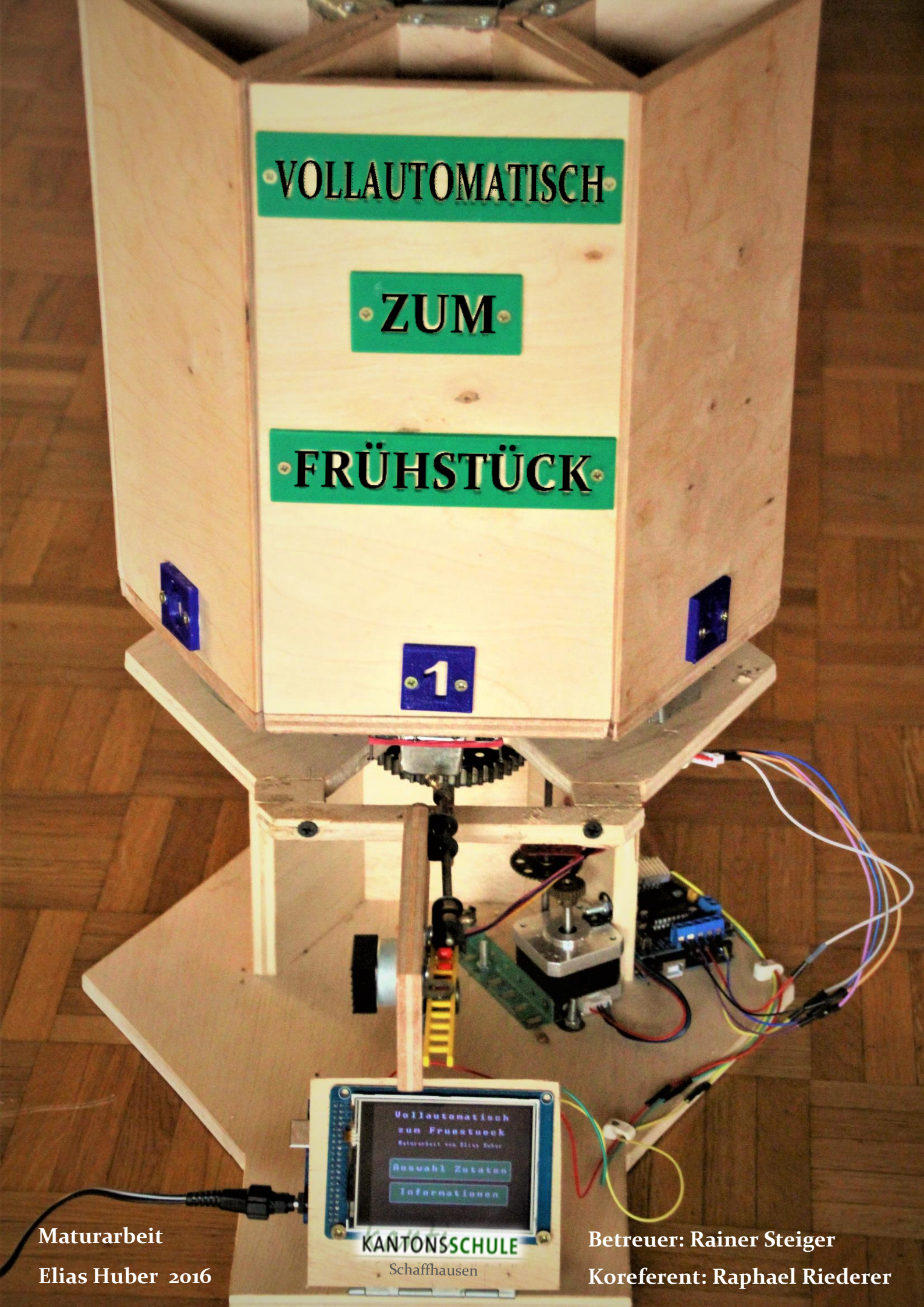



Maturarbeit

Elias Huber 2016

KANTONSSCHULE
Schaffhausen

Betreuer: Rainer Steiger

Koreferent: Raphael Riederer

# Vorwort

Herzlich Willkommen zu meiner Maturarbeit!

