

# **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



## **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

@TODO



## **Abstract**

WULF!!!

## **Zusammenfassung**

WOLF!!!



## **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<sup>A</sup>T<sub>E</sub>X 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.



## 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

Mechianik

**Betreuer:**

Otto

---

## Inhaltsverzeichnis

---

<b>1 Webserver und Client</b>	<b>1</b>
1.1 Begriffserklärungen . . . . .	1
1.1.1 Server . . . . .	1
1.1.2 Client . . . . .	1
1.2 Anforderungen . . . . .	1
1.2.1 Webserver . . . . .	1
1.2.2 Client . . . . .	1
1.3 Voruntersuchung . . . . .	2
1.3.1 Typescript . . . . .	2
1.3.2 Node.js . . . . .	2
1.3.3 Angular 2/4 . . . . .	2
1.3.3.1 Modules . . . . .	2
1.3.3.2 Components . . . . .	3
1.3.3.3 Templates . . . . .	3
1.3.3.4 Data binding . . . . .	3
1.3.3.5 Services . . . . .	3
1.3.4 Bootstrap . . . . .	3
1.4 Umsetzung . . . . .	4
1.4.1 Client . . . . .	4
1.4.1.1 Design . . . . .	4
1.4.1.2 Funktion . . . . .	5
1.4.2 Server . . . . .	5
1.4.2.1 Funktion . . . . .	5
1.4.2.2 Mongodb . . . . .	5
1.4.2.3 Kommunikation mit dem Java Programm . . . . .	5
1.5 Zusammenfassung und Verbesserungsmöglichkeiten . . . . .	5
<b>2 Java-Programm</b>	<b>7</b>
2.1 Anforderungen . . . . .	7
2.1.1 Programm . . . . .	7
2.1.2 Design - Benutzerinterface . . . . .	7
2.1.3 Externe Steuerung . . . . .	7
2.2 Voruntersuchung . . . . .	8
2.2.1 Wieso Java und nicht C? . . . . .	8
2.2.2 Wieso das Raspberry Pi 3 Model B? . . . . .	8

2.2.3	Auswahl eines Touchdisplays . . . . .	8
2.2.4	Wieso pi4j? . . . . .	8
2.2.5	Wieso Mongodb? . . . . .	8
2.2.5.1	Vorteil gegenüber Daten in Datei speichern . . . . .	9
2.2.5.2	Vergleich mit anderen Datenbanken . . . . .	9
2.2.6	Kommunikation mit der Web-Applikation . . . . .	9
2.3	Umsetzung . . . . .	10
2.3.1	Mongodb . . . . .	10
2.3.1.1	Allgemeines . . . . .	10
2.3.1.2	Datenbankmanagementsystem DBS . . . . .	11
2.3.1.3	Singleton . . . . .	11
2.3.1.4	Code Beispiele . . . . .	11
2.3.2	pi4j . . . . .	12
2.3.2.1	Allgemeines . . . . .	12
2.3.2.2	Pin Numbering Sheme . . . . .	13
2.3.2.3	Gewählte Pin Belegung . . . . .	13
2.3.2.4	Singleton . . . . .	14
2.3.2.5	Code Beispiele . . . . .	14
2.3.3	Server-Client-Kommunikation . . . . .	16
2.3.3.1	Server . . . . .	16
2.3.3.2	Übertragungsprotokoll . . . . .	16
2.3.3.3	Response-GET . . . . .	17
2.3.3.4	Response-PUT . . . . .	17
2.3.3.5	Verarbeiten des Requests . . . . .	17
2.3.4	Errors und Warnungsverarbeitung . . . . .	17
2.3.4.1	Allgemeines . . . . .	17
2.3.4.2	Errors und Warnungen aktivieren/deaktivieren . . . . .	18
2.3.4.3	Json-Array . . . . .	18
2.3.4.4	Listenmodel . . . . .	19
2.3.4.5	Liste . . . . .	19
2.3.5	SwingWorker . . . . .	20
2.3.5.1	EDT . . . . .	20
2.3.5.2	TimeUnit . . . . .	20
2.3.5.3	Methoden des SwingWorkers . . . . .	20
2.3.6	GUI-Fenster . . . . .	22
2.3.6.1	Java-Programm im Autostart . . . . .	22
2.3.6.2	MainWindow . . . . .	23
2.3.6.3	TimeManagement . . . . .	26
2.3.6.4	CreateUser . . . . .	29
2.3.6.5	ManualControl . . . . .	31
2.3.6.6	Positionsinformation . . . . .	33
2.3.6.7	SystemInfo . . . . .	35
2.3.6.8	Update . . . . .	36

2.4 Zusammenfassung/Verbesserungsmöglichkeiten . . . . .	38
2.4.1 Probleme mit Mongodb am Raspberry . . . . .	38
2.4.2 Probleme mit pi4j -> Snapshotversion verwendet - ansonsten werden pins nicht erkannt . . . . .	38
2.4.3 GUI auf "Touchscreen-Designändern" . . . . .	38
2.4.4 Besser Benutzerverwaltung - Mehrere Benutzer anlegen . . . . .	38
2.4.5 Selbst erstellbare Vorlagen in denen Zeiten gespeichert werden . . . . .	38
<b>3 Mechanik</b>	<b>39</b>
3.1 Einleitung . . . . .	39
3.2 Aufgabenstellung und Zielsetzung . . . . .	39
3.3 Problematik . . . . .	39
3.3.1 Problematik des automatisierten Aufschneidens . . . . .	39
3.3.2 Problematik der Dichtheit bei Klemmen . . . . .	39
3.3.3 Problematik bei Entleerung der Verpackung . . . . .	40
3.3.4 Problematik vom Geruch . . . . .	40
3.3.5 Problematik von der Reinigung nach dem Urlaub . . . . .	40
3.3.6 Problematik bei einfrieren des Futters . . . . .	40
3.4 Konzepte . . . . .	40
3.4.1 Variante 1: Automatisiertes Aufschneiden . . . . .	40
3.4.1.1 Übersicht der Prozessschritte . . . . .	40
3.4.1.2 Füllen des Futtermagazins . . . . .	41
3.4.1.3 Führen zur Schneidplatte . . . . .	42
3.4.1.4 Schnitt . . . . .	44
3.4.1.5 Pressen . . . . .	44
3.4.1.6 Entsorgen . . . . .	45
3.4.1.7 Füttern . . . . .	46
3.4.2 Variante 2: Vor aufgeschnittene Packung . . . . .	46
3.4.2.1 Förderband und Kettenglieder . . . . .	47
3.4.2.2 Walze . . . . .	47
3.4.2.3 Futterplatte . . . . .	48
3.4.3 Variante 3: Gefrorenes Futter . . . . .	48
3.5 Aufbauten und Tests . . . . .	48
3.5.1 Fütterungsexperiment . . . . .	49
3.5.1.1 Schlussfolgerung des Fütterungsexperimentes . . . . .	50
3.5.2 Schneideversuch 1.Art der 1.Variante . . . . .	50
3.5.2.1 Schlussfolgerung des Schneideversuchs 1.Art der 1.Variante . . . . .	51
3.5.3 Schneideversuch 2.Art der 1.Variante . . . . .	51
3.5.3.1 Schlussfolgerung des Schneideversuchs 2.Art der 1.Variante . . . . .	52
3.5.4 Dichtheitsexperiment der Hebelklemme . . . . .	53
3.6 Vergleich der Varianten . . . . .	55
3.6.1 Klemmen . . . . .	55
3.6.1.1 Einfache Klemme . . . . .	55
3.6.1.2 Hebel Klemme . . . . .	55

3.6.1.3	Gummiband Klemme . . . . .	55
3.6.2	Klemmen Wahlvariante . . . . .	56
3.6.3	Futterschüsseln . . . . .	56
3.6.3.1	Drehfutterplatte . . . . .	56
3.6.3.2	Futterplatte Zylinder . . . . .	56
3.6.3.3	Platte mit einer Schüssel . . . . .	57
3.6.4	Futterschüssel Wahlvariante . . . . .	57
3.6.5	Futtermagazine . . . . .	57
3.6.5.1	Futtermagazin Horizontal . . . . .	57
3.6.5.2	Futtermagazin Vertikal . . . . .	58
3.7	Konstruktion der Wahlvariante und Details . . . . .	58
3.7.1	Drehplatte . . . . .	58
3.7.2	Förderband, Kettenglied und Kettenrad . . . . .	59
3.7.3	Walze . . . . .	61
3.7.4	Hebelklemme . . . . .	61
3.8	Berechnung und Dimensionierung . . . . .	61
3.9	Simulation . . . . .	61
3.10	Bedienung und Wartung . . . . .	61
3.11	Selbstkritische Analyse und Ausblick . . . . .	62
<b>4</b>	<b>Elektronik und Mechanik</b>	<b>63</b>
4.1	Anforderung Elektronik . . . . .	63
4.1.1	Aktoren . . . . .	63
4.1.2	Sensoren . . . . .	63
4.1.3	Ansteuerungen . . . . .	63
<b>A</b>	<b>Abbildungsverzeichnis</b>	<b>67</b>
<b>B</b>	<b>Tabellenverzeichnis</b>	<b>69</b>
<b>C</b>	<b>Listings</b>	<b>71</b>
<b>D</b>	<b>Abkürzungsverzeichnis</b>	<b>73</b>
<b>E</b>	<b>Literaturverzeichnis</b>	<b>75</b>

# KAPITEL 1

---

## Webserver und Client

---

### 1.1 Begriffserklärungen

#### 1.1.1 Server

Ein Computer oder Programm, der oder das Zugriff auf eine Resource oder einen Dienst in einem Netzwerk ermöglicht

#### 1.1.2 Client

Ein Computer oder Programm, der oder 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 zu sehen sein und eine allgemeine Übersicht. Über eine Navigationsleiste sollen die weiteren Seiten erreichbar sein:

- Fütterungszeiten
- Positionsinfo
- Geräteinfo
- Update

## 1.3 Voruntersuchung

### 1.3.1 Typescript

Typescript ist eine Weiterentwicklung der Sprache Javascript, die strenge Datentypen hat. Typescript muss von einem Transpiler (=Übersetzer) in Javascript übersetzt werden. Javascript kann direkt von jedem herkömmlichen Browser ausgeführt werden.

### 1.3.2 Node.js

Node.js ist eine Laufzeitumgebung, die es ermöglicht, dass Javascript direkt auf einem Rechner ausgeführt werden kann.

### 1.3.3 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 Komponenten. Die grobe Struktur wird in der Abbildung 1.1 dargestellt.

