Rechteck mit einer Einbuchtung. Sichtbar wäre nur die befüllte Futterschüssel. Siehe Abbildung: 3.66.

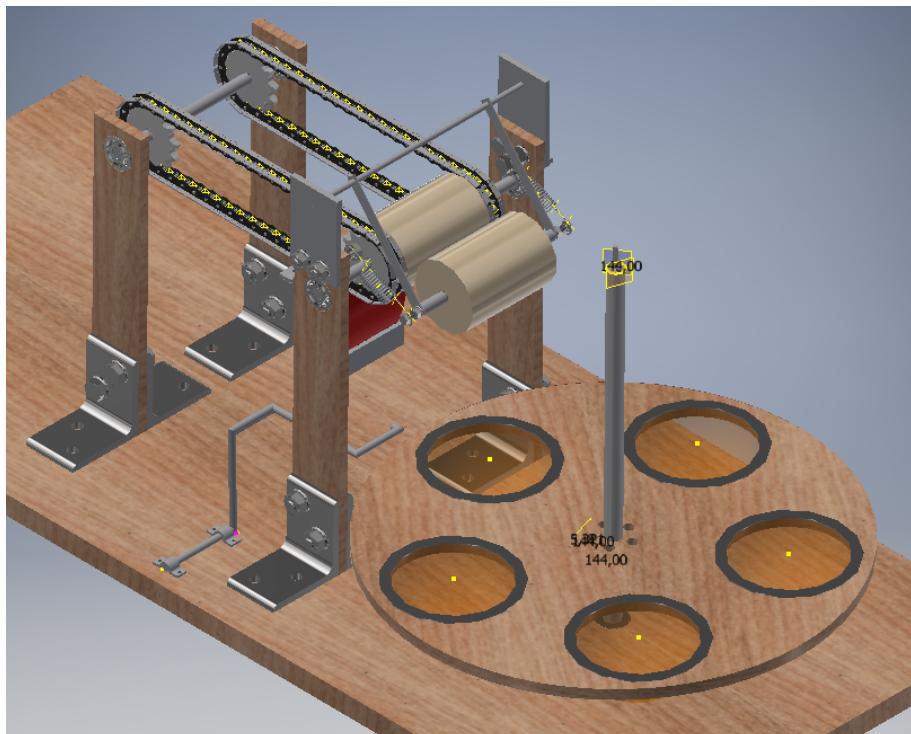


Abbildung 3.66: Gesamtansicht der Konstruktion

3.8 Berechnung und Dimensionierung

In diesem Teil unserer Diplomarbeit werden Berechnungen von wichtigen Teilen bzw. Problemstellen durchgeführt. Es wird bei jeder Berechnung der schlimmstmögliche Fall angenommen.

3.8.1 Berechnung der Hebel-Klemme

Gewählte Presskraft $F = 10\text{N}$, um die Packung dicht zu halten. Die Maße der Fläche auf die die Kraft drückt = $20\text{mm} \times 100\text{mm}$. Der Elastizitätsmodul von Aluminium ist $0.6 \cdot 10^5 \frac{\text{N}}{\text{mm}^2}$. Diese Kraft wurde deshalb so hoch dimensioniert, weil sich beim Klemmen der Verpackung Fleischstückchen dazwischen befinden könnten und somit eine gleichbleibende Pressung nicht gewährleistet wäre. Siehe Abbildung: 3.67.

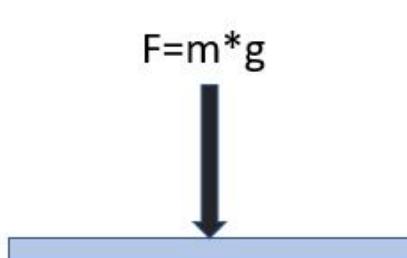


Abbildung 3.67: Skizze der Klemme

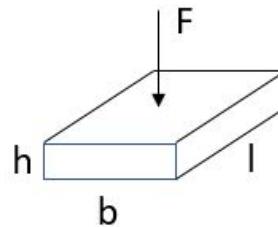


Abbildung 3.68: Querschnitt der Klemme

Formeln zur Berechnung der Durchbiegung in der Mitte der Klemme:

Die Skizze zeigt h und b der Querschnittsfläche. Die Höhe $h = 4\text{ mm}$ und ist parallel zur Kraft. Siehe Abbildung: 3.68.

Das axiale Flächenträgheitsmoment errechnet sich folgender Maßen:²

$$I = \frac{b * h^3}{12} = \frac{20 * 4^3}{12} = 106.66\text{mm}^4$$

Die Durchbiegung errechnet sich laut Böge:³

$$f = \frac{F * l^3}{48 * E * I} = \frac{10\text{N} * 100^3\text{mm}^3}{48 * 0.6 * 10^5 \frac{\text{N}}{\text{mm}^2} * 106.66\text{mm}^4} = 0.032\text{mm}$$

Selbst bei einer Kraft von 10 N ergibt sich eine Durchbiegung von $32\mu\text{mm}$. Die Steifigkeit der Klemme reicht für unsere Zwecke sicher aus.

²Anhang.

³Anhang.

3.8.2 Berechnung der Welle zur Bewegung der Drehplatte

Bei dieser Berechnung wird überprüft, ob der Vierkant auf der Welle mit den Maßen 4mm x 4mm x 20mm ein Drehmoment von maximal 18Ncm des Motors standhält. Die Platte auf der Welle hat einen Durchmesser von 400mm und eine Höhe von 10mm. Es wird der Stahl S235 verwendet mit einem $\tau_{zul} = 165 \frac{N}{mm^2}$.⁴ Siehe Abbildung: 3.69.

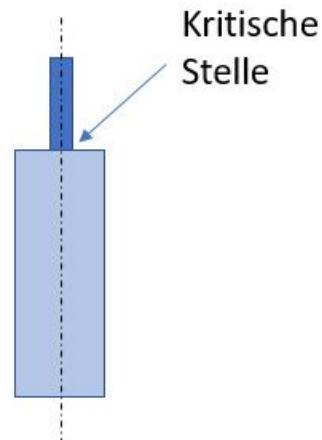


Abbildung 3.69: Kritischer Punkt an der Welle

Hier wird die Winkelbeschleunigung α beim Motormoment von 18Ncm berechnet:

$$J = \frac{1}{2} * r^2 * A_P * \rho * h = \frac{1}{2} * \frac{\pi * 400^4 mm^4}{16} * 800 * 10^{-9} \frac{kg}{mm^3} * 10mm = 20106 kgmm^2 = 0.020 kgm^2$$

$$\alpha = \frac{M}{J} = \frac{0.180 Nm}{0.0201 kgm^2} = 8.9 \frac{rad}{s^2}$$

Berechnung der Torsionsspannung:⁵

$$\tau_T = \frac{M_t}{W_t} = \frac{180 Nmm}{10.66 mm^3} = 16.87 \frac{N}{mm^2}$$

$$W_t = \frac{h^3}{6} = \frac{4^3}{6} = 10.66 mm^3$$

Wird eine 2-fache Sicherheit angenommen:

$$\tau_T = \frac{165}{2} = 82.5 \frac{N}{mm^2}$$

Zum Schluss wird noch kontrolliert ob τ_v kleiner als τ_z ist. Damit wird überprüft ob der Vierkant die Belastung standhält.

⁴Anhang.

⁵Anhang.

$$\tau_{T\text{ vorhanden}} < \tau_{zul} - > 16.87 \frac{N}{mm^2} < 82.5 \frac{N}{mm^2}$$

Fazit der Vierkant hält die Belastung mit Leichtigkeit stand.

3.8.3 Berechnung der Umfangskraft des Förderbandes

Es soll mit einer Presskraft von 400g die Packung durch die Walze ausgequetscht werden. Der Durchmesser der Walze ist 62.7mm.

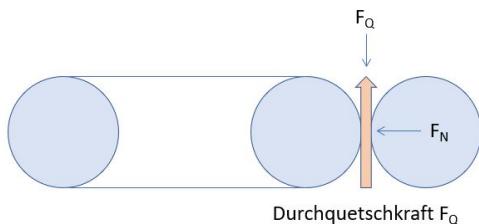


Abbildung 3.70: Skizze des Förderbandes

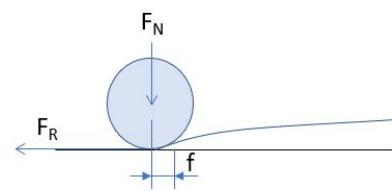


Abbildung 3.71: Reibungskraft

Mit dieser Formel wird ungefähr das μ berechnet. Mit $f=10mm$ und dem Radius der Walze = $31.35mm$. Entspricht einem Reibungskraftbereich eines Autos im Sand. Dabei ist $F_R = \mu_{Roll} * F_N = \frac{f}{r} * F_N$ Siehe Abbildung: 3.71

$$\mu = \frac{f}{r} = \frac{10}{31.35} = 0.315$$

Es wird die erforderliche Umfangskraft F_U berechnet. Die Normalkraft F_N ergibt sich aus der Zugkraft beider Federn, welche im Kapitel 3.8.4 ausgerechnet wurde. Siehe Abbildung: 3.70

$$F_Q = F_U = \mu * F_N = \mu * 2 * F_F = 0.3 * 2 * 3.11N = 1.866N$$

Das erforderliche Moment des Motors wird mit dieser Formel errechnet:

$$M_{erf} = F_U * r = 2N * 31.35mm = 62.7Nm$$

3.8.4 Berechnung der Federkonstante

Nachfolgend wird die Federkonstante ermittelt und die Feder dimensioniert.

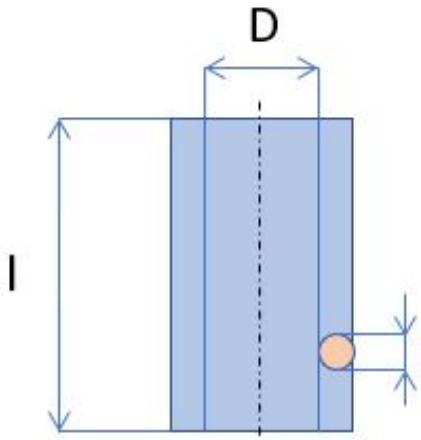


Abbildung 3.72: Federabmessung

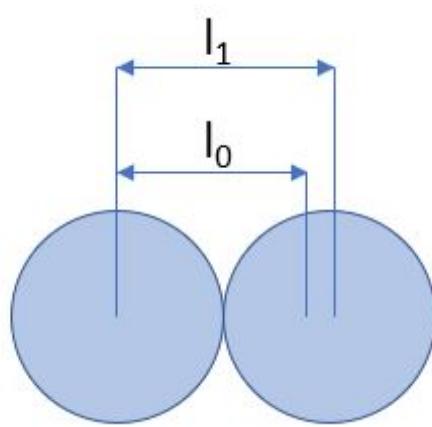


Abbildung 3.73: Walze und Federweg

Berechnung der Federkonstante, bei nicht ausquetschen der Packung $l_0=50\text{mm}$ mit 0N, entspricht der Länge der unbelasteten Feder und $l_1=62.7\text{mm}$ mit 1.962N entspricht 200 Gramm pro Feder. Siehe Abbildung: 3.73.

$$k = \frac{m * g}{l_d} = \frac{0.2 * 9.81}{12.7} = 0.154 \frac{N}{mm}$$

Bei dieser Rechnung wird die vorherige Annahme überprüft und mithilfe des Deckers die Federkonstante nachgerechnet. D ist der Durchmesser der Feder und beträgt 11mm. Das kleine d ist die Dicke des Drahtes und beträgt 1mm. Für die Anzahl der Wicklungen steht n mit 50 Umlaufungen. G ist das Schubmodul mit $80000 \frac{N}{mm^2}$. Siehe Abbildung: 3.72⁶

$$k = \frac{G * d^4}{8 * D^3 * n} = \frac{80000 * 1^4}{8 * 11^3 * 50} = 0.150 \frac{N}{mm}$$

Berechnung der Kraft der Feder bei ausquetschen der Packung, samt Aluminium Halterung. Mit $l_0=50\text{mm}$ und $l_1=70.7\text{mm}$. Die Längsänderung ist also 20,7 mm.

$$F = k * l_d = 0.150 * 20.7mm = 3.11N$$

Da zwei Federn parallel geschallten sind ergibt sich eine Quetschkraft von 6.22N

3.9 Simulation

Es werden verschiedene Simulationen erstellt, um die Richtigkeit der Rechnungen zu überprüfen.

⁶Decker.

3.9.1 Simulation der Klemme

Anhand der Simulation erkennt man, die Biegung der Fläche bei einer einwirkenden Kraft von 10N. Der simulierte Wert siehe Abbildung: 3.74 und der berechnete Wert siehe Kapitel 3.8.1 sind nahezu gleich.