Im Rahmen dieser ging es mir darum, eine Müslimaschine zu entwickeln, welche nach einer Auswahl von Zutaten mittels Touchscreen vollautomatisch ein entsprechendes Müsli zusammenmischt und dem Benutzer die enthaltenen Nährwerte mitteilt. Der Grundgedanke beinhaltet dabei, dass für die Auswahl lediglich zwei Motoren verwendet werden, mit welchen man sechs verschiedene Zutaten ansteuern kann.

Während des Arbeitsprozesses stand die Entwicklung eines für den Benutzer ansprechenden und faszinierenden Prototyps im Vordergrund, welcher möglichst kostengünstig, auf Grundlage von Arduino Boards und handwerklichen Mitteln, eigenständig entwickelt wurde. Dabei war der Werdegang, von der ersten Idee über viele Versuche und auch Misserfolge zur fertigen Apparatur zentral.

Ich habe beim Schreiben der Arbeit versucht, dem Leser auch einen Einblick in die Probleme und nötig gewesenen Kompromisse zu geben. So richtet sich die folgende Arbeit nicht nur an Technikfreaks, eher im Gegenteil. Für mich selber war das Anpacken dieser Arbeit ein bisschen wie ein Sprung ins kalte Wasser, in ein Gebiet, in dem ich wenig Vorwissen und gar keine Erfahrung, aber schon immer ein Interesse gehabt habe. Ich habe selber während der Arbeit eine Menge unterschiedlichster Erfahrungen gesammelt und habe so versucht, ein paar meiner Konflikte und Erlebnisse, sofern es mir angebracht erschien, mit einfliessen zu lassen. Die Arbeit richtet sich also an jeden, der ein gewisses technisches Interesse besitzt und bereit ist, eine sicher nicht perfekte, aber hoffentlich nicht langweilige Arbeit zu lesen.

**Inhalt**

# 1. EINLEITUNG

## 1.1 ÜBERBLICK

Die Grundidee meiner Maturarbeit ist es, eine Maschine zu entwickeln, welche dem Bediener eine einstellbare Menge und Auswahl an Müslizutaten zusammenmischt und ihm eine Übersicht über die enthaltenen Nährwerte liefert. Dabei war es mir von Anfang an ein wichtiger Punkt, die ganze Apparatur mit nur zwei Motoren zu betreiben und trotzdem aus sechs verschiedenen Zutaten wählen zu können. Dies aus Gründen der Originalität und Einmaligkeit der Maschine sowie aus Kostengründen.

Die Bedienung soll leicht verständlich sein und das ganze Gerät soll über längere Zeiträume wartungsfrei nutzbar sein. Der gesamte Betriebsvorgang läuft selbstständig ab und die Maschine füllt das fertig gemischte Müsli nach Beendigung des Vorgangs in eine von dem Benutzer bereitgestellte Schale.

## 1.2 INSPIRATION

Schon von klein auf war ich begeistert für Technik und habe früher davon geträumt, alle möglichen Dinge zu erfinden. Auch bin ich recht sportbegeistert und habe zeitweise sehr auf meine Ernährung geachtet. Dannzumal habe ich mir oft morgens ein Müsli zubereitet, wobei ich zur Abwechslung verschiedene Mischungen ausprobiert habe. In Anbetracht dieser zwei Interessen war ich von dem Einfall, eine Müslimaschine zu bauen, sofort überzeugt.

## 1.3 ZIELSETZUNG

Ich habe mir für meine Maturarbeit vor allem folgende Ziele gesetzt:

- Zum einen möchte ich einen funktionalen «Müsliroboter» bauen, welcher nach Beendigung des Projekts nicht in einem Lager verstaubt, sondern welcher das Frühstück real erleichtert, Zeit spart und entsprechend genutzt werden kann.
- Ausserdem möchte ich zeigen, dass es möglich ist, ohne umfassende Vorkenntnisse und Ausrüstung ein solches Projekt zu verwirklichen, weder mit extern hergestellten Spezialanfertigungen, noch unter hohem finanziellen Aufwand, vor allem durch handwerkliches Geschick und Kreativität.
- Es geht mir jedoch keinesfalls darum, die perfekte Vorgehensweise beim Entwickeln eines solchen Projekts zu zeigen und irgendwelche Konzepte und Herangehensweisen darzustellen, welche ich letztendlich eh nicht benutzt habe.