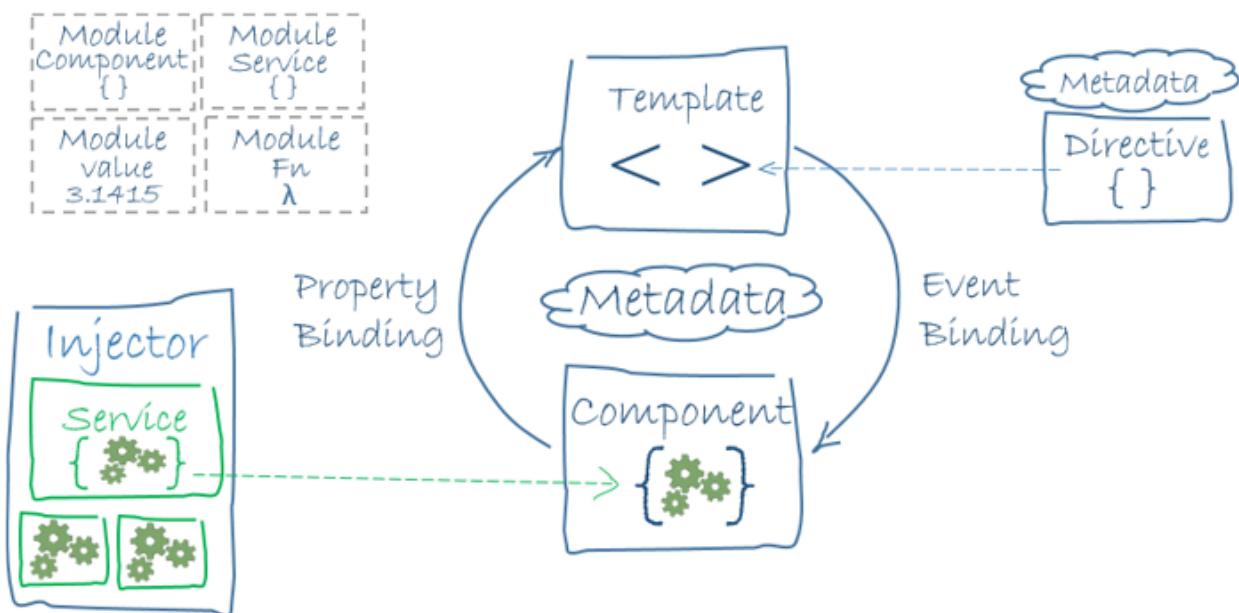


Abbildung 1.1: Angular Struktur

#### 1.3.3.1 Modules

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

- Components
- Services

Angular selbst besteht ebenfalls aus *Modules*, den sogenannten Libraries. Der Name jedes *Modules* beginnt mit @angular z.B. @angular/core.

### 1.3.3.2 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.

### 1.3.3.3 Templates

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

#### \*ngFor

Stellt ein HTML Element so oft dar, wie Elemente in einem Array vorhanden sind.

Listing 1.1: \*ngFor Beispiel

```
1 <a *ngFor="let lang of languages" href="{{lang.href}}>{{lang.text}}</a>
```

#### \*ngIf

{{variable}}

(click)

[variable]

<app-route>

### 1.3.3.4 Data binding

### 1.3.3.5 Services

## 1.3.4 Bootstrap

Bootstrap

## 1.4 Umsetzung

### 1.4.1 Client

#### 1.4.1.1 Design

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 rechten Seite sind die aktiven Fütterungszeiten aufgelistet.



Abbildung 1.2: Startseite

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 1.3. Der Button 'Speichern' wird deaktiviert, sobald eine Zeit ungültig eingegeben wurde, oder wenn die Zeiten nicht in aufsteigender Reihenfolge sortiert sind.

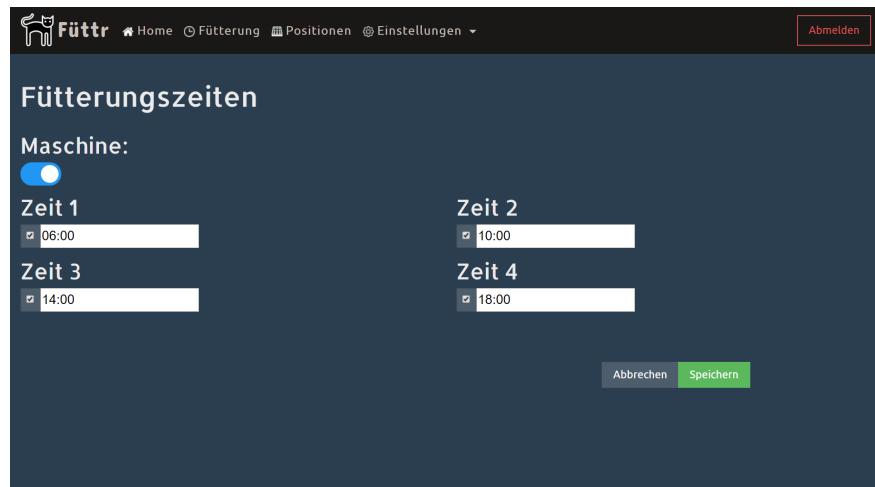


Abbildung 1.3: Fütterungszeiten

Auf der Geräteinformations-Seite werden die wichtigsten Daten über das Gerät ange-



zeigt. diese Daten sind:

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

Die Seriennummer ist eindeutig und wird von einem Server zugeteilt. 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 Herunterfahrens-Button klickt, der sich auf der linken Seite ganz unten befindet, wird er gewarnt, dass die Maschine nur mehr über das Aus- und wieder Einstecken des Netzteils startbar ist.

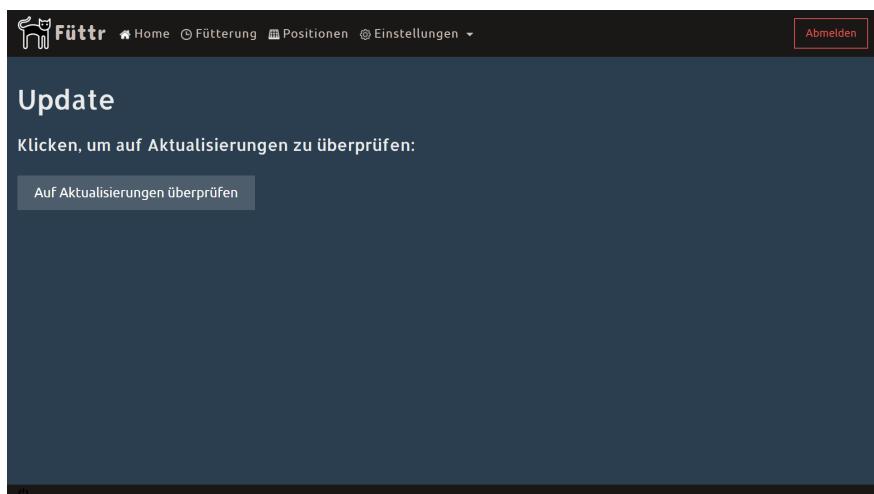


Abbildung 1.5: Update

#### 1.4.1.2 Funktion

#### 1.4.2 Server

##### 1.4.2.1 Funktion

##### 1.4.2.2 Mongodb

##### 1.4.2.3 Kommunikation mit dem Java Programm

## 1.5 Zusammenfassung und Verbesserungsmöglichkeiten



# KAPITEL 2

---

## Java-Programm

---

### 2.1 Anforderungen

#### 2.1.1 Programm

Die Hauptaufgabe des Programms ist es dem Benutzer eine Möglichkeit zum Steuern der Katzenfütterungsanlage zu Verfügung zu stellen. Weiters soll das Programm die Motoren steuern und die Sensoren in der Anlage auswerten können. Für diese Aufgabe sollen die IO-Pins am Raspberry verwendet werden.

#### 2.1.2 Design - Benutzerinterface

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. Der Benutzer soll mit Hilfe eines kleinen Touchdisplays die Möglichkeit haben die Anlage zu steuern. Deswegen muss darauf geachtet werden, dass alle GUI-Fenster sinnvoll per Touch-Gesten verwenbar sind.

#### 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 (Server) mit der Web-Applikation (Client) 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. Vorteile von C:

- 1 Echtzeitfähige Steuerung der Motoren und Sensors
- 2 Hardwarenahe Programmierung für die Pins

Vorteile von Java:

- 1 Erstellen einer GUI ist einfacher
- 2 Implementieren eines Servers ist einfacher

Nach dem Gegenüberstellen der Vorteile wurde Java als Programmiersprache gewählt.

### 2.2.2 Wieso das Raspberry Pi 3 Model B?

Schon zu Beginn der Diplomarbeit war klar, dass mit deinem Raspberry gearbeitet werden soll. Nun musste entschieden werden welches Model verwendet werden soll. Wir haben das Raspberry Pi 3 Model B aufgrund folgender technischer Daten gewählt:

- 1 Rechenleistung
- 2 Anzahl der GPIO-Pins
- 3 WLAN-Fähigkeit

### 2.2.3 Auswahl eines Touchdisplays

Das Display muss folgendet Anforderungen erfüllen:

- 1 Es muss ein Touchscreen-Display sein
- 2 Es muss einfach an das Raspberry anschließbar sein
- 3 Es sollte nicht zu teuer sein

Aufgrund dieser Anforderungen wurde das Touchdisplay von Raspberry gewählt.

### 2.2.4 Wieso pi4j?

Da Java als Programmiersprache gewählt wurde, musste eine Möglichkeit die GPIO-Pins anzusteuern gefunden werden. Da bei der Recherche außer pi4j Java kaum etwas gefunden wurde, wurde pi4j gewählt. Weiters vorteilhaft ist, dass das Ansteuern der Pins via Code nicht sehr kopliziert ist.

### 2.2.5 Wieso Mongodb?

Eine Datenbank wurde gewählt, weil es gegenüber des Speichers der Daten in eine Datei mehrere Vorteile aufweist.

### **2.2.5.1 Vorteil gegenüber Daten in Datei speichern**

Vorteile einer Datenbank:

- 1 Keine Probleme mit Pfaden
- 2 Daten sind alle in einem Punkt gespeichert und nicht im System verteilt
- 3 Der benötigte Code für die Datenbank macht das Programm übersichtlicher

### **2.2.5.2 Vergleich mit anderen Datenbanken**

Vorteile von Mongodb gegenüber anderen Datenbanken (zB mySQL):

- 1 Mongodb ist schemenlos (Daten benötigen keine bestimmte Struktur)
- 2 Mongodb ist kostenfrei

## **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 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, haben den Datentyp JSON.

## 2.3 Umsetzung

Bei der Umsetzung, also beim Schreiben des Programms, wurde wie folgt vorgegangen. Zuerst wurden alle GUI-Fenster per Hand grob designed. Anschließend wurden diese im Netbeans als JFrame Form erstellt. Genaueres über die GUI-Fenster folgt uner Punkt 2.3.4. Danach wurden grundlegende Funktionen die das Programm zu erfüllen hat implementiert. Weiters wurden noch: 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 gespeichertem Datensatz automatisch ein einmaliger Identifikator gegeben. Weiters ist es kostenfrei und man benötigt keine Lizenzen.

Bei einer schemenbehafteten Datenbank werden die Daten in Reihen und Spalten gegliedert. Um eine solche Datenbank effizient nutzen zu können wird auch ein einmaliger Identifikator.

In Mongodb sind Zeilen Collections und Spalten Documents.

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 Dateien veraltet werden. Unter verwalten wird das Ändern, Hinzufügen und Löschen von Daten verstanden. Es können auch neue Dateien angelegt oder alte Dateien gelöscht werden.

Befehle:

- **show dbs** ... zeigt alle angelegten Dateien an
- **show collections** ... zeigt alle collections in einer Datei
- **use** ... verwenden einer Datei, falls die Datei noch nicht existiert wird sie neu erstellt  
(Beispiel: use <Dateiname>)
- **drop()** ... löschen einer Datei  
(Beispiel: <Dateiname>.drop() )
- **find()** ... suchen nach bestimmten Daten  
(Beispiel: db.data.find() )
- **count()** ... zählt die 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(<DokumentID>, <Daten>) )
- **deleteOne()** ... löschen eins Dokumentes  
(Beispiel: db.data.deleteOne(<DokumentID>) )

### 2.3.1.2 Datenbankmanagementsystem DBS

Ein Datenbankmanagementsystem verwaltet eine oder mehrere Datenbanken. Mehrere Datenbanken werden dabei benötigt, wenn mehrere Anwendungen oder Programme jeweils eine eigene Datenbank brauchen. Ein Beispiel für ein DBS 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 dieser Klasse aus auf die Datenbank zugegriffen wird und nur eine Verbindung geöffnet wird.

Ein weiterer Vorteil davon 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.