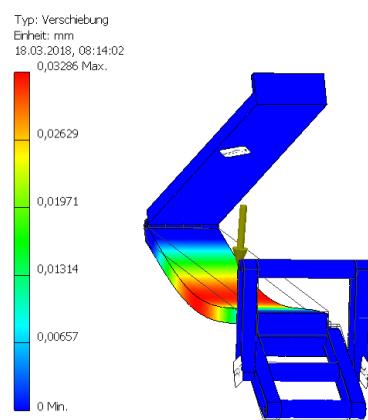


Abbildung 3.74: Hebel-Klemme Simulation

3.9.2 Simulation der Feder

Die Simulation zeigt die Federauslenkung, welche die Feder bei einer Kraft von 3,11N auseinander drückt siehe Abbildung: 3.75. Der ausgerechnete Federweg beträgt 20,7mm siehe Kapitel 3.8.4.

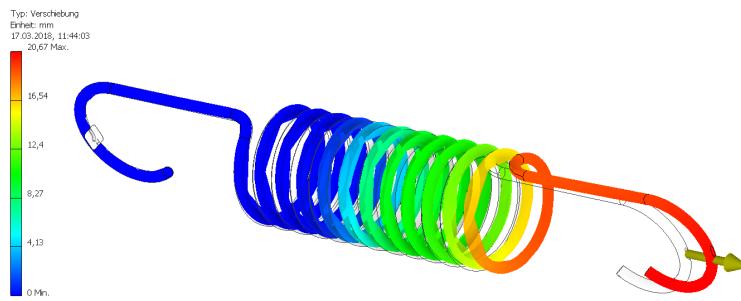


Abbildung 3.75: Federauslenkungs Simulation

3.10 Bedienung und Wartung

Bei der Entwicklung der Maschine wurde darauf geachtet, diese so Benutzerfreundlich wie nur möglich zu gestalten. Darum lässt sich das Gehäuse der Maschine bis zur Hälfte Aufklappen, um den Benutzer einen leichten Zugang zu verschaffen. Es mussten auch keine speziellen Vorkehrungen getroffen werden, um den Benutzer zu schützen, da es keine gefährlichen Stellen in der Maschine gibt, den Personen schaden bzw. verletzen könnten. Die Befestigung der Futterpackungen ist auch ein Leichtes, durch den großen Platz der zur Verfügung steht und wenn man sich an folgende Schritte hält:

- 1 Einspannen der Hinterseite der Packungen auf die Aluminiumplatten des Förderbandes.
- 2 Die Hebelklemme kurz nach der vom Futterhersteller vorgeschriebenen Schneidelinie festklemmen. (Dabei zu beachten, dass sich der Hebel in Richtung der Walze auf der rechten Seite befindet).

- 3 Mit einer Schere oder einem Messer die Schneidelinie durchtrennen und den Abfall entsorgen.
- 4 Die gewünschte Anzahl an Packungen festhängen, maximal 10 Packungen.

Weiteres zur Wartung, nachdem der Benutzer die Maschine mehrmals verwendet hat, sollte man folgende Teile der Maschine reinigen:

- 1 Die Walze, hierbei kann durch das Ausquetschen der Packungen etwas Gelee an den beiden Walzen hängen bleiben. Einfach mit einem nassen Tuch abwischen.
- 2 Die Futterschüsseln können mit der Hand entnommen werden indem man auf der Unterseite der Platte die Schüssel nach oben drückt und mit der freien Hand entnimmt. Danach die Schüssel waschen und in die Platte zurück legen.
- 3 Die Oberfläche der Futterplatte kann nach dem entfernen der Schüssel mit einem nassen Tuch gereinigt werden.

3.11 Selbstkritische Analyse und Ausblick

Das Hauptproblem von meiner konstruierten Variante war das Dichthalten der Packung mit einer Klemme die sich automatisch, wenn das Förderband in Betrieb ist, öffnet. Darum wurde von uns die Hebel-Klemme konstruiert. Die beim Betrieb des Förderbandes mit dem Hebel in eine Vorrichtung einhakt und diese dann die Klemme öffnet.

Für Nachfolgeprojekte wäre besonders darauf zu achten, dass die Maschine von automatisch gereinigt wird und womöglich eine Schneidefunktion besitzt, die den Benutzer mehr entlasten könnte.

Wenn ich in Zukunft noch einmal eine Katzenfuttermaschine bauen würde, würde ich diese nicht aus Säckchen, sondern aus den kleinen Metallfutterpackungen mit der Lasche als Deckel verwendet. Der Benutzer müsste dann nur die Laschen so weit nach oben biegen, dass der Deckel nicht aufgeht und in das Magazin einlegen. Die Packung wird daraufhin in Bewegung gebracht und die Lasche durch einen Haken geleitet. Dann öffnet sich die Packung und die Katze kann daraus fressen. Dies hätte den Vorteil immer eine frische Schüssel zu haben und es könnten theoretisch so viele Packungen wie der Benutzer will, in die Maschine eingelegt werden.

KAPITEL 4

Elektronik und Mechanik

4.1 Anforderung Elektronik Variante 1

4.1.1 Aktoren

Die Aufgabe der Aktoren im elektrischen Teil besteht darin, ein Förderband und eine Drehplatte anzutreiben. Für beide Anwendungen werden Motoren mit einem hohen Drehmoment und einer niedrigen Drehzahl benötigt. Weiteres dürfen sich die Motoren nicht drehen, wenn sie nicht angesteuert werden. Zur Positionierung, Befestigung und Durchführung anderer linearer Bewegungen werden Hubmagnete in Form von Zylindermagneten benötigt. Diese haben die Aufgabe Futterbeutel zu verschieben, zu positionieren, festzuhalten und einen Scherenhebel zum öffnen der Futterbeutel zu betätigen.

4.1.2 Sensoren

Die Aufgabe der Sensoren im elektrischen Teil besteht darin, zu erkennen, ob ein Objekt vor ihnen steht. Dadurch kann überprüft werden, ob Futterschüsseln und Futterbeutel an der richtigen Stelle stehen.

4.1.3 Ansteuerungen

Die Aufgabe der Ansteuerung ist es, mithilfe eines Raspberries und einem Arduino Nano, Motoren anzusteuern so wie Sensoren abzufragen.

4.2 Anforderung Elektronik Variante 2

4.2.1 Aktoren

Die Aufgabe der Aktoren im elektrischen Teil besteht darin, ein Förderband und eine Drehplatte anzutreiben. Für beide Anwendungen werden Motoren mit einem hohen Drehmoment und einer niedrigen Drehzahl benötigt. Zusätzlich dürfen sich die Motoren nicht drehen, wenn sie nicht angesteuert werden.

4.2.2 Sensoren

Die Aufgabe der Sensoren im elektrischen Teil besteht darin, zu erkennen, ob ein Objekt vor ihnen steht. Dadurch kann überprüft werden, ob Futterschüsseln und Futterbeutel an der richtigen Stelle stehen.

4.2.3 Ansteuerungen

Die Aufgabe der Ansteuerung ist es, mithilfe eines Raspberries und einem Arduino Nano, Motoren anzusteuern so wie Sensoren abzufragen.

4.3 Variante 1

4.3.1 Aktoren

4.3.1.1 Hubmagnete



Abbildung 4.1: Hubmagnet

Zylinderhubmagnete bestehen aus einem zylindrischen Dauermagnetkern. Umgeben ist dieser Magnetkern von einer Spule und einem Gehäuse, welches als Schutz vor Festkörpern dient. Wird an die Spule eine Spannung angelegt, fließt ein Strom durch die Spule und erzeugt ein Magnetfeld welches den Magnetkern anzieht oder abstößt. Durch eine gezielte Übersteuerung des Magnetzyllinders kann die Kraft des Magnetfeldes kurzzeitig verstärkt werden, jedoch verhindert die erhöhte Hitzeentwicklung einen dauerhaften Betrieb im übersteuerten Zustand. Der übersteuerte Zustand wird mit einer geringeren relativen Einschaltdauer gegenüber der Zykluszeit bewirkt.

Der Strom, der durch die Spule fließt, kann sich durch die kurze Einschaltdauer nicht einpendeln was der Spule erlaubt mit einem durchschnittlich höheren Strom betrieben zu werden, das wiederum zu einem stärkeren Magnetfeld und somit zu einer stärkeren Kraft führt. Die genauen Werte der Kraft zur relativen Einschaltdauer lassen sich dem Datenblatt jedes einzelnen Hubmagneten entnehmen. Ein Vorteil von Hubmagneten ist es, dass sie in allen möglichen Formen, Hublängen, Hubkräften und Wechsel- oder Gleichspannungsbereichen verfügbar sind. Hubmagnete können mit drei verschiedenen Hubzuständen erworben werden. Monostabile Hubmagnete halten sich mithilfe einer Feder bei stromlosem Zustand immer an der Anfangs oder Endposition. Bistabile Hubmagnete verweilen im stromlosem Zustand an ihrer aktuellen Position, also entweder im eingefahrenen oder ausgefahrenen Zustand. Tristabile Hubmagnete bleiben, wie bistabile Hubmagnete, stromlos an ihrer aktuellen Position. Jedoch kann diese Hubmagnetart auch an einer vorgegebenen Mittelposition verweilen.

4.3.1.2 Motoren

Nach einigen Problemen und Variantenänderungen in der Mechanik, wurde die Motorenauswahl erst in Variante 2 vorgenommen, da die Auswahl für Variante 1 sinnfrei wäre.

4.3.2 Sensoren

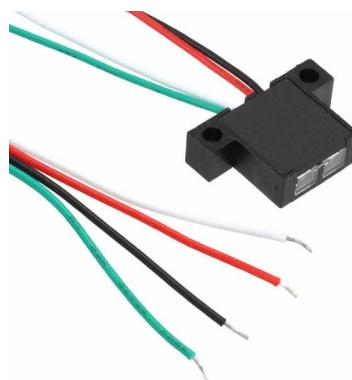


Abbildung 4.2: Sensor

Die Auswahl des Sensors für unsere Anwendung wurde durch folgende Kriterien beeinflusst: Befestigungsmöglichkeiten, Preis und zusätzlich benötigte Ansteuerungen. Kapazitive und induktive Näherungsschalter lassen sich einfach befestigen, sind jedoch in der Anschaffung teuer. Die Reichweite der Sensoren ist nicht sehr groß, was aber für unsere Anwendung nicht relevant ist. Für diese Arten von Sensoren werden zusätzliche Ansteuerungen benötigt. Diese Ansteuerungen sind meistens etwas teurer und benötigen zur Kommunikation mit einem Raspberry eine Schnittstelle. Aus diesen Gründen haben wir uns in diesem Anwendungsbereich für einen optischen Näherungsschalter entschieden. Dieser beinhaltet ein fertiges Modul bestehend aus einer Infrarot LED und einem Infrarot Fototransistor. Dieses Modul ist preiswert und leicht zu befestigen. Dadurch, dass

unsere Anwendung keine genaue Abstandsmessung erfordert, wird keine zusätzliche Ansteuerung benötigt. Zur Stromversorgung reichen 5V Gleichspannung mit einem laut Datenblatt berechneten Vorwiderstand. Die Auswertung erfolgt durch einen digitalen Input-Pin des Raspberries. Sobald sich eine Reflektorfläche vor dem Sensor bewegt, wird der Fototransistor niederohmig und ermöglicht einen Stromfluss. Dadurch liegt eine Spannung von 3,3V am Input-Pin an und wird vom Raspberry als HIGH-Zustand angesehen. Wenn die Reflektorfläche vom Modul nicht mehr erfasst wird, sperrt der Fototransistor, der Stromfluss wird unterbunden und am Input-Pin liegen nur mehr 0V an, was als LOW-Zustand angesehen wird.

Kennlinie des Sensors:

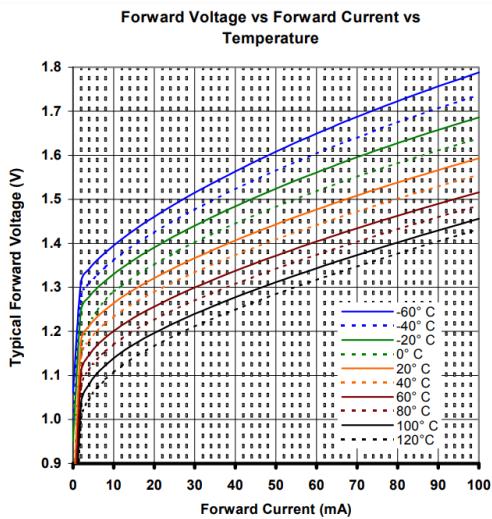


Abbildung 4.3: Diodenspannung zu Diodenstrom

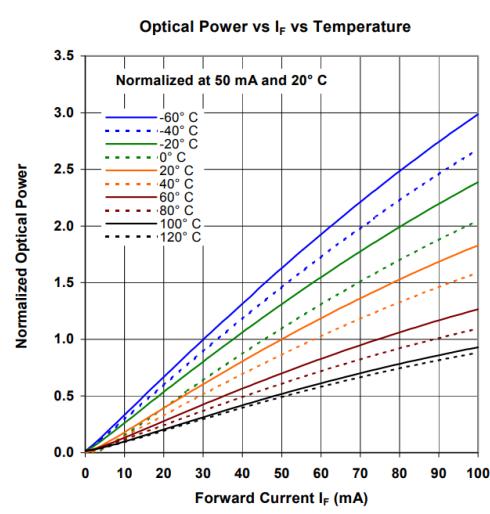


Abbildung 4.4: Optische Übertragungsstärke zu Diodenstrom

4.3.3 Ansteuerungen

Da Variante 1 nur kurze Zeit verfolgt wurde, wurden genaue Ansteuerungen für Motoren und Hubmagnete nicht herausgesucht. Der Sensor benötigt keine zusätzliche Ansteuerung.