Vielmehr ist es mein Ziel, den Entwicklungsprozess real und (auch für absolute Laien) nachvollziehbar darzustellen, mit (fast) allen Kompromissen und menschlichen Fehlern die dabei entstanden.

- Der Fokus auf den Werdegang soll auch beim fertigen Automaten ersichtlich sein, so wird bewusst an einigen Stellen Einblick in sein spannendes «Innenleben» gewährt. Er soll somit nicht wie ein fertiges Konsumentenprodukt erscheinen, sondern als der Prototyp, welcher er ist.

# 2. GRUNDLEGENDES

In den folgenden Abschnitten werde ich einige zentrale Bestandteile meiner Maturarbeit zum besseren Verständnis etwas genauer erklären. Alle kursiv markierten Wörter sind im Glossar erläutert, einige Begriffe werden auch direkt bei ihrer Erwähnung erklärt werden. Dies soll Lesern, welche mit der Materie vertraut sind, Zeitaufwand ersparen, die folgende Arbeit aber auch für alle andern Leser verständlich sein.

## 2.1 ARDUINO

### 2.1.1 DAS KONZEPT

Die Arduino Soft- sowie Hardware bilden das Herzstück meiner Maturarbeit, weshalb ich hier erläutern möchte, was es damit auf sich hat.

Arduino (ausserhalb USA u. UK offiziell Genuino) ist eine Open-Source Mikrocontroller Umgebung und enthält Hardware als auch Software.[1] Es stellt also sowohl ein Programmiertool zur Verfügung, als auch die sogenannten Arduino-Boards, welche es auch von anderen Anbietern in unterschiedlichsten Fassungen gibt. Das erste Arduino kam 2005 auf den Markt, und war gedacht als kostengünstiges Ansteuerungsgerät von Sensoren und Motoren, welches auch für Laien gut verständlich ist. Es ist heute eine der beliebtesten Plattformen zum (selbst-) Bauen von Robotern und zum Tüfteln generell[2].

1

Abb. 1: Eine Auswahl von Arduino Boards

### 2.1.2 DIE SOFTWARE

Die Arduino Software basiert auf der Programmiersprache C/C++, wobei das Interface für bessere Anfängerfreundlichkeit optisch vereinfacht wurde. Eine grosse Hilfe bieten ausserdem zahlreiche Bibliotheken (libraries) welche zusätzliche Funktionen, zum Beispiel speziell für die Ansteuerung eines Motors, bereitstellen. [3]

[1] Quelle : http://www.arduino.org/?gclid=CMaez6Lm9c4CFUK4Gwod8IcJQg
[2] Quelle : https://en.wikipedia.org/wiki/Arduino#Other_IDE
[3] Quelle : https://de.wikipedia.org/wiki/Arduino_(Plattform)

Genauere Details zum Code-Aufbau, Code-Beispiele sowie Erklärungen folgen im Kapitel Arduino Sketches.

### 2.1.3 BOARDS UND SHIELDS

Wie das Bild (Abb. 1) verdeutlicht, gibt es die unterschiedlichsten Boards mit allen denkbaren Anwendungsbereichen. So gibt es zum Beispiel eines, welches man in Kleider einnäht und welches dort Messungen vornimmt ((1) vorherige Seite).[4] Ich werde mich allerdings auf eine kurze Vorstellung des Arduino Unos beschränken.

Arduino Shields sind Boards, welche man in der Regel einfach auf das Arduino aufstecken kann. Sie sind meistens für spezifische Anwendungsbereiche gedacht. So habe ich in meiner Arbeit zum Beispiel ein Motorshield verwendet, welches eine separate Stromversorgung der Elektromotoren ermöglicht.