### 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 zähleIn:

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();
```

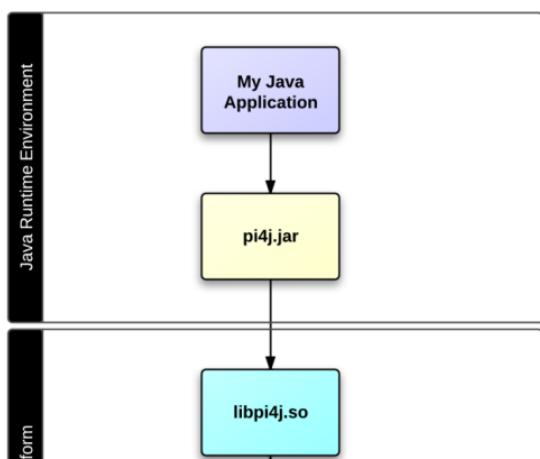
Diese Zeile Code sucht in der Collection **collUser** nach dem Dokument mit dem Identifikator **new BasicDBObject("identifier", "User")**. Das gefundene Dokument wird dann der Varibale **document** zugewiesen.

### 2.3.2 pi4j

#### 2.3.2.1 Allgemeines

Pi4j ist eine freie Software, welche Bibliotheken zur Verfügung stellt, mit denen es möglich ist, von einem Java Programm, auf die IO-Pins eines Raspberrys zuzugreifen. Dabei kann ein GPIO-Pin als In- oder Ouput definiert werden. Wenn ein Pin als Output definiert wird, kann er den State High (+5V) oder Low (0V) haben. Mit einem Input Pin, können Signale gemessen werden. Das Ergebnis der Messung ist wieder 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.

Pi4j stellt eine Verbindung von der JVM (Java Virtuaul Machine) zu dem nativen System des Raspberys her. Dadurch wird es möglich von dem Programm auf die Pins zuzugreifen. Die folgende Grafik zeigt welche Bibliotheken dazu verwendet werden



Pi4j stellt eine Verbindung von der JVM (Java Virtuaul Machine) zu dem nativen System des Raspberys her. Dadurch wird es möglich von dem Programm auf die Pins zuzugreifen. Die folgende Grafik zeigt welche Bibliotheken dazu verwendet werden.

### 2.3.2.2 Pin Numbering Scheme

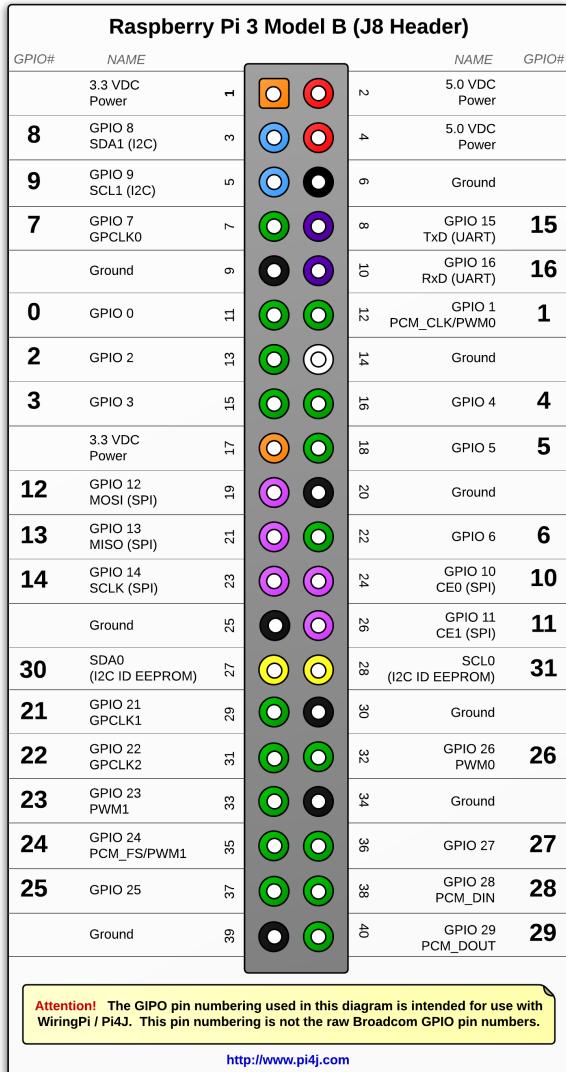


Abbildung 2.2: Pin Numbering Scheme

### 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 jeweils eine H-Brücke angesteuert. Das bedeutet, dass jeder der zwei Motoren jeweils mit vier Transistoren angesteuert wird. Jeder Transistor wird von einem eigenen Pin angesteuert. Für die Sensoren wird jeweils ein Pin zum Auswerten benötigt. In Summe werden zehn GPIO-Pins benötigt.

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

Pin	Verwendungszweck
GPIO_00	Sensor Schüsselplatte
GPIO_01	Sensor Förderband (Futtersackerl)
GPIO_02	Motor Schüsselplatte: Transistor 1
GPIO_03	Motor Schüsselplatte: Transistor 2
GPIO_04	Motor Schüsselplatte: Transistor 3
GPIO_05	Motor Schüsselplatte: Transistor 4
GPIO_06	Motor Förderband: Transistor 1
GPIO_10	Motor Förderband: Transistor 2
GPIO_08	Motor Förderband: Transistor 3
GPIO_09	Motor Förderband: Transistor 4

Tabelle 2.1: Belegung der GPIO-Pins

Wenn einer der beiden Sensoren betätigt wird, liefert dieser das Signal High (+5V). Wenn der Sensor nicht betätigt ist liefert dieser Low (0V).

Um einen Motor einzuschalten müssen jeweils zwei diagonal zueinander liegende Transistoren aktiviert werden. Der Motor dreht sich im Uhrzeigersinn, GPIO-Pins von Transistor 1 und Transistor 4 High sind. Gleichzeitig muss sicher gestellt werden, dass die GPIO-Pins der Transistoren 2 und 3 Low sind. Wenn das nicht sichergestellt ist kann es zu einem Kurzschluss zwischen der Spannungsversorgung und GND kommen. Um den Motor gegen den Uhrzeigersinn zu drehen müssen die GPIO-Pins der Transistoren 2 und 3 auf High sein.

### 2.3.2.4 Singleton

Die benötigten Methoden von pi4j wurden in einem Singleton implementiert. Es musste ein Singleton verwendet werden, weil der benötigte Controller für die Pins nur einmal erzeugt werden kann. Wenn der Controller trotzdem öfters erzeugt wird, wird ein Error geworfen. Der Singleton wird nun dazu verwendet, dass auf die Pins von unterschiedlichen Klassen zugegriffen werden kann. Der Zugriff von mehreren Klassen ist ohne einen Singleton programmiertechnisch nicht lösbar.

### 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 = 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:

- Zugreien 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);
```

- Zugreien 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 in einem eigenen Thread gestartet. In diesem 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 Thread wieder geschlossen.

Der Server wurde auch als Singleton implementiert damit von mehreren Klassen ausgehend 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 ein gültige Request (Anfrage) ist und wie die Response (Antwort) auf den jeweiligen Request aussieht.

Wenn der Request des Clients mit einem **GET** beginnt, bedeutet dass, das der Client Daten vom Server fordert. Beginnt der Request mit einem **PUT**, bedeutet dass, das der Client dem Server Daten schicken will. Nach jedem **GET** oder **PUT** folgt ein URL der 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	URL	
GET	/errors_warnings	Der Server schickt dem Client die aktuell anzuseigenden Errors und Warnungen in einer Json-Array
PUT	/ChangeMachineState	Der Server ruft eine Methode auf um den Maschinenzustand zu ändern

Tabelle 2.2: Übertragungsprotokoll

### 2.3.3.3 Response-GET

Wenn der Server einen Request mit dem Beginn **GET** bekommt, muss er dem Client die Daten schicken. Bei der Katzenfütterungsanlage werden dem Client vom Server die ganzen auftretenden Errors und Warnungen geschickt. Die Errors und Warnungen werden dem Client als Json-Array geschickt. Diese Json-Array wird in der Klasse, welche alle Errors and Warnungen verarbeitet, erstellt. Die Json-Array 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 Maschinenzustand geändert wird, die Methode **machineStateChanger()** aufgerufen, welche ihn 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 eingelenkt.
- 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 URL überprüft. Im URL wird dem Server mitgeteilt was er machen soll. Der Inhalt des URL 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.

Dazu dient die **ErrorAndWarningHandler\_Singleton** Klasse. In dieser Klasse wird beim Auftreten eines Fehlers eine 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.

### 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, String errorTime)
{
    error_hasFeedingFailed = errorOn;
    failedFeedingTime = errorTime;
}
```

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.

Mit dem zweiten Parameter **errorTime** wird die Zeit übergeben, zu der die Fütterung fehlgeschlagen ist. Diese Zeit wird dann in der Errormeldung ausgegeben.

### 2.3.4.3 Json-Array

Diese Klasse stellt auch die Json-Array zur Verfügung, welche dann dem Client (Web-Applikation) geschickt wird.

Die JSONArray sieht wie folgt aus:

Listing 2.16: Json-Objekt mit Errors und Warnungen

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

#### 2.3.4.4 Listenmodel

Die Zeile, in der die Klasse für das Listenmodel angelegt wird, sieht wie folgt aus:

Listing 2.17: Listenmodel-Klasse erstellen

```
public class ErrorAndWarningModel extends AbstractListModel
```

Nach dem Klassennamen muss **extends AbstractListModel** hinzugefügt werden. Dies bedeutet, dass das erstellte Listemodel eine Vererbung von **AbstractListModel** ist.

Das Listenmodel wird wie folgt angelegt:

Listing 2.18: Liste erstellen im Model

```
private final List<String> errorAndWarning;
```

Damit ist nun festgelegt, dass die Elemente in der Liste Strings sind.

#### 2.3.4.5 Liste

Eine Liste in Java wird dazu verwendet, um mehrere Daten in einem Objekt zu speichern. Die Liste kann wie folgt erstellt werden:

Listing 2.19: Liste erstellen

```
List<String> list = new ArrayList();
```

Das Objekt der Liste verfügt über eigene Methoden.

- Anzahl der Elemente in der Liste zählen:

Listing 2.20: Anzahl der Elemente abfragen

```
list.size();
```

- Hinzufügen eines Elementes:

Listing 2.21: Element hinzufügen

```
list.add("Das ist ein Element!");
```

- Leeren der Liste (Löschen aller Elemente)

Listing 2.22: Liste leeren

```
list.clear();
```

Die Errors und Warnungen, die in der Liste gespeichert werden, werden in der GUI in einer Liste (jList) dargestellt. Siehe Kapitel 2.3.6.2.

### 2.3.5 SwingWorker

Der SwingWorker in Java wird benötigt, wenn eine Anwendung mehr Zeit benötigt. Mithilfe des SwingWorkers kann diese Aktion im Hintergrund ausgeführt werden. Der Vorteil davon ist, dass die GUI verwendbar bleibt und nicht blockiert. Dafür wird dann mit Hilfe des SwingWorkers ein Hintergrund-Thread geöffnet in dem die Aktion, die mehr Zeit benötigt, abgearbeitet wird.

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

Listing 2.23: SwingWorker Klasse erstellen

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

Im Generic steht als erstes der Rückgabewert von **done()** und als zweites der Rückgabewert von **process()**.

#### 2.3.5.1 EDT

Im Event Dispatch Thread (EDT) wird die GUI 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.

#### 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 ein **TimeUnit.MILLISECONDS.sleep(500)**; verwendet. Mit dieser Methode wartet das Programm, an der Stelle wo die Methode aufgerufen wird, für 500 Millisekunden. Dadurch wird der Prozessor nicht so stark ausgelastet.

#### 2.3.5.3 Methoden des SwingWorkers

- **doInBackground()**

Mit **doInBackground()** kann ein neuer Hintergrund-Thread geöffnet werden. Dieser Thread läuft parallel zum EDT. Der Hintergrund-Thread wird geschlossen, wenn der Worker beendet wird oder sich der Worker selbst beendet, weil er alle Methoden abgearbeitet hat. In einem Hintergrund-Thread darf nicht auf GUI-Elemente zugegriffen werden.

Im folgenden Beispiel wird ein Hintergrund-Thread gezeigt, welcher nur durch das Beenden des Workers beendbar ist.

Listing 2.24: 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;
}

```

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

Listing 2.25: SwingWorker doInBackground() ohne while Schleife

```

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

publish( timeOfDay);

TimeUnit.MILLISECONDS.sleep(500);
return 1;
}

```

- **done()**

Wenn ein Hintergrund-Thread beendet wird, wird **done()** im EDT aufgerufen. Im **done()** können dann GUI-Elemente bearbeitet werden.