4.4 Variante 2

4.4.1 Aktoren

4.4.1.1 Hubmagnete

Für die zweite Variante werden keine Hubmagnete benötigt.

4.4.1.2 Motoren



Abbildung 4.5: Motor

Die Motoren wurden mithilfe folgender Kriterien ausgesucht: Preis, Drehzahl, Kraft, stromloses Verhalten und Spannung. Wir haben uns darauf festgelegt nur mit sicherer niedriger Gleichspannung zu arbeiten. Daher kommen nur Motoren mit einem Spannungsbereich von 0-24V in Frage. Aus diesem Grund werden die Auswahl eines Asynchronmotors ausgeschlossen. Die Anwendung fordert, dass sich die Motoren im stromlosem Zustand nicht bewegen lassen. Dadurch können folgende Motorarten ausgeschlossen werden: Synchronmotoren, reine Gleichstrommotoren, Schrittmotoren und Servomotoren, falls diese Motoren nicht über eine aktive Bremse verfügen. Ideal für unsere Anwendungen sind Gleichstrom-Schneckengetriebe-Motoren. Diese Motoren lassen sich aufgrund des mechanischen Aufbaus im stromlosem Zustand nicht drehen. Sie verfügen meistens auch über ein Getriebe, welches eine Übersetzung in eine niedrigere Drehzahl zur Folge hat. Gleichzeitig wird mit diesem Getriebe auch das Drehmoment erhöht, wodurch der Motor eine höhere Kraft aufbringen kann. Da Schneckengetriebemotoren in der Automobil Industrie als Scheibenwischermotoren verwendet werden, sind diese sehr preisgünstig zu erwerben.

4.4.1.3 Lüfter



Abbildung 4.6: Lüfter

Zur Kühlung des Raspberries, findet ein externer 12V-Lüfter Verwendung, welcher einen Luftstrom um den Raspberry und den Mikrocontroller (μ C)ontroller erzeugt.

4.4.2 Sensoren

Der ausgewählte optische Sensor wurde in Variante 1 bereits beschrieben. Für genauere Informationen lesen Sie bitte in Punkt 4.3.2 nach. Es werden zwei dieser Sensoren zur Positionierung verwendet. Der Sensor muss nur erkennen, ob sich eine Reflektorfläche vor ihm befindet. Bei der Futterschüsseldrehplatte wird am Rand der Scheibe, an der Position jeder Futtergeschüssel ein Reflektorstreifen verwendet. Dadurch kann mit der Software gezählt und erkannt werden, welche Futtergeschüssel sich aktuell vor dem Sensor befindet. Im Futterbeutelmagazin wird der Sensor dazu verwendet, um erkennen zu können, ob ein Futterbeutel für die Fütterung verfügbar ist. Wird vom Sensor, nach Drehen des Motors, kein weiterer Futterbeutel erkannt, wird von der Software, falls nötig, ein Fehler ausgegeben und der Benutzer informiert. Der Reflektorstreifen wird auf der Verschlussklemme jedes Futterbeutels befestigt.

4.4.3 Ansteuerung

Der Gesamtschaltplan wurde in ProfiCad realisiert. Der Schaltplan beinhaltet die Ansteuerung der Motoren und Sensoren, sowie die Visualisierung der Verkabelung der einzelnen Elemente. Für ein einfacheres Verständnis wird der Gesamtschaltplan in kleinere Elemente aufgeteilt und in Unterpunkten beschrieben.

4.4.3.1 Motoransteuerung Variante 1

Der Gleichstrom Schneckengetriebemotor in unserer Anwendung kann in beide Richtungen betrieben werden, je nach Polarität der Anschlüsse. Um eine softwaretechnisch einfache Ansteuerung ermöglichen zu können, wird oft eine H-Brücke zur Ansteuerung der Motoren verwendet. Diese H-Brücke besteht aus vier Transistoren. Für Motoren mit höheren Stromanforderungen werden MOSFET-Transistoren verwendet. Die Schaltung besteht aus zwei P-Kanal MOSFET T1, T3 und aus zwei N-Kanal MOSFET T2, T4. Die Schaltung laut Abbildung 4.9, wurde aus dem Web¹ entnommen und abgeändert. Dabei wurden die Bauteilwerte der MOSFETs und Teile der Schaltung übernommen. Der Raspberry steuert über Optokoppler die MOSFETs. Dabei wird über eine entsprechende Vorschaltung entweder das Gate vom MOSFET auf Ground- oder 12V-Potenzial gezogen. Zur Veranschaulichung wurden in Abbildung 4.9 die Raspberry Pins mit einer Klemme ersetzt, wobei die Klemme 1, T1, Klemme 2, T2, Klemme 3, T3 und Klemme 4, T4 steuert.

Es gibt bei dieser Ansteuerungsart folgende erwünschte Zustände der Klemmen:

Klemme	Uhrzeigersinn	gegen Uhrzeigersinn	Auslaufen	Bremsen	Bremsen
1	HIGH	LOW	LOW	HIGH	LOW
2	LOW	HIGH	LOW	LOW	HIGH
3	LOW	HIGH	LOW	HIGH	LOW
4	HIGH	LOW	LOW	LOW	HIGH

Tabelle 4.1: H-Brückenzustände

Jede unterschiedliche Beschaltungsart als die oben angeführten, würde zu einem Kurzschluss führen und muss softwaretechnisch verhindert werden.

Kennlinie der Transistoren:

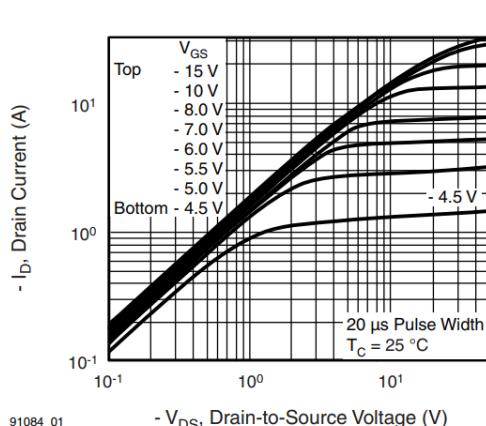


Abbildung 4.7: P-Kanal

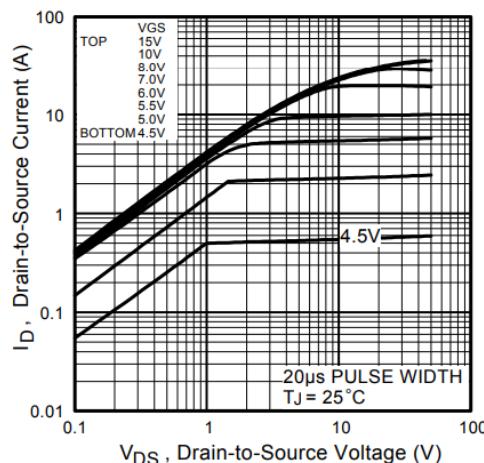


Abbildung 4.8: N-Kanal

¹Lewis Loflin. *H-Bridge Motor Control with Power MOSFETS*.

http://www.bristolwatch.com/ele/h_bridge.htm.

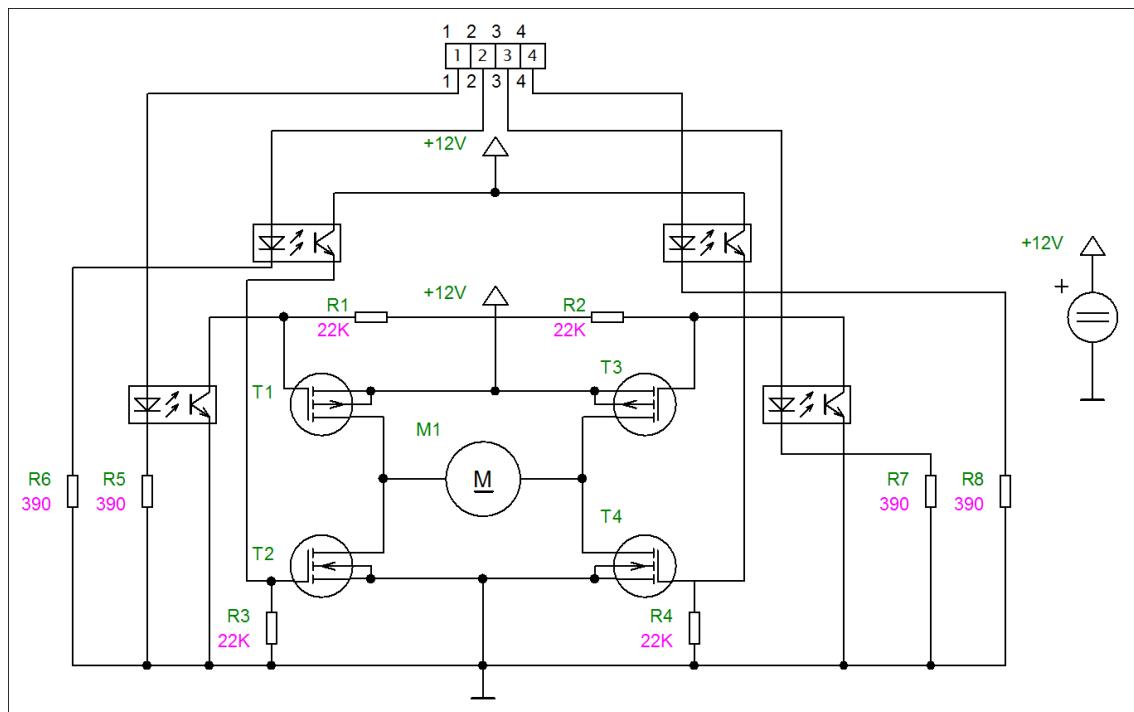


Abbildung 4.9: H-Bridge

Dieser Schaltplan dient nur als Beispiel-Schaltung und wurde aufgrund der Motoransteuerung Variante 2, nicht mehr weiter verfolgt. Um die Schaltung wirklich benutzen zu können, müsste eine Sicherung gegen einen Kurzschluss eingebaut werden. Für unsere Anwendung müsste eine Drehzahlregelung der Motoren Verwendung finden.

4.4.3.2 Motoransteuerung Variante 2



Abbildung 4.10: Motorsteuerung

Aus sicherheits- und kostentechnischen Gründen, wurde von einer manuell gesteuerten H-Brücke auf ein fertiges Motorsteuerungsmodul umgestiegen. Der Vorteil dieser Ansteuerung besteht darin, dass ein Kurzschluss der H-Brücke hardwaretechnisch ausgeschlossen werden kann und damit nicht von der Software abgesichert werden muss. Das Motormodul verfügt außerdem über eine interne Strommessung für jeden der zwei unterstützten Motoren. Diese Messung kann bei Bedarf abgefragt werden. Für unsere Anwendung ist dies jedoch nicht erforderlich. Der Motortreiber unterstützt zwei Motoren, welche mit bis zu 30 Ampere bei 16V betrieben werden können, bevor der IC überhitzt oder andere Bauteile zerstört werden. Die Ansteuerung jedes Motors erfolgt über ein digitales Signal. Motor Nr.1 kann über den Pin AI0 aktiviert (enabled) werden. Dies dient zur Sicherung, dass der Motor nicht aus Versehen betätigt werden kann, solange der AI0 Pin nicht auf HIGH ist. Der enable Pin für Motor Nr.2 ist AI1. Um Motor Nr.1 im Uhrzeigersinn drehen zu lassen, muss der Pin D7 mit einem HIGH Signal und der Pin D8 mit einem LOW Signal angesteuert werden. Um Motor Nr.2 gegen den Uhrzeigersinn drehen zu lassen, muss der Pin D8 mit einem HIGH Signal und der Pin D7 mit einem LOW Signal angesteuert werden. Um den Motor Nr.1 bremsen zu können, muss Pin D8 und D7 mit entweder einem HIGH oder einem LOW Signal angesteuert werden. Für Motor Nr.2 ist die Funktionsweise gleich, jedoch muss der Pin D7 mit Pin D4 und Pin D8 mit Pin D9 ersetzt werden. Die externe Stromversorgung für den IC von 5V erfolgt über den VCC Pin. Die externe Stromversorgung für die Motoren erfolgt über den PWR Pin. Zusätzlich muss der GND Pin noch mit einem der beiden Ground Anschlüsse der externen Stromversorgungen verbunden werden. Motor Nr.1 wird über die Outputs A1 und B1 versorgt. Motor Nr.2 wird über die Outputs A2 und B2 versorgt. Beim Anschluss der Motoren muss auf die Polung geachtet und gegebenenfalls getestet werden, ob sich die Motoren auch wirklich beim Ansteuern der D7 oder D4 Pins im Uhrzeigersinn drehen und nicht gegen den Uhrzeigersinn. Falls beim Motor gekennzeichnet, sollte der Pluspol des Motors mit dem A1 beziehungsweise dem A2 Pin verbunden werden. Der Motortreiber verfügt auch über eine interne Drehzahlregelung, welche extra für beide Motoren über zwei Pulse Width Modulation (PWM) Input Pins gesteuert werden kann. Die Drehzahl kann über den Duty Cycle des PWM Signals geregelt werden.