Das Arduino Uno ist das Basic Arduino Board und ich werde hier einen Teil der vorhandenen Hardware anhand des nebenstehenden Bildes aufzeigen.

So ist der USB Port(1) zu sehen, welcher zum einen der Datenübertragung zwischen dem Arduino und dem Computer dient, aber gleichzeitig auch als Powerquelle (5V) fungieren kann. Direkt darüber ist der Reset-Button(2), der bei Druck das laufende Programm abbricht und neu startet. Für separate Stromzufuhr befindet sich unten links der Netzstecker(3) welcher mit einer Spannung von 9-12V arbeitet, die bei Bedarf in den Transformatoren(4) umgewandelt werden kann. In der oberen Pin-Reihe befinden sich die Digital-Pins(5), welche *digitale* Signale sowohl lesen als auch senden können. In der unteren Reihe befinden sich zusätzlich zu den Power-Pins(6) (z.B. Erdung) die analogen Pins(7), welche *analoge* Signale senden und empfangen können.

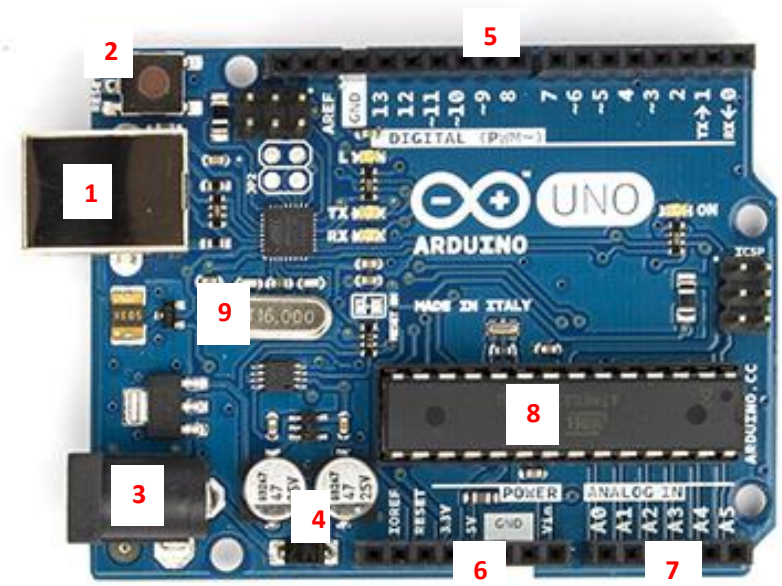



Abb. 2: Arduino Uno

[4] Quelle : http://www.play-zone.ch/de/elektronik-kit-zubehoer/avr-arduino-freeduino/boards-kompatibel/adafruit-flora-wearable-electronic-platform-v2.html

Letztlich gibt es noch den Mikrocontroller(8), dessen Frequenz von einem Quarz(9) geliefert wird.[5]

Spezifischeres sowie Anwendungen zu einigen Arduino Boards und Shields sind ebenfalls im weiteren Verlauf der Arbeit zu finden.

[5] Quelle : https://www.arduino.cc/en/Main/arduinoBoardDiecimila

## 2.2 MOTOREN

Da ich für das Mischen sowie für die Auswahl der Zutaten Elektromotoren verwendet habe, möchte ich die für mich wichtigsten zwei Motortypen kurz vorstellen.

### 2.2.1 GLEICHSTROM-MOTOR

Der Gleichstrommotor (auch DC-Motor oder Kommutatormotor) ist der wohl simpelste Elektromotor, welchen man auch aus dem Physikunterricht kennt. Er kann auch unter Belastung anlaufen und seine Drehzahl ist leicht veränderbar. Man kann ihn auch als Gleichstromgenerator verwenden.[6]

Zentral für das Funktionieren dieses Motors ist ein Magnetfeld, welches sowohl durch einen Permanentmagneten, als auch durch eine Spule erzeugt werden kann.