Listing 2.26: SwingWorker done()

```

@Override
protected void done()
{
    // TODO
}

```

- **process()**

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

Listing 2.27: SwingWorker process()

```

@Override
protected void process(List<String> chunks)
{
    (timeOfDay = chunks.get((chunks.size() - 1));
    lbTimeOfDay.setText(timeOfDay);
}

```

- **cancel()**

Mit **cancel** kann ein Worker beendet werden. Dafür muss folgender Befehl aufgerufen werden:

Listing 2.28: SwingWorker abbrechen

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

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

## 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 mittels Touch-Gesten zu bedienen.

Beim Starten des Raspberries wird das Programm automatisch gestartet. Dabei wird das MainWindow, 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.

### 2.3.6.1 Java-Programm im Autostart

Um auf einem Betriebssystem, das auf Linux basierendem, 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

Dies zweite Möglichkeit ist **autostart**. Hier ist der X-Server, welcher für eine grafische Ausgabe benötigt wird, enthalten. 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 &**

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 **MainWindow**.

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

### 2.3.6.2 MainWindow

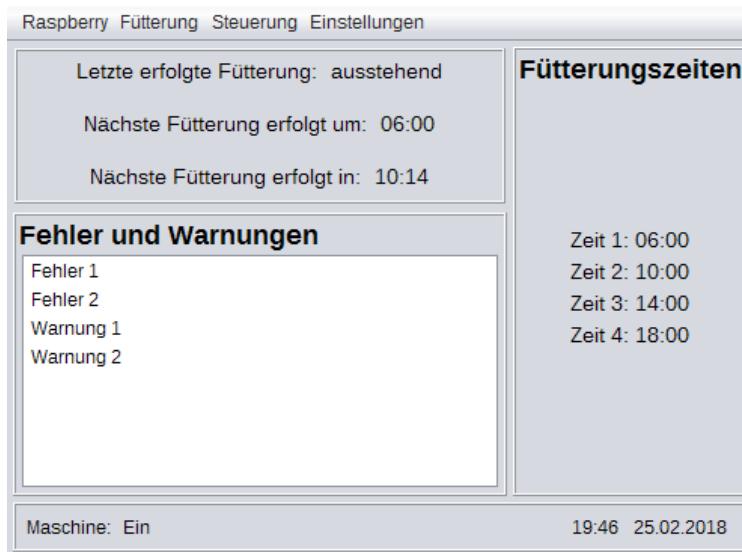


Abbildung 2.3: MainWindow

Das **MainWindow** besteht aus einer **Java JFrame Form**. In diesem Fenster wird dem Benutzer ein allgemeiner Überblick über die Anlage zur Verfügung gestellt. Somit können alle wichtigen Informationen schnell und leicht ersichtlich. Weiters ist das **MainWindow** das Hauptfenster des Programms. Über diese sind alle Dialog-, Steuerungs- und Informationsfenster erreichbar. Diese Fenster werden von Kapitel 2.4 bis Kapitel 2.9 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. Somit wird ein nur Objekt des **MainWindow** erstellt. Das Singleton-Objekt wurde wie folgt erstellt:

Listing 2.29: MainWindow createInstance()

```

public static MainWindow createInstance()
{
    if (instance != null)
    {
        throw new RuntimeException("instance 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 ein jeweils eigener Error geworfen.

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

Listing 2.30: 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 2.31: 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. Folgende Funktionen sind blockiert, wenn die Fütterung aktiviert ist:

1 manuelle Steuerung

2 Update

Das Blockieren das Funktionen wird mit folgender Methode gemacht:

Listing 2.32: 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)
    {
        menu_update.setEnabled(false);
        menu_manualControl.setEnabled(false);
    }
    else
    {
        menu_update.setEnabled(true);
        menu_manualControl.setEnabled(true);
    }
}
```

### 2.3.6.3 TimeManagement



Abbildung 2.4: TimeManagement

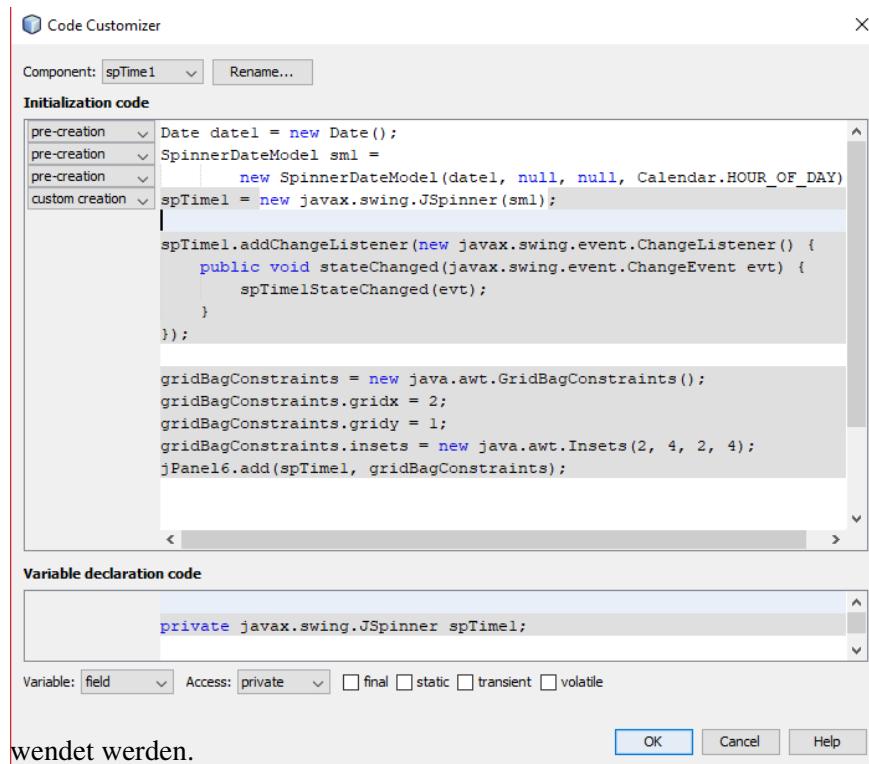
In der Klasse **TimeManagement** ist es dem Benutzer möglich alle Zeiten zu verwalten. Das GUI-Fenster dieser Klasse ist ein Dialogfenster. Bei Dialogfenstern ist es üblich, dass sie mit **Ok** und **Abbrechen** bedienbar sind.

Es können bis zu vier verschiedene Zeiten gewählt werden. Dabei müssen die Zeiten, bei Zeit 1 beginnend, aufsteigen 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 die 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.

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



wendet werden.

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

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

Diese Schritte sind für alle der vier verwendeten Spinner ver-

Abbildung 2.5: Spinner Konfiguration

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

Listing 2.33: 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 Spinner mit den momentanen Zeiten gefüllt. Zusätzlich werden noch die Häkchen, die festlegen ob eine Zeit aktiv ist, gesetzt.

Bevor das Fenster geschlossen wird, wenn **Abbrechen** gedrückt wird, wird 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 2.34: Spinner Event

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

Diese Event wird aufgerufen, wenn der Inhalt der Spinners verwendet wird. Dabei wird dann eine Boolean-Variable, die bekannt gibt, ob etwas verwendet 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. Zusätzlich kann dann ausgewählt werden, ob wirklich ohne zu speichern beendet werden soll.

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

Listing 2.35: Zeitendokument Getter-Methode

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

newTimeDoc = new BasicDBObject("identifier", "Times")
    .append("time1", time1)
    .append("time1_active", time1_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 anschließen mit folgender **Getter-Methode** geworfen damit sie in im **MainWindow** zugänglich ist:

Listing 2.36: Zeitendokument

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

### 2.3.6.4 CreateUser



Abbildung 2.6: CreateUser

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 wird der Benutzer in das Programm übernommen und in die 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. Diese Eingabefeld wird speziell dazu verwendet Passwörter einzugeben. Eine Besonderheit davon ist, dass die Passwörter nicht als Text sondern als Sternchen dargestellt werden. Eine weitere 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 Password im Programm besser verarbeiten zu können wird es wie folgt in einen String umformatiert:

Listing 2.37: char zu String

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

Um eine gewisse Sicherheit zu gewährleisten wird das Password, bevor es in der Datenbank gespeichert wird, gehasht. Um das Password zu hashen wird der **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.

Das Password wurde wie folgt gehasht:

Die Klasse **CreateUser** bietet dem Benutzer die Möglichkeit einen Benutzer anzulegen. Mit dem angelegten Benutzer ist es möglich sich auf der Webseite der Anlage anzumelden. Die GUI ist in Abbildung 2.6 zu sehen.

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

Listing 2.38: 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;
}

```

Mit dem Parameter **String salt** kann noch ein String übergeben werden welcher vor dem Hashen des Passworts am Passwort angehängt wird. Somit wird die Entschlüsselung des Passwort nochmals erschwert und die Sicherheit erhöht sich.

Bevor das Fenster geschlossen wird, wenn **Update** gedrückt wird, 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 2.39: Textfeld Event

```

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

```

Wenn die Variable **false** ist, also noch nicht gespeichert wurde, wird dem Benutzer die Warnung ausgegeben werden, dass die veränderten Daten noch nicht gespeichert sind.

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

Listing 2.40: Benutzerdokument

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

Dieses Dokument wird anschließend noch mit einer **Getter-Methode** geworfen damit es im **MainWindow** zugänglich ist:

Listing 2.41: Benutzerdokument Getter-Methode

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

### 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 Futterschüsseln, immer nachdrehen kann. Das gleiche gilt für das Förderband.

Die GUI ist in Abbildung 2.7 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 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 aus geschalten 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 Stop-Knopf wieder stopbar.

Falls die Motoren vom Benutzer vor dem Schließen des Fensters nicht gestoppt werden, werden diese automatisch vom Programm gestopt. Dies dient dazu, dass sichergestellt ist, dass die Motoren immer gestopt werden.

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

Listing 2.42: Motoren stoppen und Fenster schließen

```
private void onSchließen(java.awt.event.ActionEvent evt)
{
    // stop engines when closing the control dialog - security measurement
    pi4j_instance.stopEngine1();
    pi4j_instance.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 in die Anlage zu sehen 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. Aus dieser Klasse ausgehend werden die Output-Pins und in weiterer Folge die Motoren gesteuert.

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

Listing 2.43: Motoren drehen

```
private void onMoveEngine1Counterclockwise(java.awt.event.ActionEvent evt)
{
    pi4j_instance.moveEngine1Counterclockwise();
}
```

Um die aktuellen Positionen der Motoren und Zustände der Sensoren zu ermitteln und ausgeben wird ein SwingWorker verwendet. Dies ist unter Kapitel 2.3.6.6 genauer beschrieben

### 2.3.6.6 Positionsinformation



Abbildung 2.8: Positions-informati-on

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 2.8 zu sehen.

Die GUI der Klasse ist ein Informationsfenster. Da dieses Fenster kein Dialogfenster ist, werden **Ok** und **Abbrechen** 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 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

- Unbetägt
- Betägt

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 werden in einer Dauerschleife 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 2.44: 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 2.45: 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()** werden alle 100 Millisekunden die Motordrehrichtungen und die Sensorzustände abgefragt.