4.4.3.3 PWM-Signal-erzeugung für Motoransteuerung

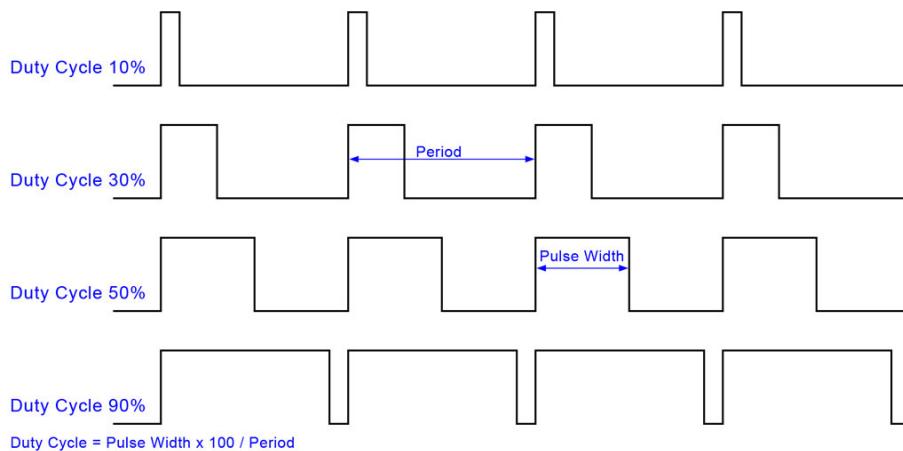


Abbildung 4.11: PWM-Signal

Ein PWM Signal ist ein periodisches Rechteck-Signal. Eine Periode wird als Duty Cycle bezeichnet. Dieser Duty Cycle enthält eine Prozentangabe, welche aussagt, wie viel Prozent des Duty Cycles HIGH und wie viel Prozent LOW sind. Ein Signal mit einem Duty Cycle von 100% sagt aus, dass eine Periodendauer zu 100% aus einem HIGH Signal und zu 0% aus einem LOW Signal besteht. Ein Duty Cycle von 25% sagt aus, dass eine Periode zu 25% aus einem HIGH Signal und zu 75% aus einem LOW Signal besteht. Das PWM Signal kann von manchen ICs intern aufgenommen und der Duty Cycle ausgemessen werden. Dabei muss jedoch beachtet werden, dass die Frequenz des PWM Signals nicht die maximale Schaltfrequenz des ICs übersteigt. In den meisten Fällen wird diese Frequenz, wie bei unserem Motortreiber, im Datenblatt angegeben. Das PWM kann entweder über eigene ICs mit PWM Generatoren oder μ Cs mit internen PWM Generatoren erzeugt werden. Das System muss nur über eine Echtzeitfähigkeit besitzen, damit der Duty Cycle keine Takte beim Zählen überspringen und nicht die Frequenz fehlerhaft werden kann. Unser PWM Generator wurde mithilfe eines Arduino Nanos realisiert. Mithilfe des μ C Atmega328P wird der interne Zähler (Timer 2) dazu verwendet, einen Output Pin in den richtigen Abständen zu toggeln (von HIGH auf LOW oder umgekehrt schalten) um ein PWM Signal zu erzeugen.

- verwendete Register:
 - TCCR2A: Im TCCR2A Register wird festgelegt, in welchem Modus der Timer laufen soll und was mit den Pins OC2A und OC2B passieren soll, wenn der der aktuelle Timer-Wert mit einem der beiden Output Compare Register übereinstimmt. Die Bits COM2A1 und COM2A0 setzen im Fast-PWM Modus fest, ob der OC2A Pin nicht beeinflusst wird und nicht verbunden ist, vom WGM22 Bit abhängig ist oder bei einem Compare Match, also wenn das Output Compare Register OCR2A mit dem aktuellen Timer Wert übereinstimmt, der OC2A Pin gesetzt und beim Reset des Timers resetet wird (non-inverting Mode), oder bei einem Compare Match der OC2A Pin resetet und

beim Reset des Timers gesetzt wird (inverting Mode).

Die selbe Funktion haben die Bits COM2B1 und COM2B0, nur dass der Pin OC2B verändert und mit dem Compare Register OCR2B verglichen wird.

Die Bits WGM21 und WGM20 setzen den Modus des Signalgenerators, die vom Timer unterstützt werden. Es gibt grundsätzlich drei Modi, in denen der Signalgenerator laufen kann: CTC, Fast PWM, PWM Phase Correct.

- * Im CTC Modus kann der TOP Wert, zu dem der Counter zählt, immer bei jedem vollständigen Zählvorgang geändert werden. Der Zähler zählt in Form eines Sägezahn Signals und toggled den Pin immer, wenn der Top Wert erreicht wird.
 - * Im Fast PWM Modus zählt der Zähler in Form eines Sägezahnsignals und es wird bei jedem Zählschritt überprüft, ob der aktuelle Wert des Zählers mit dem Output Compare Register übereinstimmt. Was passiert, wenn die Werte übereinstimmen, wird in den COM2xx Bits festgelegt. Die Frequenz des Ausgangssignals am Pin kann nur über einen Prescaler verändert werden. Das bedeutet, dass die Frequenz nur sieben Frequenzen mit dem internen Timer annehmen kann. Das OCR2x Register kann während des Betriebes geändert werden. Damit kann der Duty Cycle während des Betriebes geändert werden.
 - * Der PWM Phase Correct Modus, lässt den Zähler von Bottom zu Top und wieder zu Bottom zählen. Das bewirkt, dass er Zähler in Form eines Dreieckssignals zählt. Somit ist das Ausgangs PWM Signal immer symmetrisch um den BOTTOM und TOP Wert des Zählers. Daher bleibt das Signal immer Phasen-korrekt. Der Nachteil dieses Modus ist, dass er nur die halbe Frequenz des Fast PWM Modus erreichen kann. Der PWM Phase Correct Modus kann auch nur sieben Frequenzen annehmen.
- TCCR2B: Im TCCR2B Register wird für unsere Anwendung der Prescaler festgelegt, um die Frequenz des PWM-Signales einzustellen. Dazu werden die Bits 0 bis 2 verwendet. Das sind die Bits CS20, CS21 und CS22. Die Bits FOC2A, FOC2B und WGM22 werden in unserer Anwendung nicht benötigt und können ignoriert werden.
- OCR2A: Das Register OCR2A ist das Output Compare Match Register für den OC2A Pin. Das Register kann einen Wert zwischen 0 und 255 haben, wobei im Fast PWM Modus 0 einem Duty Cycle von 0% und 255 einem Duty Cycle von 100% entspricht.
- OCR2B: Das Register OCR2B ist das Output Compare Match Register für den OC2B

Pin. Das Register kann einen Wert zwischen 0 und 255 haben, wobei im Fast PWM Modus 0 einem Duty Cycle von 0% und 255 einem Duty Cycle von 100% entspricht.

- DDRD: Im Register DDRD können I/O Pins als Output Pins gesetzt werden. Es muss nur das Bit des entsprechenden Pins gesetzt werden.
- DDRB: Im Register DDBR können I/O Pins als Output Pins gesetzt werden. Es muss nur das Bit des entsprechenden Pins gesetzt werden.

- Frequenz berechnen:

Die maximal Frequenz des PWM-Signals für die Motoransteuerung beträgt 20kHz. Das bedeutet, dass man einen Frequenzwert so nahe wie möglich, aber unter der maximalen Frequenz wählen soll.

Berechnen lässt sich die Frequenz des PWM-Signals mit folgender Formel:

$$f_{OC2xPWM} = \frac{f_{clk_I/O}}{N * 256}$$

$f_{OC2xPWM}$ ist die Frequenz, die das PWM-Signal haben wird.

$f_{clk_I/O}$ ist die Taktfrequenz des Atmega328P, also 16MHz.

N ist der Wert des Prescalers, welcher im TCCR2B Register festgelegt wird.

Frequenz Berechnung:

$$f_{OC2xPWM} = \frac{f_{clk_I/O}}{N * 256} = \frac{16000000}{8 \cdot 256} = 7812,5 \text{ Hz}$$

- Programm für Atmega328P:

```

1 void app_main (void)
2 {
3     DDRD = (1<<PD3);
4     DDBR = (1<<PB3);
5     TCCR2A = (1<<COM2A1) | (1<<COM2B1) | (1<<WGM21) | (1<<WGM20);
6     TCCR2B = (1<<CS21);
7     OCR2A = 0x5f;
8     OCR2B = 0x5f;
9
10 }
```

- DDRD: Im DDRD Register wird der PD3 Pin auf Output geschaltet.
- DDRB: Im DDRB Register wird der PB3 Pin auf Output geschaltet.
- TCCR2A: Im TCCR2A Register wird der Non-Inverting Fast-PWM Modus eingestellt für OC2A und OC2B.
- TCCR2B: Im TCCR2B Register wird der Prescaler festgelegt.
- OCR2A: Im OCR2A Register wird der Duty Cycle festgelegt. 0 entspricht 0% und 255 entspricht 100%
- OCR2B: Im OCR2B Register wird der Duty Cycle festgelegt. 0 entspricht 0% und 255 entspricht 100%

Ein Signal, dass nach dem oben angeführten Programm generiert wird, sieht wie in Abbildung 4.12 aus.

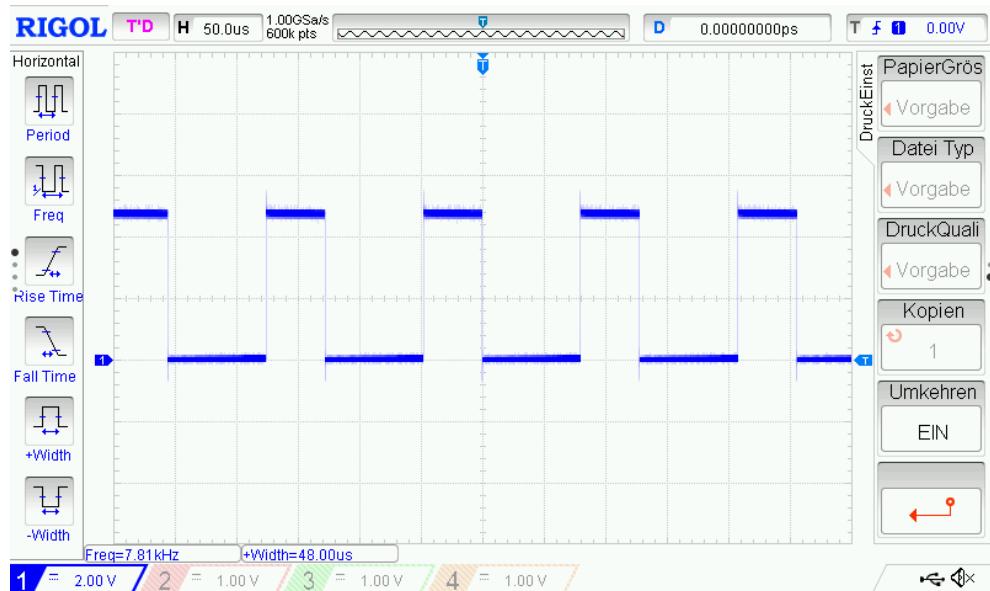


Abbildung 4.12: PWM-Signal

4.4.3.4 Optokoppler

Die Optokoppler in unserer Anwendung werden zur Potenzialerhöhung oder Potenzialverminde-
rung von 3,3V auf 5V oder umgekehrt verwendet. Durch eine LED und einen Phototransistor
werden die Schaltkreise getrennt. Wenn an der LED eine Spannung anliegt, beginnt sie zu leuch-
ten und erhöht damit die Leitfähigkeit des Phototransistors. Dadurch ermöglicht ein Stromfluss
der LED-Schaltung einen Stromfluss in der Phototransistor-Schaltung.