### 2.2.2 SCHRITTMOTOR

Ein Schrittmotor (auch Stepper) kann sich um eine feste Schrittgrösse und deren Vielfache drehen, aufgrund eines schrittweise rotierendem, elektromagnetischen Feldes[7]. Damit bietet er den Vorteil, dass sich identische Bewegungen exakt nachvollziehen und mehrfach ausführen lassen. Die einzelnen Magnete müssen allerdings separat angesteuert werden.[8]

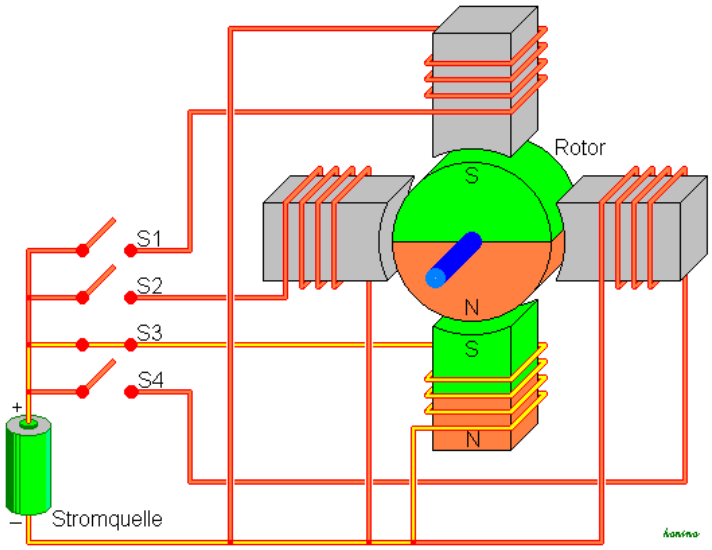



Abb. 3: Schema eines (unipolaren) Schrittmotors mit vier Schritten für eine volle Umdrehung.

Es besteht die Möglichkeit eines Überspringens von Schritten durch eine hohe (externe) Belastung, was sich z.B. durch einen Positionsgeber verhindern lässt.[9] Man unterscheidet hauptsächlich zwischen uni- und bipolaren Motoren, die Unterschiede liegen in der Ansteuerung der Spulen sowie der Anzahl der Pins.

---

[6] Quelle : https://de.wikipedia.org/wiki/Gleichstrommaschine
[7] Quelle : https://de.wikipedia.org/wiki/Schrittmotor
[8] Quelle : http://www.differencebetween.com/difference-between-stepper-motor-and-vs-dc-motor/
[9] Quelle : https://de.wikipedia.org/wiki/Schrittmotor

# 3. KONZEPT

In diesem Kapitel werde ich genauer darauf eingehen, wie der Automat aufgebaut ist, aus welchen Elementen er besteht und wie diese zusammenarbeiten. Dies soll lediglich einen Überblick geben, damit die folgenden Schritte nachvollzogen werden können. Details sind in den Kapiteln «Probleme – Lösungsansätze» sowie im Praxisteil zu finden.