### **2.3.6.7 SystemInfo**

bild fehlt weil server nicht gelaufen ist

### 2.3.6.8 Update

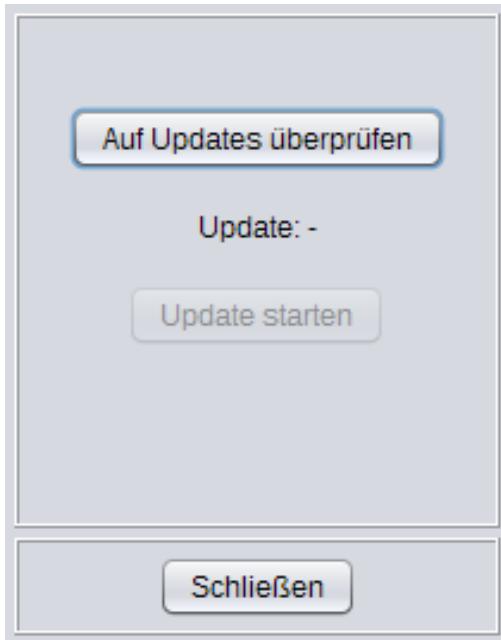


Abbildung 2.9: Update suchen



Abbildung 2.10: 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.

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 2.9. 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**. Also sieht das Fenster aus wie in Abbildung 2.10.

Beim Überprüfen auf Updates wird die lokale Versionsnummer am Raspberry und die Versionsnummer am Repository überprüft. 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 herunter geladen. Dies wird mit einem **git pull** Befehl gemacht. Anschließend wird am Raspberry eine Datei bearbeitet, welche dem Raspberry beim Startet 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 Funktion aufgerufen:

Listing 2.46: Update

```
private void onUpdate(java.awt.event.ActionEvent evt)
{
    try
    {
        Process process = Runtime.getRuntime().exec("git pull");
        process.waitFor();

        write("/home/pi/updatefile/build.txt");

        pTextUpdateErfolgreich.setVisible(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");
    }
}
```

## **2.4 Zusammenfassung/Verbesserungsmöglichkeiten**

**2.4.1 Probleme mit Mongodb am Raspberry**

**2.4.2 Probleme mit pi4j -> Snapshotversion verwendet - ansonsten werden pins nicht erkannt**

**2.4.3 GUI auf "Touchscreen-Designändern**

**2.4.4 Besser Benutzerverwaltung - Mehrere Benutzer anlegen**

**2.4.5 Selbst erstellbare Vorlagen in denen Zeiten gespeichert werden**

# KAPITEL 3

---

## Mechanik

---

### 3.1 Einleitung

### 3.2 Aufgabenstellung und Zielsetzung

Ziel ist es eine Katzenfütterungsanlage zu entwickeln. Ausgangslage sind Katzenfutterbeutel aus Aluminium-Kunststoff-Folie, die zeitlich gesteuert das Futter in den Behälter füllen sollen. Das Futter soll der Katze zugänglich gemacht werden und die leeren Beutel geruchsisoliert entsorgt werden. Die Anlage soll zum Beispiel während Kurzurlauben oder zum normalen Füttern der Katze oder des Hundes verwendet werden. Nach Aktivieren der Anlage wird in davor gewählten definierten Zeitpunkten 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 hat. Darum wird eine hoher Anpressdruck der Klingen erwartet. Zudem darf die Klinge, auch wenn sie lang ist, nicht verbiegen. Damit die Aluminium-Kunststoff-Folie nicht zwischen den Klinge gelangt und diese auseinander presst. Das hat Zufolge, dass das die Packung zerknittert und schwerer zu schneiden ist.

#### 3.3.2 Problematik der Dichtheit bei Klemmen

Bei der zweiten Variante wird erwartet das 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. Ein weiterer Nebeneffekt ist, das Schädlinge in die Packung gelangen können und die Katze, da sie wählerisch sein kann, das Futter nicht frisst.

### **3.3.3 Problematik bei Entleerung der Verpackung**

Beim Entleeren der Verpackung bei geleierartiger Füllung treten einige Probleme auf die sich meist nur mit dem Pressen der Verpackung lösen lassen. Das Futter kann erstens in der Verpackung auf der Folie haften und somit durch die Schwerkraft nicht vollständig entleert werden. Zweitens die Luft muss von der Öffnung bis zur geschlossenen Seite gelange, damit der Luftdruck nicht das Entleeren verhindert(ähnlich wie beim Entleeren einer volle Ketchupflasche).

### **3.3.4 Problematik vom Geruch**

Bei automatischen Füttern eine Katze ist der Geruch ein großes Problem, da Katzenfutter schon am ersten Tag einen strengen Geruch hat und der sich über die Tage steigert. Die einzige Maßnahme die getroffen werden kann, ist die ganze Maschine Luftpicht zu gestalten (die einzige Ausnahme wäre, wenn es dem Benutzer nicht stört und dieser erleichtert ist wenn seine Katze gefüttert ist).

### **3.3.5 Problematik von der Reinigung nach dem Urlaub**

Durch das Pressen der Packungen kann das Gelee an der Walze 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 kann und somit die beiden Walzen reinigen. Die Futterschüsseln lassen sich durch die Konstruktion der unten beschriebene Variante (Drehplatte) leicht entfernen lassen. Die Futterplatte kann bei eines Falles einer Verschmutzung durch ihre wasserfeste Beschichtung gereinigt werden.

### **3.3.6 Problematik bei 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. Weiters ist das Dichthalten der Truhe ein großer Schwerpunkt, 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. Weiters werden durch Scheinskizzen die einzelnen Varianten verdeutlicht um so einen Eindruck der zu realisierten 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.

#### **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

### 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 das die vom Futterhersteller für die Öffnung vorgesehene Seite in Richtung des Schneidewerks zeigt (die schmale Seite mit der Einkerbung). 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 und somit keine freie Bewegung ermöglicht ist. Der Oberteil des Bandes schließt eben mit der Schneideplattenhöhe ab um ein leichtes gleiten der Packung durch den Greifer zu ermöglichen, ohne das es Höhenunterschiede überwinden muss. Weiters wird über zwei Räder ein Band gespannt an denen die Wände in Abstand der Dicke der Packung festgemacht werden. Das Band wird mit Hilfe 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 können je nach Länge eine gewissen Anzahl an Futterpackung gelegt werden, natürlich nur auf der Oberseite da die Packungen ansonsten aus den Abteilungen fallen.



Abbildung 3.1: Magazin Modellaufbau von Vorne zu ermöglichen, ohne das es Höhenunterschiede überwinden muss. Weiters wird über zwei Räder ein Band gespannt an denen die Wände in Abstand der Dicke der Packung festgemacht werden. Das Band wird mit Hilfe 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 können je nach Länge eine gewissen Anzahl an Futterpackung gelegt werden, natürlich nur auf der Oberseite da die Packungen ansonsten aus den Abteilungen fallen.

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

In der Abbildung: 3.3 wird das Magazin von Oben gezeigt.

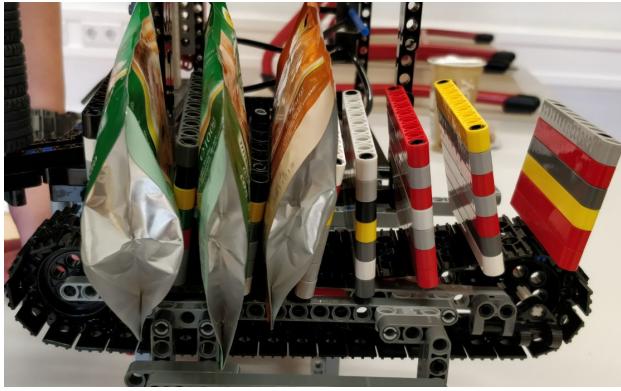


Abbildung 3.2: Magazin Modellaufbau von der Seite

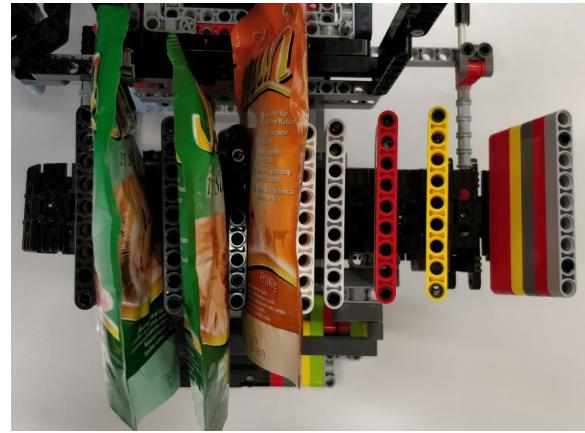


Abbildung 3.3: Magazin Modellaufbau von Oben

### 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 an Stelle zu halten ohne dass der Schnittpunkt verrutscht und die Schneide nicht mehr die Einkerbung, die leichter zu Schneiden ist, trifft. Das 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 sie 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 das Katzenfutterpackerl 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.

### 3.4.1.4 Schnitt

In der richtigen Position muss man mit 2 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 gelangt zu viel Kunststoffmaterial zwischen die Schneideflächen und durch die Länge der Schneiden biegen sie sich auseinander und somit kommt kein ordentlicher Schnitt zusammen (Zugspannung statt örtliche Schubspannung). Bei öfteren Auftritten dieses besprochenen 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. Siehe Abbildung: 3.7



Abbildung 3.7: Schnitt

Eine andere Alternative wäre ein feingezähntes Schneiderad mit hoher Drehzahl. Versuche mit einem gezähnten Messer haben gezeigt etwa 9 Schnitte mit einem Messer erforderlich waren um eine 10cm Verpackung vollständig aufzuschneiden.

### 3.4.1.5 Pressen

Nach dem Aufschneiden wird mit einer Rolle die Packung ausgepresst. Dazu werden zuerst die ersten 2 Magnetzyliner gelöst bis die Rolle vorbei ist. 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 2 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 Verpackung und wird nicht von der Walze zerquetscht. Nach der Beseitigung der Verpackung werden zuerst die beiden Magnetzyliner von der Maschine entfernt in Anfangsposition gebracht, damit die Walze ohne Probleme in Startposition zurückkehren kann. Siehe Abbildungen: 3.8, 3.9, 3.10



Abbildung 3.8: Ausquetschen Beginn



Abbildung 3.9: Ausquetschen Mitte



Abbildung 3.10: Ausquetschen Ende

Eine andere 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 2 Magnetzyliner benötigt werden und die in einem Schaltplan zu berücksichtigt sind. Außerdem benötigen sie zusätzlich Platz. Die Klappe befindet sich hinter der Futterverpackung. Siehe Abbildung: 3.11.