Kennlinie des Optokopplers:

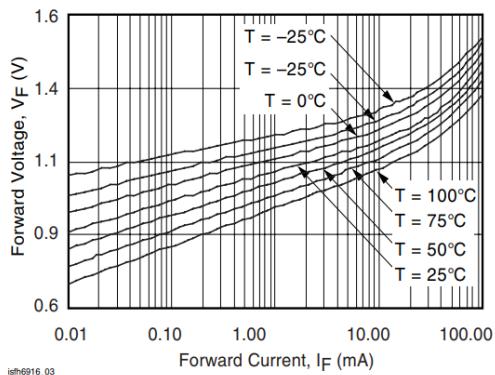


Abbildung 4.13: Diodenstrom zu
Diodenspannung

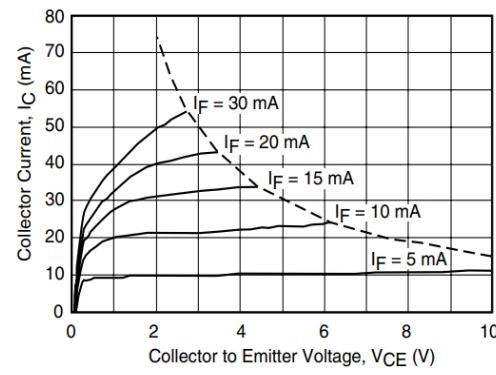
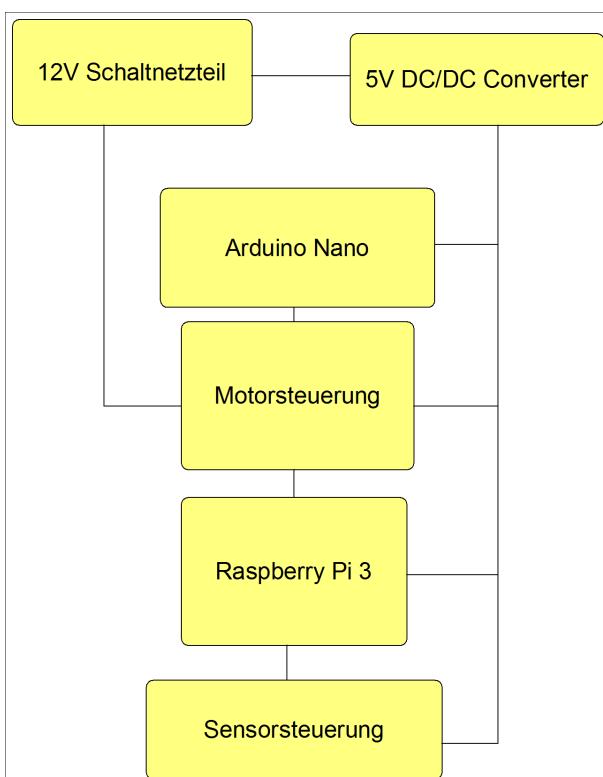


Abbildung 4.14: Transistorstrom zu
Transistorspannung

4.4.3.5 Gesamtschaltplan



Das Blockschaltbild vereinfacht die Ge-
samtschaltung und zeigt an, welche Bau-
teile miteinander verbunden sind. Die ein-
zelnen Bauteile werden in den weiteren
Punkten noch genauer beschrieben. Ver-
bindungen der Schaltpläne erfolgen mit-
tels der Klemmen X1 bis X6.

Abbildung 4.15: Blockschaltbild

4.4.3.6 Verkabelung der Motoransteuerung mit dazugehörigen Elementen

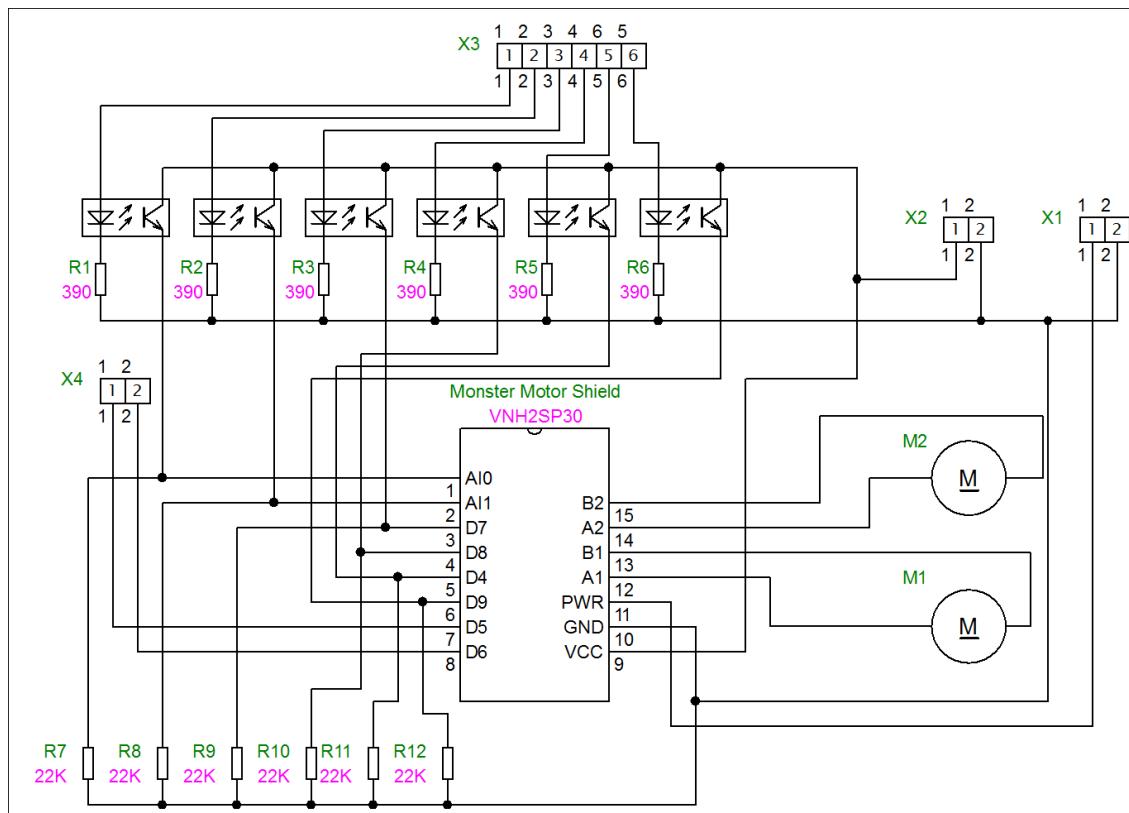


Abbildung 4.16: Motoransteuerung

Die Motoransteuerung besteht aus folgenden Komponenten:

- Dem Motortreiber mit dem IC VNH2SP30.
- Der Klemme X4, X3, X2, X1.
- Sechs Optokopplern, welche die Ausgangsspannung von den GPIO-Pins auf 5V anheben.
- Den beiden Motoren M1 und M2.
- Den Widerständen R1 bis R6, welche den Strom begrenzen, damit der Optokoppler nicht überlastet wird.
- Den Pulldown-Widerständen R7 bis R12 welche ein floaten, also eine fehlerhafte Spannung an den Input-Pins der Motorsteuerung verhindern sollen.

Genereller Ablauf:

1. Der Arduino Nano erzeugt ein dauerhaftes PWM-Signal, welches an die PWM-Inputs D6 und D5, über die Klemme X4, der Motoransteuerung gelegt wird.
2. Die GPIO-Pins geben ein High-Signal über die Klemme X3, von 3,3V aus, welches die Optokoppler ansteuert.
3. Der Optokoppler schaltet seinen internen Phototransistor durch und ermöglicht einen Stromfluss.
4. Der Strom fließt nun von der 5V Spannungsversorgung über die Klemme X2, über den Optokoppler in den Input-Pin der Motoransteuerung.
5. Je nach Ansteuerung der Pins, dreht sich einer oder beide Motoren im oder gegen den Uhrzeigersinn, oder werden gebremst. Dabei fließt ein Strom von der 12V Spannungsversorgung über die Klemme X1, über die Motoransteuerung zu den Motoren.
6. Die Motoransteuerung regelt die Spannung an den Motoren mittels des PWM-Signals und regelt auch die Drehrichtung der Motoren je nach Ansteuerung.

Berechnungen:

- R1 bis R6:

Die Spannung, die abfällt und der Strom, der fließt, kann aus dem Datenblatt des Optokopplers entnommen werden.

$$U_{Optokoppler5mA} = 1,15V$$

$$U_{Optokoppler10mA} = 1,2V$$

$$I_{Optokoppler} = 5/10mA$$

$$R_{1,2,3,4,5,6} = \frac{U}{I} = \frac{U_{GPIO} - U_{Optokoppler5mA}}{I_{Optokoppler}} = \frac{3,3V - 1,15V}{0,005A} = 430\Omega$$

$$R_{1,2,3,4,5,6} = \frac{U}{I} = \frac{U_{GPIO} - U_{Optokoppler10mA}}{I_{Optokoppler}} = \frac{3,3V - 1,2V}{0,010A} = 210\Omega$$

Da der Ausgangstrom durch den Eingangsstrom bestimmt wird, kann zur vereinfachten Berechnung durch die Kennlinie von Abbildung 4.14 ein Eingangsstrombereich angenommen werden. Dadurch kann mithilfe der Kennlinie von Abbildung 4.13 die abfallende Spannung am Optokoppler abgelesen werden. Mithilfe dieser Werte kann ein Widerstandsbereich berechnet werden, in dem der Widerstandswert liegen muss, damit der Ausgangstrom hoch genug ist. Der berechnete Widerstandsbereich von $R_{1,2,3,4,5,6}$ beträgt 210Ω - 430Ω . Es wird für diese Schaltung ein 390Ω Widerstand ausgewählt.

4.4.3.7 Verkabelung des Sensors mit dazugehörigen Elementen

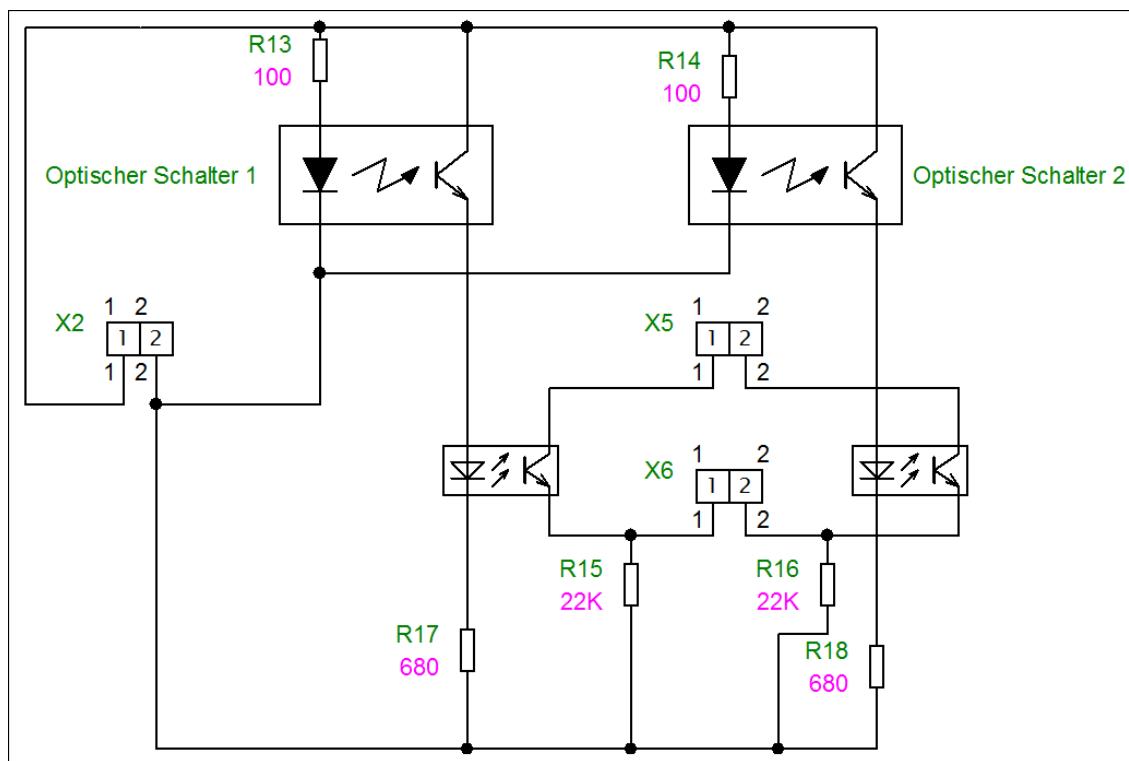


Abbildung 4.17: Sensoransteuerung

Die Sensoransteuerung besteht aus folgenden Komponenten:

- Den optischen Sensoren OPB732WZ.
- Der Klemme X2, X5, X6.
- Zwei Optokopplern, welche die Spannung der Sensoren von 5V auf ein 3,3V Potenzial bringen.
- Den Widerständen R13 und R14, welche einen Spannungsabfall erzeugen, damit der Sensor nicht überlastet wird.
- Den Widerständen R17 und R18, welche einen Spannungsabfall erzeugen, damit der Optokoppler nicht überlastet wird.
- Den Pulldown-Widerständen R15 und R16 welche ein floaten, also eine fehlerhafte Spannung an den Input-Pins des Raspberries verhindern sollen.

Genereller Ablauf:

1. Wenn der Sensor eine Reflektor-Fläche vor sich erkennt, schaltet der Phototransistor im Sensor durch.
2. Der durchgeschaltete Phototransistor ermöglicht einen Stromfluss von der Spannungsquelle über die Klemme X2, in den Optokoppler.
3. Dadurch schaltet der interne Phototransistor des Optokopplers durch.
4. Der durchgeschaltete Phototransistor ermöglicht einen Stromfluss von einem der 3,3V-Pins über die Klemme X5, in einen der GPIO-Pins, über die Klemme X6 des Raspberries.
5. Somit kann erkannt werden, welcher der Sensoren eine Reflektor-Fläche erkennt und weitere softwaretechnische Maßnahmen können erfolgen.

Berechnungen:

- R17 und R18:

Die Spannung, die abfallen muss und der Strom, der fließt, kann aus den Datenblättern des Sensors und des Optokopplers entnommen werden.

$$U_{Optokoppler5mA} = 1,15V$$

$$U_{Optokoppler10mA} = 1,2V$$

$$U_{CE(SAT)} = 0,4V$$

$$I_{Optokoppler} = 5/10mA$$

$$R_{17,18} = \frac{U}{I} = \frac{5V - U_{Optokoppler5mA} - U_{CE(SAT)}}{I_{Optokoppler}} = \frac{5V - 1,15V - 0,4V}{0,005A} = 690\Omega$$

$$R_{17,18} = \frac{U}{I} = \frac{5V - U_{Optokoppler10mA} - U_{CE(SAT)}}{I_{Optokoppler}} = \frac{5V - 1,2V - 0,4V}{0,01A} = 340\Omega$$

Da der Ausgangsstrom durch den Eingangsstrom bestimmt wird, kann zur vereinfachten Berechnung durch die Kennlinie von Abbildung 4.14 ein Eingangsstrombereich angenommen werden. Dadurch kann mithilfe der Kennlinie von Abbildung 4.13 die abfallende Spannung am Optokoppler abgelesen werden. Mithilfe dieser Werte kann ein Widerstandsbereich berechnet werden, in dem der Widerstandswert liegen muss, damit der Ausgangsstrom hoch genug ist. Der berechnete Widerstandsbereich von $R_{17,18}$ beträgt 340Ω - 690Ω . Es wird für diese Schaltung ein 680Ω Widerstand ausgewählt.

- R13 und R14:

Die Spannung die abfallen muss und der Strom der fließt, kann aus dem Datenblatt des Sensors entnommen werden.

$$U_{F50mA} = 1,45V$$

$$U_{F30mA} = 1,375V$$

$$I_F = 30/50mA$$

$$R_{13,14} = \frac{U}{I} = \frac{U_5V - U_{F50mA}}{I_F} = \frac{5V - 1,45V}{0,05A} = 72\Omega$$

$$R_{13,14} = \frac{U}{I} = \frac{U_5V - U_{F30mA}}{I_F} = \frac{5V - 1,375V}{0,03A} = 121\Omega$$

Da die optische Übertragungsstärke durch den Eingangsstrom bestimmt wird, kann zur vereinfachten Berechnung durch die Kennlinie von Abbildung 4.4 ein Eingangsstrombereich angenommen werden. Dadurch kann mithilfe der Kennlinie von Abbildung 4.3 die abfallende Spannung am Sensor abgelesen werden. Mithilfe dieser Werte kann ein Widerstandsbereich berechnet werden, in dem der Widerstandswert liegen muss, damit die optische Übertragungsstärke hoch genug ist. Der berechnete Widerstandsbereich von $R_{13,14}$ beträgt 72Ω - 121Ω . Es wird für diese Schaltung ein 100Ω Widerstand ausgewählt.

- R15 und R16:

Diese Widerstandswerte werden mit $22k\Omega$ angenommen.

4.4.3.8 Spannungsversorgung

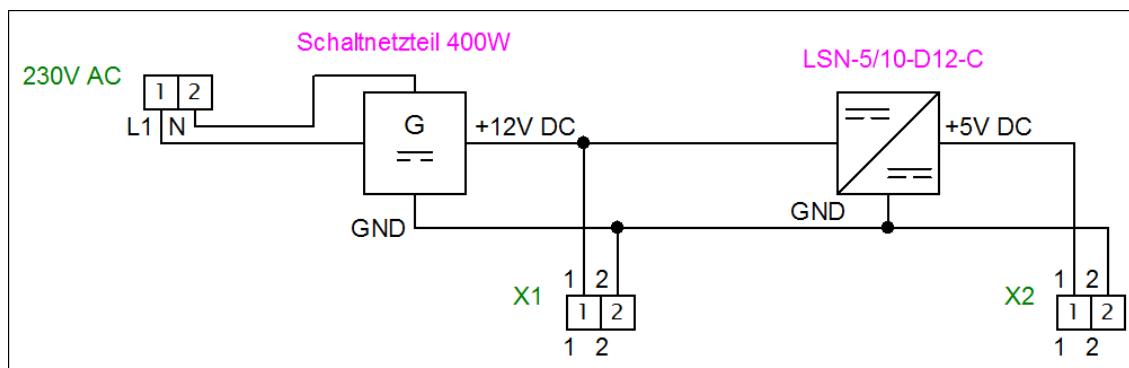


Abbildung 4.18: Spannungsversorgung Schaltplan

Die Spannungsversorgung erfolgt mittels eines 12V 400W Schaltnetzteils, welches aus 230V Wechselspannung 12V Gleichspannung erzeugt. Die 5V Versorgung erfolgt mittels eines 5V DC/DC Buck Down Converters. Die Spannungen werden an die Klemmen X1 und X2 gelegt.

4.4.3.9 Raspberry Verkabelung

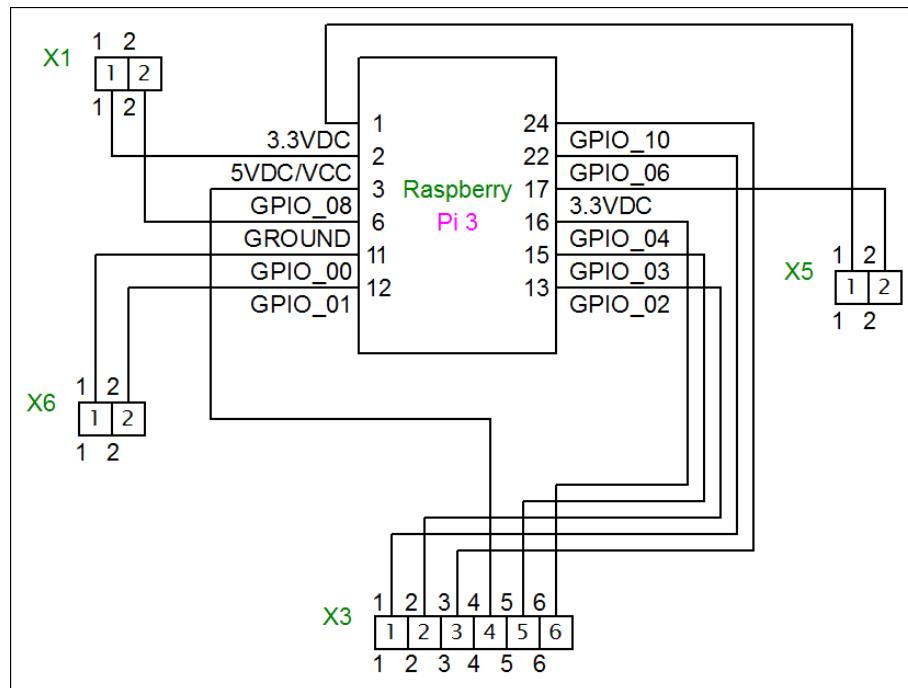


Abbildung 4.19: Raspberry Schaltplan

Der Schaltplan zeigt, welche Raspberry-Pins mit welcher Klemme verbunden sind. Die Pins werden mit den Klemmen X3, X5, X6 verbunden. Der Raspberry wird über die Klemme X2 versorgt.

4.4.3.10 Arduino Verkabelung

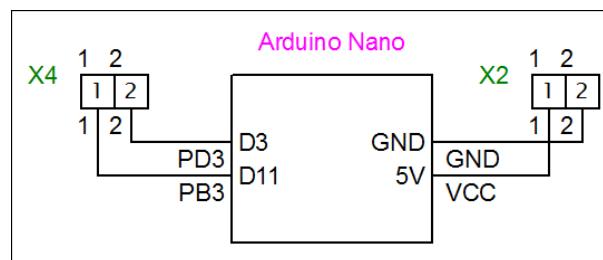


Abbildung 4.20: Arduino Schaltplan

Der Schaltplan zeigt, welche Arduino Nano-Pins mit welcher Klemme verbunden sind. Die Pins werden mit der Klemme X4 verbunden. Der Arduino wird über die Klemme X2 versorgt.

4.5 Mechanische Konstruktion

4.5.1 Motorhalterung Futterschüsseldrehplatte

Der Motor der Drehplatte wird an der Spitze der Welle befestigt. Der Motor wird über die Motoraufnahme mit der Decke der Anlage verschraubt. Damit wird der Motor auf der richtigen Höhe gehalten und ist gegen Drehung gesichert. Die genauen Höhe der Platte kann noch nicht bestimmt werden, da die Gesamthöhe der Anlage und damit die Deckenhöhe nicht festgelegt wurde. Die Position der Befestigung auf der Decke ist noch nicht festgelegt, da die genaue Position der Wände der Anlage noch nicht festgelegt wurden und die Werte von diesen Positionen abhängig sind. Die Motoraufnahme wird mit drei M5 Schrauben an der Decke befestigt. Der Motor wird mit zwei M10 Schrauben mit Muttern an der Motoraufnahme befestigt.

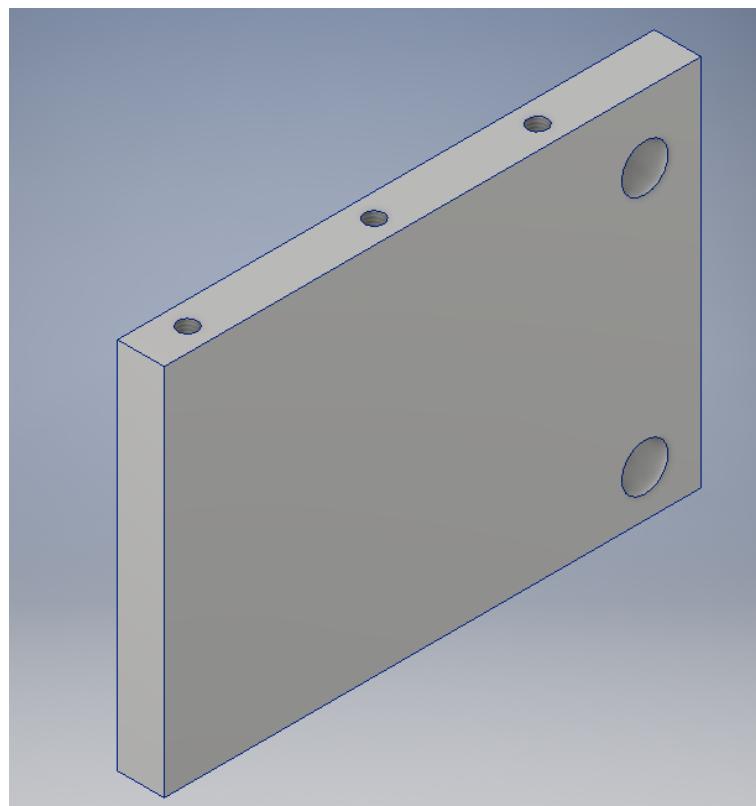


Abbildung 4.21: Motoraufnahme

4.5.2 Motorhalterung Förderband

Der Motor für das Förderband wird am Ende der Welle befestigt. Der Motor wird auf eine Aufnahme geschraubt und mithilfe eines Winkels an der rechten Stütze des Förderbandes befestigt. Die genaue Position variiert nach Förderbandlänge und muss nach Auslegung dimensioniert werden. Der Winkel wird mit drei M5 Schrauben an die Motoraufnahme geschraubt. Der Motor wird mit zwei M10 Schrauben mit Muttern an der Motoraufnahme befestigt. Der Winkel wird mit vier M6 Schrauben mit Muttern an der Förderbandstütze befestigt.