Abbildung 3.11: Auswurf Beginn

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

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

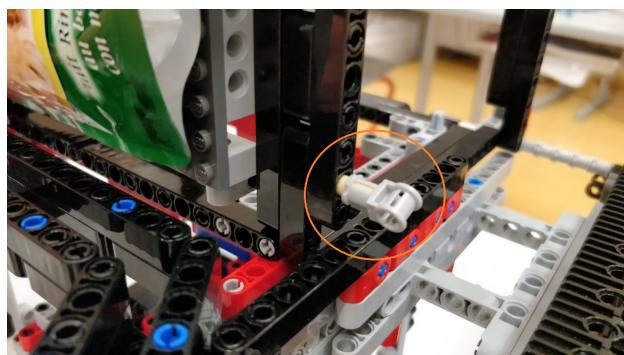


Abbildung 3.12: Bolzen drinnen



Abbildung 3.13: Bolzen entfernen

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

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



Abbildung 3.14: Klappe öffnen

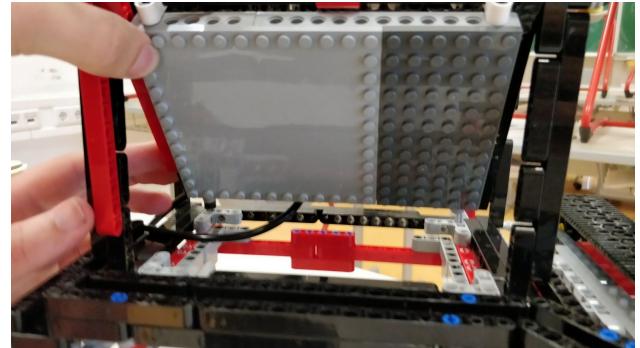


Abbildung 3.15: Fertiger Auswurf

### 3.4.1.7 Füttern

Die Maschine besitzt 5 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 es 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 anderen Schüsseln die auf der Platte montiert sind, dass die Katze nicht soweit zur Futterschüssel hat d.h. sie muss nur mit dem Kopf zur Plattenoberfläche und nicht Platte + Schlüsselhöhe, somit spart man je nach Schüssel wertvolle Zentimeter. Die Platte ist mit einer Gummischicht überzogen damit die Schüssel, durch der Kopf der Katze, wenn sie frisst, nicht verrutscht. Sie lässt sich aus der Platte entnehmen, indem der Benutzer mit der Hand die Futterschüssel von unten durch das loch drückt und mit der anderen Hand entnimmt. Danach werden die Schüsseln gewaschen, getrocknet und die Schüssel in das Loch fallen gelassen. Siehe Abbildung: 3.16

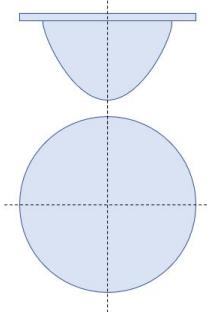


Abbildung 3.16: Loch Futterschüssel

### 3.4.2 Variante 2: Vor aufgeschnittene Packung

Diese Variante wurde entwickelt um den Schneidemechanismus zu umgehen, da dieser keine leichte Unterfangen ist. Hierbei wird jedoch der Benutzer aufgefordert 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 gesetzt. Zum Großteil funktioniert das Prinzip mit der Schwerkraft der die Packung entleeren soll.

### 3.4.2.1 Förderband und Kettenglieder

Beim Förderband erkennt man wo sich die Futterpackungen befinden sollen. Es 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 hat (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. Dadurch 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 der Öffnung(Pfeilspitzen) nach unten zeigen. Siehe Abbildung: 3.17.

Die dunklere von den blauen Flächen ist die 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.

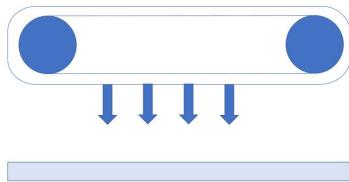


Abbildung 3.17: Foerderband

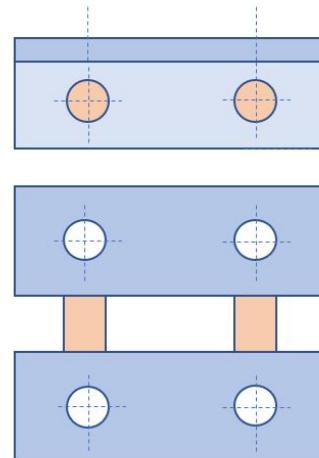


Abbildung 3.18: Kettenglied

### 3.4.2.2 Walze

Nach dem 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 Welle auf einer eigen gefertigten Welle. Beide Walzen werden durch eine Feder aneinander gepresst, nur so stark, damit die Halterung, an der die Packung festgemacht ist, durchkommt. Dennoch so stark damit sich die Packung entleert. Die Walze an der eigen gefertigten Welle wird mit zwei Aluplatten und einem Scharnier in Stellung gehalten.

Auf der Walze ist ein Pfeil gezeichnet, der Pfeil stellt eine Futterpackung da und die Pfeilrichtung zeigt die Richtung in der die Packung ausgepresst wird. Siehe Abbildungen: 3.19.

Das dunkelblaue ist das eigentliche Scharnier, die hellblauen Flächen sind die Verlängerungen. Siehe Abbildungen: 3.20.

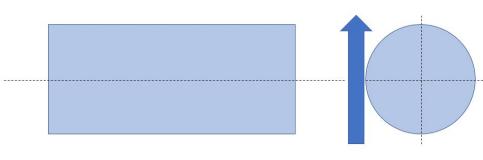


Abbildung 3.19: Walze

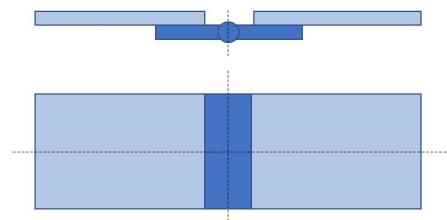
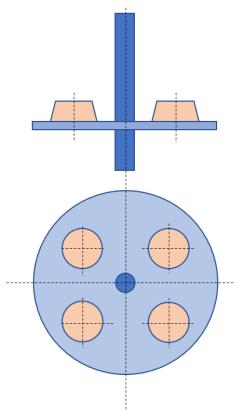


Abbildung 3.20: Scharnier

### 3.4.2.3 Futterplatte

Nach dem Pressen wird das Futter in die Schüssel gequetscht bzw. es rinnt in die Schüssel. Die Platte hat eine gewisse Anzahl an Schüsseln, je nach Bedarf maximal fünf Schüsseln. Diese sind auf einer Platte platziert, durch den Plattenmittelpunkt geht eine Welle die die Platte nach links und rechts drehen kann. Siehe Abbildung: 3.21



### 3.4.3 Variante 3: Gefrorenes Futter

Abbildung 3.21: Futterplatte

In dieser Variante wurde überlegt, ob man das Futter nicht einfrieren kann, dieses danach aus der Gefriertruhe zu holen und zu erwärmen. Der Vorteil hierbei ist, dass keine Schädlinge in das Futter gelangen können da es tiefgefroren ist und es kann die Portionsgröße eingestellt werden wie viel die Katze bekommt da der Benutzer die Menge des Futters selbst bestimmen kann. Weiters müsste man nicht über das Schneide Problem nachdenken, da es eine knifflige Angelegenheit ist die Packung bei jeden Schnitt perfekt 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 Magnetzylindern gearbeitet werden, zur Verschiebung der Abdeckung. Zweitens kann ein Loch aus dem der Greifer das Futter entnimmt und dicht halten muss in die Gefriertruhe geschnitten werden. Falls es undicht ist, wird es darin zu warm, das Futter schmilzt und verdirbt schlussendlich. Hinzuzufügen ist auch noch das Katzen wenn es um Futter geht sehr wählerisch sind und somit wenn das Futter gefroren ist, hat man zu einem Teil das Kondenswasser des aufgetauten Futters und zum Anderen schmeckt eingefrorenes Essen anders, also nicht so wie es die Katze gewohnt 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 wird nur an den beiden unteren Ecken gehalten. Siehe Abbildungen: 3.22, 3.23

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

In der Abbildung: 3.25 sieht man das nach 10 Minuten der Inhalte ganz 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 leichte Gelee-artige aber eher dünnflüssige Konsistenz des Futters rinnt es abhängig von der Zeit aus der Packung.
- Dadurch der ganze Inhalt leicht gleitet kann ohne Probleme durch eine Walze das restliche Futter ausgepresst werden.
- Durch das Eigengewicht und der Schwerkraft wird der Entleerungsprozess erleichtert.

### 3.5.2 Schneideversuch 1. Art der 1. Variante

Schnitt anhand einer praxischen Anwendung dargestellt. Der Beutel wird mithilfe einer Papierschneidemaschine 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 kontra ausgewählt bzw. Tests ausgewählt.

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

- Durch den hohen Anpressdruck der Schneide an die Schneidplatte ließ sich die Packung vollständig öffnen.
- Es gelang beim 1.Versuch mit nur einen durchgehenden Schnitt.
- Die lange Klinge machte auch keine Problem indem die Packung zwischen der Schneidfläche und Klinge gelang und diese auseinander drückt.
- Durch die vom Futterhersteller vorgegebene Einkerbung kann mit relativ wenig Kraft eine Rissfortpflanzung hervorrufen(genaue Beschreibung unter Variante 1, Führen zur Schneidplatte).
- Konnte wie beim Greifer ein genaues Einspannen hervorrufen.

### 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 geöffnet zu haben, mussten mehrere Schnitte verwendet werden. Siehe Abbildung: 3.29

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

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

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



Abbildung 3.29: Schneidemittel



Abbildung 3.30: Anfangs-schnitt 2.Art



Abbildung 3.31: Mittelschnitt 2.Art



Abbildung 3.32: Endschnitt 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 auf diese Weise der Schneidemechanismus gelöst wäre sollte, müsste man die mit einer anderen Alternative lösen. Das wäre eine Sägezahnrad das mit einer Elektromotor angetrieben wird.
- Kein gerader Schnitt erwartet, wegen das reißen der Verpackung, hervorgerufen durch die zackige Klinge(würden funktionieren mit zusätzlicher Sicherung).

### 3.5.4 Dichtheitsexperiment der Hebelklemme

Die für unsere Maschine entscheidende Klemme wurde 3D gedruckt und getestet wie sich die Packung nach fünf Tagen verhält. Von der Klemme wurde nur die Umrandung der einzelnen Teile gedruckt das heißt die Teile sind in der Mitte hohl und dehnbarer als massive Teile für die Klemme. 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.

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

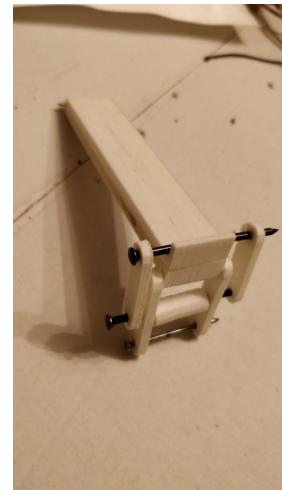


Abbildung 3.33: 3D-Klemme

Tag 1: Die Klemme wurde samt der Packung senkrecht Aufgehängt und an den von Futterhersteller vorgegeben Schneidepunkt aufgeschnitten. Siehe Abbildung: ??er erste Eindruck war die Klemme hält dicht ohne menschliche Einflüsse. Sobald die Klemme wenig 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 werden bogen sich durch die menschliche Einwirkung und den inneren der Packung auseinander. Dennoch war die Klemme, wenn sie sich in Ruhelage befindet dicht. Siehe Abbildung: 3.35.



Abbildung 3.34: Klemmen Probe



Abbildung 3.35: Klemme Tag 1

Tag 2: Zur Überprüfung ob die Packung dicht hält wurde eine weiße Unterlage darunter geschoben in diesem Fall ein Blatt-Küchenrolle. Auch hier sieht man durch einen bildlichen Beweis das 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 sehen 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 im Urlaub ist hält auch die in stabilere Klemme dicht. Siehe Abbildung: 3.39.



Abbildung 3.38: Klemme Tag 4



Abbildung 3.39: Klemme Tag 5

## 3.6 Vergleich der Varianten

In diesem Kapitel der Diplomarbeit werden sämtliche Ideen schemenhaft gezeichnet und erklärt.

### 3.6.1 Klemmen

Es wurden verschiedene Varianten durchdacht auf welche Weise 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 und an diesen Stellen kann Flüssigkeit austreten. Außerdem hält sie bei Zugbelastung nur wenig stand. Siehe Abbildung: 3.40

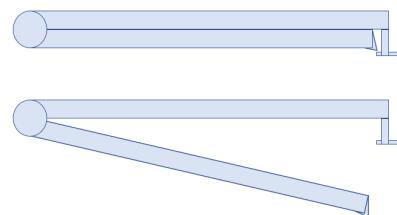


Abbildung 3.40: Einfache Klemme

#### 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. Weiters können die Klemmen auf einer Stange aufgesammelt werden und liegen nicht an unerwünschten Positionen an denen man nicht herankommt. Siehe Abbildung: 3.41

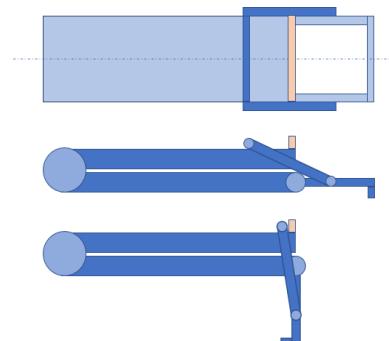


Abbildung 3.41: Hebel Klemme

#### 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 das das Gummiband spröder werden kann und irgendwann reißen kann, also ein hoher Wartungsaufwand. Die Klemmen kann man auch nicht kontrolliert sammeln und somit sind sie schwerer zugänglich.