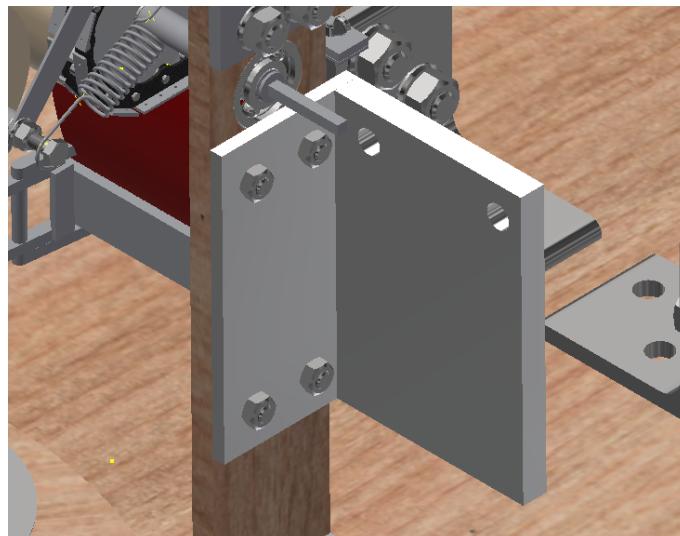
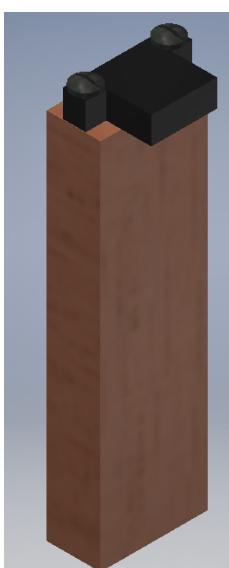


Abbildung 4.22: Motorhalterung Förderband

4.5.3 Sensorhalterung Futterschüsseldrehplatte



Der Sensor der Drehplatte wird so nah wie möglich auf Höhe der Drehplatte befestigt. Der Sensor wird mit einer Holzleiste direkt auf die Grundplatte geschraubt. Der Sensor wird auf die Holzleiste mit zwei M2,5 Holzschrauben geschraubt und die Leiste wird mit zwei M2,5 Holzschrauben von der Unterseite der Grundplatte auf deren Oberseite verschraubt. Die genaue Position auf der Grundplatte kann noch variieren. Die Höhe der Holzleiste kann sich je nach Höhe der Futtergeschüsseldrehplatte verändern.

Abbildung 4.23: Sensorhalterung
Drehplatte

4.5.4 Sensorhalterung Förderband

Der Sensor muss so nah wie möglich an der Klemme auf Höhe der Reflektorfläche befestigt werden. Der Sensor wird mit zwei Holzplatte an der linken Förderbandstütze befestigt. Der Sensor wird auf der ersten Holzleiste mit zwei M2,5 Holzschrauben befestigt. Die erste Holzleiste wird auf der zweiten Holzleiste mit zwei M2,5 Holzschrauben befestigt. Die zweite Holzleiste wird mit zwei M2,5 Schrauben und Muttern an die Förderbandstütze geschraubt. Die genaue Position kann variieren und muss in der Praxis ausgetestet werden. Die genauen Maße der Holzleisten und deren Befestigungen können noch variieren, je nach Größe der Stützen, der Befestigung des Förderbandes und weiteren Faktoren.



Abbildung 4.24: Sensorhalterung mit Position

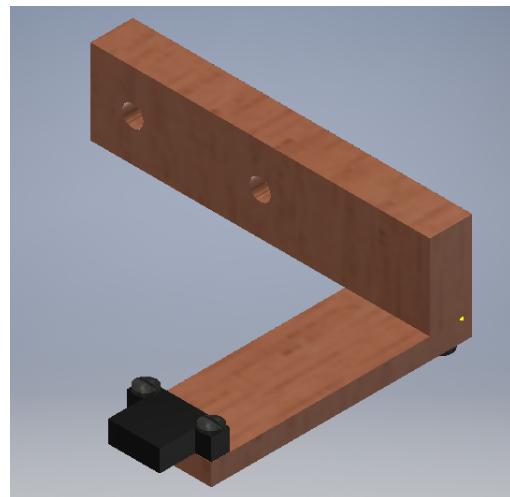


Abbildung 4.25: Sensorhalterung

4.5.5 Halterung Raspberry

Der Raspberry wird an die Unterseite des 7"Touchdisplays befestigt. Das Touchdisplay selber wird in eine Vertiefung in der Decke der Anlage geschraubt.

4.5.6 Lüfterhalterung für Raspberry

Der Lüfter wird am Ende eines Schachts, welcher sämtliche Elektronik, bis auf das 12V Schaltnetzteil, umschließt befestigt. Der Lüfter wird von der Innenseite an ein Loch in der Wand befestigt und von vier M4 Schrauben mit Muttern gehalten. Die genaue Befestigung des Kanals und des Lüfters variiert noch je nach verfügbarem Platz.

4.5.7 Förderband Stützen



Die Halterungen des Förderbandes sind in unserer Konstruktion überdimensioniert gezeichnet, da das Förderband je nach Benutzer eine gewünschte Länge annehmen kann. Die vier Halterungen aus Holz müssen daher standardmäßig eine Förderbandlänge von einem Meter tragen können. Die Halterungen sind mit zwei rechten Winkel an der Grundplatte befestigt. An der Oberseite der vorderen Halterungen ist eine extra Platte angeschraubt, die als Federbefestigung und Halterung der losen Walze dient. Für die Feder wird eine Schraube durch die Platte geschraubt, an der diese eingehängt wird. Für die Halterung der Walze wird ein Bolzen zwischen den beiden gegenüberliegenden Halterungen befestigt, an der zwei kleine Platten die Walzen an einer bestimmten Position halten. Unter der angeschraubten Platte wird ein Lager eingesetzt um die Welle zu lagern wie in Abbildung 4.27 zu sehen. An den beiden hinteren Halterungen wird nur ein Lager für die Welle gepresst wie in Abbildung 4.28 zu sehen.

Abbildung 4.26: Förderband Stütze links

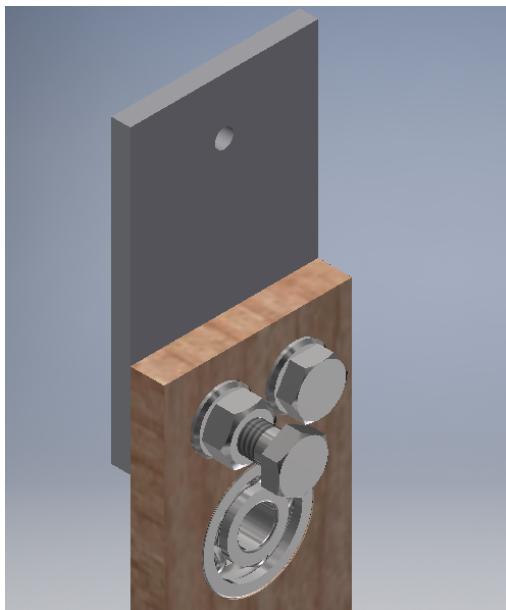


Abbildung 4.27: Stütze mit Platte



Abbildung 4.28: Stütze ohne Platte

4.6 Zusammenfassung und Selbstkritik

4.6.1 Relevante Meilensteine

01.06.2017 Projektstart

13.10.2017 Grundkonzept Freeze

08.12.2017 Konstruktion + Elektronik fertig

02.03.2018 Dokumentation fertig

4.6.2 Probleme

- **Konstruktion:** Die ersten Konstruktionsvarianten stellten sich als viel zu aufwendig heraus, weshalb die geplante Zeit bis zum Grundkonzept Freeze nicht ausreichte und sich der Grundkonzept Freeze um Monate verschob. Des weiteren haben sich Probleme in der Elektronik ergeben, weshalb sich das Konzept der Elektronik auch nach dem Grundkonzept Freeze verändert hat.
- **Elektronik:** Da ich versucht habe, so viel wie möglich von der Ansteuerung selbst zu machen, hat ein Großteil der Schaltung nicht funktioniert und musste verbessert oder ersetzt werden.
- **Dokumentation:** Das Einarbeiten in Latex hat mehr Zeit in Anspruch genommen als erwartet, weshalb sich das Ende der Dokumentation verzögerte.

4.6.3 Zukünftige Verbesserungen

- **Konstruktion:** Bei erneuter Arbeit an einem Projekt, sollten eher einfache Varianten verwendet werden, die gegebenenfalls noch verbessert und ergänzt werden können, als direkt mit einer schwierigen Variante zu beginnen, welche zu viel Zeit in Anspruch nimmt.
- **Elektronik:** Bei der Elektronik muss ich in Zukunft versuchen eher bereits fertige Module für Schaltungen zu verwenden, als diese selbst zu entwerfen. Fertige Module sind meistens günstiger zu erwerben, als die einzelnen Bauteile und verfügen meist über zusätzliche Funktionen, welche die eigene Schaltung nicht bieten könnte. Als Beispiel wird die Motoransteuerung hergenommen. Die fertige Motoransteuerung ist günstiger als die selbst gezeichnete H-Brücke und verfügt zusätzlich über eine Drehzahlregelung mittels PWM-Signal, internen Sicherungen gegen Kurzschlüsse und kann die Ströme der Motoren messen.

- Dokumentation:** Ich würde in Zukunft die Dokumentation parallel zur generellen Arbeit im Projekt beginnen, um eine Überforderung zum Schluss des Projektes zu vermeiden.

4.6.4 Tatsächlicher Zeitpunkt der erreichten Meilensteine

01.06.2017 Projektstart

10.03.2017 Grundkonzept Freeze

24.03.2018 Konstruktion + Elektronik fertig

27.03.2018 Dokumentation fertig

4.7 Stückliste

Name	ID	Stückzahl	Anmerkung	Link
Optek Optischer Näherungsschalter	OPB732WZ	2	-	https://at.rs-online.com/web/p/products/9087113/?gclid=Cj0KCQjw6NjNBRDKARIaFn3NMq0RvGj3r_5q7xtFIs2eGGmyLAEnp-a446vnWDjUisFB9L3bRjBDXuaAqj1EALw_wCB&cm_mmc=AT-PLA-DS3A-_google-_PLA_AT_DE_Automation-_Sensoren_Und_Messwandler_-_PRODUCT+GROUP&matchtype=&grossPrice=Y&gcisrc=aw.de
Optokoppler	SFH6916	2	4 Optokoppler pro Bauteil	https://www.neuhold-elektronik.at/catshop/product_info.php?products_id=4181
Motorsteuerung	VNH2SP30	1	Fertiges Modul aus 2 ICs und zusätzlicher Beschaltung	https://www.ebay.at/item/30A-VNH2SP30-Dual-Stepper-Motor-Driver-Monster-Moto-Shield-Module-Motortrieber/122543212784?hash=item1c882508f0:g:QGkAAOSwsy9amWPw
Gleichstrom Motor	FD MY2007 U222 60x	2	Schneckengetriebe	https://www.neuhold-elektronik.at/catshop/product_info.php?cPath=96_99&products_id=5522
DC/DC Wandler 5V	MURATA LSN-5/10-D12-C 5 V-/ 10 A	1	12V zu 5V	https://www.neuhold-elektronik.at/catshop/product_info.php?cPath=222_361&products_id=6778
Schaltnetzteil 12V	-	1	400W	https://www.amazon.de/dp/B0111MEUWY/ref=twister_B0111MEF4?encoding=UTF8&th=1
Lüfter 12V	EE80251S2-0000-999	1	80x80x25mm	https://www.conrad.at/de/axiallufter-12-vdc-6286-mh-1-x-b-x-h-80-x-80-x-25-mm-sunon-ee80251s2-0000-999-323905.html
Widerstand	-	6	Kohleschicht 390Ω	https://www.conrad.at/de/kohleschicht-widerstand-390-axial-bdrahtet-0204-01-w-5-tru-components-1-st-1557169.html
Widerstand	-	2	Kohleschicht 150Ω	https://www.conrad.at/de/kohleschicht-widerstand-150-axial-bdrahtet-0204-01-w-5-1-st-400157.html
Widerstand	-	2	Kohleschicht 10Ω	https://www.conrad.at/de/kohleschicht-widerstand-10-axial-bdrahtet-0207-025-w-5-yageo-cfr-25jt-52-10r-1-st-1417641.html
Widerstand	-	2	Kohleschicht 680Ω	https://www.conrad.at/de/kohleschicht-widerstand-680-axial-bdrahtet-0207-025-w-5-yageo-cfr-25jt-52-680r-1-st-1417677.html
Widerstand	-	8	Kohleschicht 22kΩ	https://www.conrad.at/de/kohleschicht-widerstand-22-k-axial-bdrahtet-0207-025-w-5-yageo-cfr-25jt-52-22k-1-st-1417666.html
Arduino Nano	-	1	Atmega328P zur PWM generierung	https://www.amazon.de/AptoFun-Org-ATmega328P-FT232RL-Development-kompatibel/dp/B014TE52RS/ref=sr_1_1_sspa?ie=UTF8&qid=1521579402&sr=8-1-spons&keywords=arduino+nano&psc=1

Tabelle 4.2: Stückliste

4.8 Quellenverzeichnis

Verwendung für Kapitel	Datum	URL
4.3.1.1	22.10.2017	http://tremba.de/orientierung.php
4.3.1.1	22.10.2017	http://tremba.de/zylindermagnete/zylindermagnete.php
4.3.1.1	22.10.2017	http://tremba.de/zylindermagnete/db-zylindermagnete-ZMF-2551z.002.pdf
4.3.1.1	22.10.2017	http://tremba.de/zylindermagnete/db-zylindermagnet-ZMF-1130d.002.pdf
4.3.1.1	22.10.2017	http://tremba.de/zylindermagnete/db-zylindermagnet-ZMF-1130d.002.pdf
4.3.1.1	22.10.2017	http://tremba.de/kurzhubmagnete/db-hubmagnete-KHM-1113.001.pdf
4.3.1.1	22.10.2017	http://tremba.de/hubmagnete/db-hubmagnete-HMF-3830d-15.002.pdf
4.3.2, 4.4.2, 4.4.3.7	25.10.2017	http://at.rs-online.com/web/p/products/9087113/?grossPrice=Y&cm_mmc=AT-PLA-DS3A--google--PLA_AT_DE_Automation_-_Sensoren_Und_Messwandler--PRODUCT+GROUP&matchtype=&gclid=Cj0KCQjw6NjNBRDKARIIsAFn3NMq0RvGj3r_5q7xtFIs2eGGmyLAENp-a446vhWDjUIsFB9L3bRjBDXUaAqj1EALw_wcB&gclsrc=aw.ds
4.3.2, 4.4.2, 4.4.3.7	25.10.2017	http://docs-europe.electrocomponents.com/webdocs/14a6/0900766b814a6875.pdf
4.3.2, 4.4.2, 4.4.3.7	25.10.2017	http://www.mouser.com/ds/2/414/OP265-266-45966.pdf
4.4.3.1	01.12.2017	http://forum.arduino.cc/index.php?topic=430436.0
4.4.3.1	01.12.2017	http://www.bristolwatch.com/ele/h_bridge.htm
4.4.3.1	01.12.2017	https://www.conrad.at/de/mosfet-vishay-irf9630pbf-1-p-kanal-74-w-to-220-162541.html?insert=U3&gclid=Cj0KCQiAmITRBRC5ARIsAE0Zmr5pGS-Z97ve6qOgyJbyrJ1jLlzOkaKVUCalomia_ZBUXX-yv1mTz2gaAhY5EALw_wcB
4.4.3.1	01.12.2017	https://www.conrad.at/de/mosfet-infineon-technologies-irf630n-1-n-kanal-82-w-to-263-3-162421.html
4.4.3.1	14.02.2018	http://www.produktinfo.conrad.com/datenblaetter/150000-174999/162421-da-01-en-IRF_630_N.pdf
4.4.3.1	14.02.2018	http://www.produktinfo.conrad.com/datenblaetter/150000-174999/162541-da-01-en-TRANSISTOR_HEXFET_IRF9630PBF_TO_220_VIS.pdf
4.4.3.2, 4.4.3.6	02.03.2018	https://www.ebay.at/itm/30A-VNH2SP30-Dual-Stepper-Motor-Driver-Monster-Moto-Shield-Module-Motortreiber/122543212784?hash=item1c882508f0:g:QGkAAOSwsy9amWPw
4.4.3.2, 4.4.3.6	02.03.2018	http://www.st.com/content/ccc/resource/technical/document/datasheet/group2/66/b8/f5/2c/9a/66/41/c7/CD00043711/files/CD00043711.pdf;jcr:content/translations/en.CD00043711.pdf
4.4.3.2, 4.4.3.6	02.03.2018	http://www.instructables.com/id/Monster-Motor-Shield-VNH2SP30/
4.4.3.3, 4.4.3.10	04.03.2018	http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-avr-Microcontroller-ATmega328-328P_Datasheet.pdf
4.4.3.4, 4.4.3.6, 4.4.3.7	11.12.2017	https://www.neuhold-elektronik.at/datenblatt/N6465.pdf
4.4.3.8	12.03.2018	https://www.neuhold-elektronik.at/catshop/product_info.php?cPath=222_361&products_id=67786

Tabelle 4.3: Quellen

Anhang

ANHANG A

Abbildungsverzeichnis

1.1	Kommunikationen innerhalb des Projekts	1
1.2	Angular Struktur	8
1.3	Angular Databinding	10
1.4	Startseite	13
1.5	Fütterungszeiten	13
1.6	Geräteinformationen	14
1.7	Update	14
2.1	Abhängigkeiten	38
2.2	Pin Numbering Sheme	39
3.2	Magazin Modellaufbau von der Seite	52
3.3	Magazin Modellaufbau von Oben	52
3.4	Magazin Auszug	53
3.5	Magazin Auszug Mitte	53
3.1	Magazin Modellaufbau von Vorne	53
3.6	Schneidebereit	54
3.7	Schnitt	55
3.9	Ausquetschen Mitte	55
3.10	Ausquetschen Ende	55
3.8	Ausquetschen Beginn	56
3.12	Bolzen drinnen	56
3.13	Bolzen entfernen	56
3.14	Klappe öffnen	57
3.15	Fertiger Auswurf	57
3.11	Auswurf Beginn	57
3.17	Foerderband	58
3.18	Kettenglied	58
3.19	Walze	59

3.20 Scharnier	59
3.16 Einhänge Futterschüssel	60
3.22 Halterung	61
3.23 Entleerung Anfang	61
3.24 Entleerung nach 5min	62
3.25 Entleerung nach 10min	62
3.26 Einlegen	63
3.27 Anfangsschnitt	63
3.28 Endschnitt	64
3.29 Schneidemittel	65
3.30 Anfangsschnitt 2.Art	65
3.31 Mittelschnitt 2.Art	65
3.32 Endschnitt 2.Art	65
3.21 Futterplatte	66
3.34 Klemmen Probe	67
3.35 Klemme Tag 1	67
3.36 Klemme Tag 2	67
3.37 Klemme Tag 3	67
3.38 Klemme Tag 4	68
3.39 Klemme Tag 5	68
3.40 Fütterungsexperiment Ende	69
3.41 Futterkonsistenz	69
3.42 Futter in der Schüssel	69
3.33 3D-Klemme	70
3.43 Einfache Klemme	70
3.44 Hebel Klemme	71
3.45 Gummiband Klemme	72
3.46 Drehplatte	72
3.47 Platte Zylinder	73
3.48 Einschüsselplatte	74
3.49 Futtermagazin Horizontal	74
3.50 Futtermagazin Vertikal	75
3.51 Drehplatte Inventor	75
3.52 Futterschüssel Inventor	75
3.54 Kette Inventor	76
3.55 Kettenglied Inventor	77
3.56 Kettenrad Inventor	77
3.53 Drehplattenwelle Inventor	78
3.57 Förderbandwelle Inventor	78
3.58 Walze Inventor	79

3.59	Feder Inventor	79
3.60	Welle für die Walze Inventor	79
3.62	Pressvorgang	80
3.63	Walzenhalterung	80
3.64	Aufhängung der Futterpackung	81
3.65	Öffnen der Hebel-Klemme	81
3.66	Gesamtansicht der Konstruktion	81
3.67	Skizze der Klemme	82
3.68	Querschnitt der Klemme	82
3.61	Hebel Klemme	83
3.70	Skizze des Förderbandes	84
3.71	Reibungskraft	84
3.72	Federabmessung	85
3.73	Walze und Federweg	85
3.69	Kritischer Punkt an der Welle	86
3.75	Federauslenkungs Simulation	87
3.74	Hebel-Klemme Simulation	88
4.1	Hubmagnet	90
4.2	Sensor	91
4.3	Diodenspannung zu Diodenstrom	92
4.4	Optische Übertragungs- stärke zu Diodenstrom	92
4.5	Motor	93
4.6	Lüfter	94
4.7	P-Kanal	96
4.8	N-Kanal	96
4.9	H-Bridge	96
4.10	Motorsteuerung	97
4.11	PWM-Signal	98
4.12	PWM-Signal	102
4.13	Diodenstrom zu Diodenspannung	103
4.14	Transistorstrom zu Transistorspannung	103
4.15	Blockschaltbild	104
4.16	Motoransteuerung	105
4.17	Sensoransteuerung	107
4.18	Spannungsversorgung Schaltplan	109
4.19	Raspberry Schaltplan	110
4.20	Arduino Schaltplan	110
4.21	Motoraufnahme	111
4.22	Motorhalterung Förderband	112

4.23 Sensorhalterung Drehplatte	113
4.24 Sensorhalterung mit Position	113
4.25 Sensorhalterung	113
4.26 Förderband Stütze links	114
4.27 Stütze mit Platte	115
4.28 Stütze ohne Platte	115

ANHANG B

Tabellenverzeichnis

1.1	Übertragungsprotokoll Webserver - Webclient	23
1.2	Übertragungsprotokoll Webserver - Java	28
2.1	Belegung der GPIO-Pins	40
2.2	Signalverlauf	40
2.3	Übertragungsprotokoll	43
4.1	H-Brückenzustände	95
4.2	Stückliste	117
4.3	Quellen	118

ANHANG C

Listings

1.1	Zuweisen von Design an HTML	5
1.2	JSON Beispiel	6
1.3	Verbinden mit dem DBMS	7
1.4	Auswählen der Datenbank	7
1.5	Auswählen der Collection	8
1.6	Auslesen aller Datensätze mit einem Identifier	8
1.7	Überschreiben eines Datensatzes mit einem Identifier	8
1.8	app.module (Auszug) Importieren von ng-bootstrap	15
1.9	Schriftgrößen entsprechend der Bildschirmbreite	15
1.10	HTML Schalter	17
1.11	CSS Schalter	17
1.12	Routes Definition	19
1.13	Weiterleitung mittels HTML Attributes	20
1.14	Weiterleiten mittels TypeScript	20
1.15	Zeiten Parser	20
1.16	Animieren des Knopfs	21
1.17	Server Singleton	24
1.18	Middlewares	25
1.19	Sprachenauswahl	25
1.20	Login Methode	26
1.21	Update Methode	27
1.22	Verbindung mit der Datenbank	27
1.23	WLAN Konfigurationsbeispiel	29
2.1	Mit Mongodb verbinden	37
2.2	Collection auswählen	37
2.3	Anzahl der Collections zählen	37
2.4	Nach Dokument suchen	37

2.5 Ein Dokument hinzufügen	37
2.6 Ein Dokument updaten	37
2.7 Verbindung zur Datenbank trennen	37
2.8 Konkretes Beispiel: Dokument suchen	37
2.9 GPIO-Controller erstellen	41
2.10 Zugriff auf einen Pin als Input	41
2.11 Zugriff auf einen Pin als Output	41
2.12 Pinzustand abfragen	41
2.13 Pinzustand verändern	42
2.14 Controller herunterfahren	42
2.15 Error setzen	44
2.16 JSON-Object mit Errors und Warnings	45
2.17 SwingWorker Klasse erstellen	46
4.1 µC-Programm	100

ANHANG D

Abkürzungsverzeichnis

HTTP HyperText Transfer Protocol	2
TCP Transmission Control Protocol	2
URI Uniform Resource Identifier	3
HTTPS HyperText Transfer Protocol Secure	3
npm Node Package Manager	4
HTML HyperText Markup Language	5
DOM Document Object Model	5
CSS Cascading Style Sheets	5
JSON JavaScript Object Notation	6
JWT JSON Web Token	7
DBMS Database Management System	7

API Application Programming Interface	9
CLI Command Line Interface	11
CDN Content Delivery Network	11
WLAN Wireless Local Area Network	29
GUI Graphical User Interface	31
GPIO General Purpose Input/Output	31
pi4j Pi for Java	33
JNI Java Native Interface	38
JVM Java Virtual Machine	38
SPI Serial Peripheral Interface	39
UART Universal Asynchronous Receiver Transmitter	39
TCP/IP Transmission Control Protocol Internet Protocol	42
EDT Event Dispatch Thread	47
URL Uniform Resource Locator	
IP Internet Protocol	
μC Mikrocontroller	94

PWM Pulse Width Modulation	97
---	----

ANHANG E

Literatur

Lewis Loflin. *H-Bridge Motor Control with Power MOSFETS.*

http://www.bristolwatch.com/ele/h_bridge.htm.

A. Sinha. *Stackoverflow*. Okt. 2015.

<https://stackoverflow.com/questions/33085493/hash-a-password-with-sha-512-in-java>.