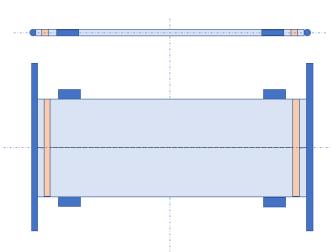


Abbildung 3.42: Gummiband Klemme

### 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 das Förderband sich bewegt, die Lasche durch eine Stange eingefädelt wird und diese durch die Bewegung Richtung Walze geöffnet werden soll.
- Weil die beiden Metallplatten aufeinander pressen hält die Verpackung durch den besonderen Verschluss luftdicht zusammen (siehe Abbildung: 3.41).

### 3.6.3 Futterschüsseln

Bei den Futtergeschüsseln mussten bestimmte Faktoren erfüllt werden bzw. auf wählerrische Katzen abgestimmt werden. Deshalb konnte nicht die einfachste Variante genommen werden die am wenigsten Aufwand benötigt 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. 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: 3.43

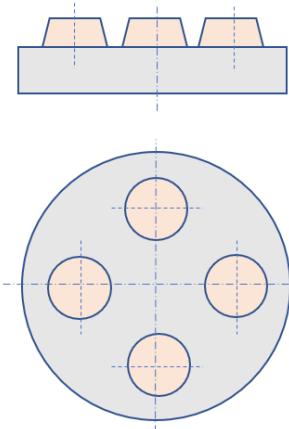


Abbildung 3.43: Drehplatte

#### 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 und alle Zylinder müssen zugleich arbeiten um die Futtergeschüssel zur richtigen Position zu führen. Siehe Abbildung: 3.44

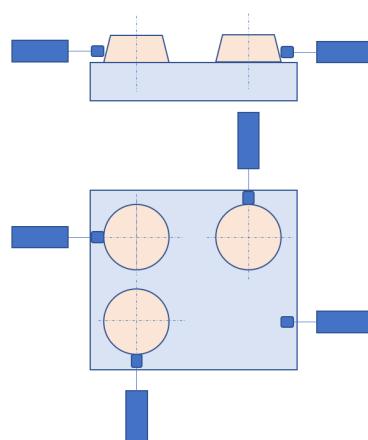


Abbildung 3.44: Platte Zylinder

### 3.6.3.3 Platte mit einer Schüssel

Die Platte mit nur einer Schüssel ist leicht zu realisieren da sie nur wenige Bauteile benötigt. Das wäre zum Einem die Platte auf der die Futterschüssel mit einer Schiene darauf platziert ist und zum Anderen als auch die zwei Magnetzyylinder die die Futtererschüssel in die Anfangs und Endposition bringt. Jedoch 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 es kommt Ungeziefer hinein. Das hat zu Folge das die Schüssel jeden Tag befüllt wird und eventuell übergeht. Siehe Abbildung: 3.45

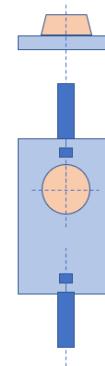


Abbildung 3.45: Einschüsselplatte

### 3.6.4 Futterschüssel Wahlvariante

Die verschiedenen Futterschüssel haben große Vor- bzw. Nachteil im Platzbedarf und Futterchlüsselanzahl. Diese wurden gründlich durchdacht und zum Folgenden Endschluss gekommen: Die Wahlvariante ist die Drehplatte die Gründe dafür werden in den Punkten erörtert:

- Ein großes Thema war die Anzahl der Futterschüsseln, hierbei war es wichtig, dass die Katze jeden Tag eine reine Schüssel zur Verfügung hat.
- Die Schüsseln sollten leicht zu entfernen sein und die Oberfläche der Platte ebenfalls leicht zu reinigen sein.
- Dadurch die Platte auf einer Welle sitzt lässt die 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. Dies sollte ohne Schwierigkeiten bzw. einem zu langen Weg erfolgen.

#### 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 mit einen Greifer in die Schneideposition zu bringen. Der Aufbau ist wie ein Förderband, zwei Räder, ein Band mit oben platzierten

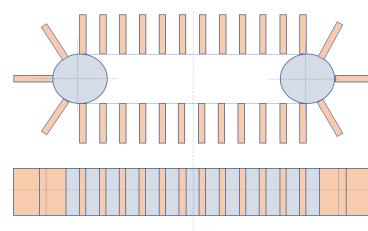


Abbildung 3.46: Futtermagazin Horizontal

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. Siehe Abbildung: 3.46

### 3.6.5.2 Futtermagazin Vertikal

Das Futtermagazin Vertikal ist ein rechteckiges Gehäuse an denen 4 Magnetzyliner platziert werden. In dieser Box kommen die 10 Futterpackungen. Der Ablauf funktioniert in einer gewissen Reihenfolge. Zuerst öffnet sich der erste linke Magnetzyliner danach der gegenüberliegende zweite Magnetzyliner. Daraufhin gelangt die erste Futterpackung auf die unteren Zylinder. Nach diesem Schritt schließen sich die beiden Magnetzylinger wieder, damit die anderen Packungen nach den öffnen der unteren Magnetzylinger nicht durch die Maschine fallen. Daraufhin wenn der Fütterungsbefehl kommt öffnen sich die unteren Zylinder und die Packung gleitet über ein Blech zur Schnittfläche. Der große Nachteil dieser Methode ist das immer wieder Fehler auftreten können. Die Futterpackung kann falsch an der Schneidfläche ankommen bzw. sich an einem bestimmten Ort verkeilen. Siehe Abbildung: 3.47

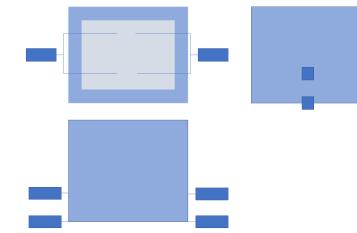


Abbildung 3.47: Futtermagazin Vertikal

## 3.7 Konstruktion der Wahlvariante und Details

### 3.7.1 Drehplatte

Die Drehplatte besteht aus Aluminium und ist 10mm dick. Es werden 5 Löcher für die Schüsseln ausgeschnitten und in der Mitte ist ein Loch, in der eine Stahlwelle durch geht. 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 in den Motor geht 4x4x20, dies ist für Aluminium sehr schmal da große Kräfte auftreten können und der Stift abreißen kann. Die Drehplatte wird auf der Welle mit einem Flansch befestigt damit das vertuschen nach oben, unten und zur Seite gesichert ist. Siehe Abbildung: 3.48

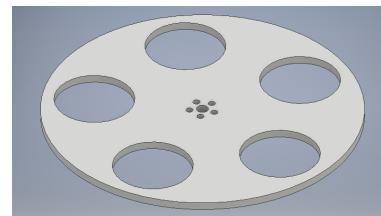


Abbildung 3.48: Drehplatte Inventor

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: 3.49.

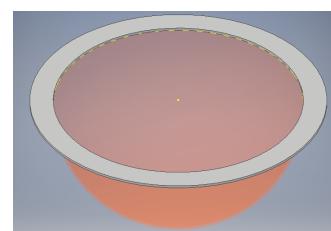


Abbildung 3.49: Futterschüssel Inventor

### 3.7.2 Förderband, Kettenglied und Kettenrad

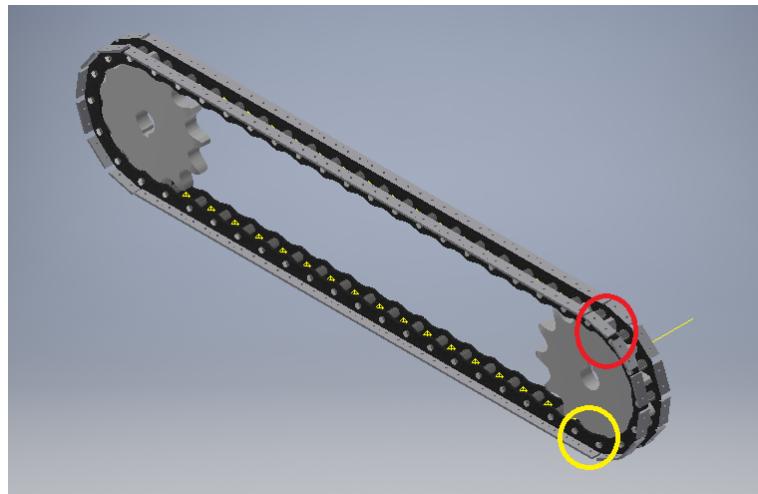


Abbildung 3.50: Kette Inventor

Das Förderband Abbildung: 3.50 wurde in folgenden Schritten konstruiert:

- 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 einen Kreis mit einem Durchmesser von 53,068mm gezeichnet. Der zweite Kreis in der gleichen 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 unsern 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 3.52.
- 7 Siebtens wird das Kettenglied konstruiert und alle 3 Komponenten in ein gemeinsames Projekt eingefügt. Siehe Abbildung: 3.51.
- 8 Achtens wird das Kettenglied am untersten Punkt des rechten Kreises abhängig gemacht.
- 9 Neuntens Durch kann durch die Funktion Muster das Kettenglied gleichmäßig auf der Zeichnung platziert werden und dadurch entsteht die fertige Kette.

10 Zehntens werden die Kettenräder auf den zuvor erstellten Ebenen abhängige 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: 3.51.

Das Kettenrad wurde aus dem Inhaltscenter platziert. Danach wurde eine Skizze angelegt und das Loch in der die Welle durch geht 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: 3.52.

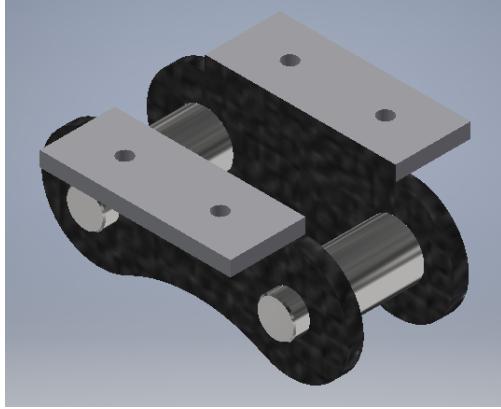


Abbildung 3.51: Kettenglied Inventor

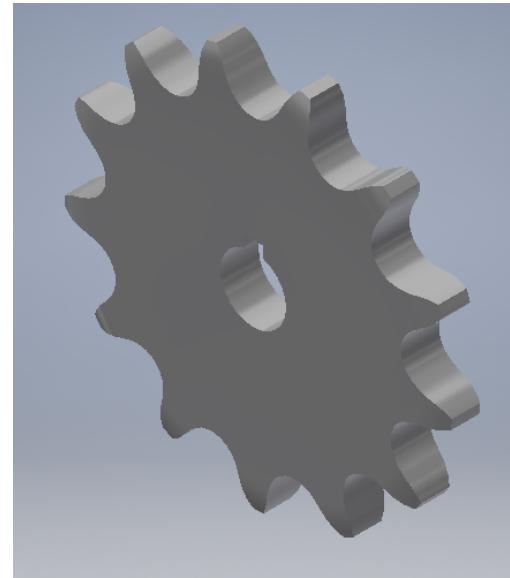


Abbildung 3.52: Kettenrad Inventor

### 3.7.3 Walze

Die beiden Walzen sind aus Kunststoff innen hohl. Diese sind auch nicht auf der Welle befestigt sondern können sich frei drehen. Es ist nur auf jeder Seite ein Sicherungsring damit die beiden Walzen ihre Postion nicht verlassen. Siehe Abbildung: 3.53.

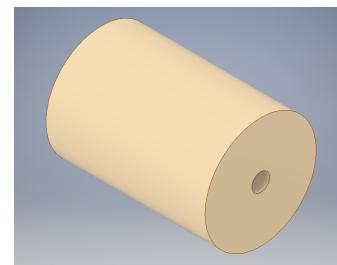


Abbildung 3.53: Walze Inventor

Die Feder drückt die beiden Walzen aneinander um die Futterpackung die durchgezogen wird auszuquetschen. Ohne die Feder und Walze könnte zu viele Reste in der Packung bleiben und so das ganze Futter verschwenden. Die Feder wurde von uns konstruiert und auf eine eigne Konstruktion gehängt die unabhängig von der Welle ist, damit die Feder nicht mitgedreht wird, sondern fix an einem Platz ist an den sie ihre Arbeit verrichtet. Siehe Abbildung: 3.54.

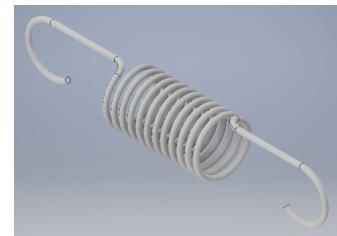


Abbildung 3.54: Feder Inventor

### 3.7.4 Hebelklemme

Die Hebelklemme dient zum Optimalen klemmen der Verpackung. Es ähnelt einen Marmeladen Verschluss von früher. Die Packung wird zwischen den zwei Platte eingelegt, danach wird der Schließmechanismus auf der oberen Platte 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 beim Stopper auftrifft, siehe gelber Kreis. Der Stopper hat die Aufgabe, dass der Hebel nicht zu weit geschlossen werden kann. Das hat zufolge das die Klemme leichter aufspringt. Siehe Abbildung: 3.55.

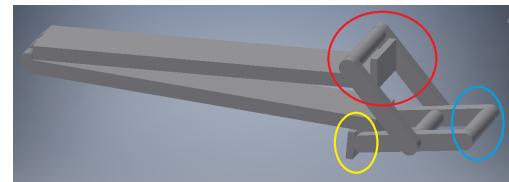


Abbildung 3.55: Hebelklemme Inventor

## 3.8 Berechnung und Dimensionierung

## 3.9 Simulation

## 3.10 Bedienung und Wartung

Bei der Entwicklung der Maschine wurde darauf geachtet, es Benutzerfreundlich als 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 befinden die 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 der Hebel in Richtung der Walze auf der rechten Seite befindet).
- 3 Mit einer Schere oder Messer die Schneidelinie durchtrennen und den Abfall entsorgen.
- 4 Die gewünschte Anzahl an Packungen festhängen, maximal 10 Packungen.

Weiter zu Wartung, nachdem der Benutzer aus dem Urlaub zurück ist sollte man folgenden 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 diese abwischen.
- 2 Die Futterschüsseln, diese können mit der Hand entnommen werden indem man auf der Unterseite der Platte die Schüssel nach oben drückt und mit der freien Hand die nimmt. Danach die Schüssel waschen und in die Platte zurück legen.
- 3 Die Futterplatte, mit einem nassen Tuch die Oberfläche nach dem entfernen der Schüsseln reinigen.

### **3.11 Selbstkritische Analyse und Ausblick**

Das Hauptproblem von meiner konstruierten Variante war das dichthalten der Packung mit einer Klemme die automatisch wenn das Förderband in betrieb ist leicht zu öffnen geht. Darum wurde die von uns konstruierte Hebel Klemme genommen. 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 das die Maschine von automatische gereinigt wird und womöglich eine Schneidefunktion einbauen die den Benutzer mehr entlasten könnte.

Wenn ich in Zukunft noch einmal eine Katzenfuttermaschine bauen würde, würde ich diese nicht aus Packungen, sondern aus den kleinen Metallfutterpackungen mit der Lasche als Deckel. Der Benutzer müsste dann nur die Laschen so weit nach oben biegen damit 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. Hat den Vorteil immer eine frische Schüssel zu haben und es können theoretisch so viele Packungen wie der Benutzer will in die Maschine eingelegt werden.

# KAPITEL 4

---

## **Elektronik und Mechanik**

---

### **4.1 Anforderung Elektronik**

#### **4.1.1 Aktoren**

Die Anforderung der Aktoren im Elektrischen Teil, bestehen darin ein Förderband und eine Drehplatte zu drehen. Für beide Anwendungen werden Motoren mit einem hohen Drehmoment und einer niedrigen Drehzahl benötigt. Weiters dürfen sich die Motoren nicht drehen, wenn sie nicht angesteuert werden.

#### **4.1.2 Sensoren**

Die Anforderung der Sensoren im Elektrischen Teil, bestehen darin erkennen zu können, ob ein Objekt vor ihnen steht. Dadurch kann überprüft werden, ob Futterschüsseln und Futterbeutel an der richtigen Stelle stehen, mögliche Fehler erkannt werden und Zählungen der Futterschüsseln durchgeführt werden, damit erfasst werden kann, welche der fünf Futterschüsseln an welcher Position steht.

#### **4.1.3 Ansteuerungen**



# **Anhang**



# ANHANG A

## Abbildungsverzeichnis

1.1	Angular Struktur . . . . .	2
1.2	Startseite . . . . .	4
1.3	Fütterungszeiten . . . . .	4
1.4	Geräteinformationen . . . . .	4
1.5	Update . . . . .	5
2.1	Abhängigkeiten . . . . .	12
2.2	Pin Numbering Sheme . . . . .	13
2.3	MainWindow . . . . .	23
2.4	TimeManagement . . . . .	26
2.5	Spinner Konfiguration . . . . .	27
2.6	CreateUser . . . . .	29
2.7	ManualControl . . . . .	31
2.8	Positionsinformation . . . . .	33
2.9	Update suchen . . . . .	36
2.10	Update verfügbar . . . . .	36
3.1	Magazin Modellaufbau von Vorne . . . . .	41
3.2	Magazin Modellaufbau von der Seite . . . . .	42
3.3	Magazin Modellaufbau von Oben . . . . .	42
3.4	Magazin Auszug . . . . .	42
3.5	Magazin Auszug Mitte . . . . .	42
3.6	Schneidebereit . . . . .	42
3.7	Schnitt . . . . .	44
3.8	Ausquetschen Beginn . . . . .	44
3.9	Ausquetschen Mitte . . . . .	45
3.10	Ausquetschen Ende . . . . .	45
3.11	Auswurf Beginn . . . . .	45
3.12	Bolzen drinnen . . . . .	45
3.13	Bolzen entfernen . . . . .	45
3.14	Klappe öffnen . . . . .	46
3.15	Fertiger Auswurf . . . . .	46

3.16	Loch Futterschüssel . . . . .	46
3.17	Foerderband . . . . .	47
3.18	Kettenglied . . . . .	47
3.19	Walze . . . . .	48
3.20	Scharnier . . . . .	48
3.21	Futterplatte . . . . .	48
3.22	Halterung . . . . .	49
3.23	Entleerung Anfang . . . . .	49
3.24	Entleerung nach 5min . . . . .	49
3.25	Entleerung nach 10min . . . . .	49
3.26	Einlegen . . . . .	50
3.27	Anfangsschnitt . . . . .	50
3.28	Endschnitt . . . . .	51
3.29	Schneidemittel . . . . .	52
3.30	Anfangsschnitt 2.Art . . . . .	52
3.31	Mittelschnitt 2.Art . . . . .	52
3.32	Endschnitt 2.Art . . . . .	52
3.33	3D-Klemme . . . . .	53
3.34	Klemmen Probe . . . . .	53
3.35	Klemme Tag 1 . . . . .	53
3.36	Klemme Tag 2 . . . . .	54
3.37	Klemme Tag 3 . . . . .	54
3.38	Klemme Tag 4 . . . . .	54
3.39	Klemme Tag 5 . . . . .	54
3.40	Einfache Klemme . . . . .	55
3.41	Hebel Klemme . . . . .	55
3.42	Gummiband Klemme . . . . .	55
3.43	Drehplatte . . . . .	56
3.44	Platte Zylinder . . . . .	56
3.45	Einschüsselplatte . . . . .	57
3.46	Futtermagazin Horizontal . . . . .	57
3.47	Futtermagazin Vertikal . . . . .	58
3.48	Drehplatte Inventor . . . . .	58
3.49	Futterschüssel Inventor . . . . .	58
3.50	Kette Inventor . . . . .	59
3.51	Kettenglied Inventor . . . . .	60
3.52	Kettenrad Inventor . . . . .	60
3.53	Walze Inventor . . . . .	61
3.54	Feder Inventor . . . . .	61
3.55	Hebelklemme Inventor . . . . .	61

## ANHANG B

---

### Tabellenverzeichnis

---

2.1 Belegung der GPIO-Pins . . . . .	14
2.2 Übertragungsprotokoll . . . . .	16



# ANHANG C

---

## Listings

---

1.1	*ngFor Beispiel . . . . .	3
2.1	Mit Mongodb verbinden . . . . .	11
2.2	Collection auswählen . . . . .	11
2.3	Anzahl der Collections zählen . . . . .	11
2.4	Nach Dokument suchen . . . . .	12
2.5	Ein Dokument hinzufügen . . . . .	12
2.6	Ein Dokument updaten . . . . .	12
2.7	Verbindung zur Datenbank trennen . . . . .	12
2.8	Konkretes Beispiel: Dokument suchen . . . . .	12
2.9	GPIO-Controller erstellen . . . . .	14
2.10	Zugriff auf einen Pin als Inpput . . . . .	14
2.11	Zugriff auf einen Pin als Output . . . . .	15
2.12	Pinzustand abfragen . . . . .	15
2.13	Pinzustand verändern . . . . .	15
2.14	Controller herunterfahren . . . . .	15
2.15	Error setzen . . . . .	18
2.16	Json-Objekt mit Errors und Warnungen . . . . .	18
2.17	Listenmodel-Klasse erstellen . . . . .	19
2.18	Liste erstellen im Model . . . . .	19
2.19	Liste erstellen . . . . .	19
2.20	Anzahl der Elemente abfragen . . . . .	19
2.21	Element hinzufügen . . . . .	19
2.22	Liste leeren . . . . .	19
2.23	SwingWorker Klasse erstellen . . . . .	20
2.24	SwingWorker doInBackground() mit while Schleife . . . . .	20
2.25	SwingWorker doInBackground() ohne while Schleife . . . . .	21
2.26	SwingWorker done() . . . . .	21
2.27	SwingWorker process() . . . . .	21
2.28	SwingWorker abbrechen . . . . .	21
2.29	MainWindow createInstance() . . . . .	23
2.30	MainWindow createInstance() Aufruf . . . . .	24

2.31 MainWindow.getInstance()	24
2.32 GUI Elemente blockieren	25
2.33 Spinner Zeitzone	27
2.34 Spinner Event	27
2.35 Zeitendokument Getter-Methode	28
2.36 Zeitendokument	28
2.37 char zu String	29
2.38 Hash Passwort	30
2.39 Textfeld Event	30
2.40 Benutzerdokument	30
2.41 Benutzerdokument Getter-Methode	31
2.42 Motoren stoppen und Fenster schließen	32
2.43 Motoren drehen	32
2.44 Label aktualisieren	33
2.45 Motor- und Sensorzustände	34
2.46 Update	36

## ANHANG D

---

### Abkürzungsverzeichnis

---

<b>API</b>	Application Programming Interface .....	3
<b>HTML</b>	HyperText Markup Language .....	3



## **ANHANG E**

---

### **Literaturverzeichnis**

---