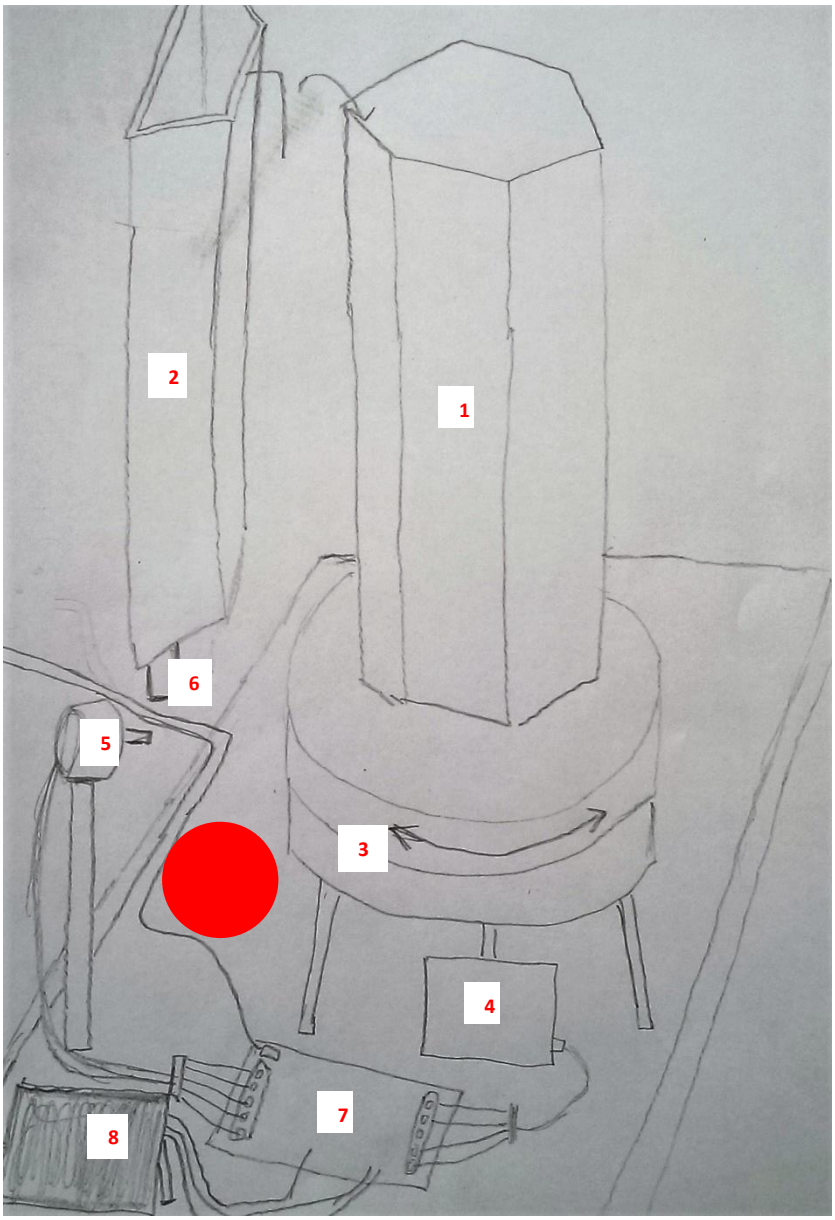



Abb. 4: «Skizze des geplanten Apparates

In der nebenstehenden Skizze habe ich mit meinen bescheidenen Zeichenkünsten versucht, dieses Konzept grafisch darzustellen. Zentral dabei ist der drehbare Turm(1), an welchen die sechs Zutatenbehälter(2), von denen ich hier zugunsten der Übersicht nur einen gezeichnet habe, angehängt werden. Dieser Turm ist über eine Art Kugellager(3) an der Grundplattform befestigt und wird durch den grossen Schrittmotor(4) gedreht. Dies macht es möglich, die Zutatenbehälter einzeln zur Position des kleinen Motors(5) zu drehen, welcher einen Öffnungsmechanismus(6) (Details Kap. 5.3) bedient, der einen Teil des Inhalts der Behälter in ein bereitgestelltes Schälchen fliessen lässt, hier als roter Kreis dargestellt. Anschliessend wird die Öffnung wieder verschlossen und der nächste Behälter zur «Entladung» in Position gedreht. Zusätzlich zu sehen ist noch das Motorshield (7) (Kap. 5.1) sowie der Touchscreen(8) für die Bedienung.

# 4. PROBLEME – LÖSUNGSANSÄTZE

In diesem Teil möchte ich die grössten Hürden aufzeigen, welche mir im Verlaufe der Arbeit begegnet sind und wie ich diese angegangen bin.

## 4.1 ANSTEUERUNG VON SOWOHL TOUCHSCREEN ALS AUCH MOTOREN MIT ARDUINO

Wie auf der Abbildung des Arduino Unos im Kapitel «Grundlegendes» zu sehen ist, hat dieses nur eine begrenzte Anzahl Pins und arbeitet normalerweise nur mit einer Spannung von 5V (bzw. 3V), auch die maximale Stromstärke sollte nicht über 500 mA liegen.[10] Für die Ansteuerung von zwei Motoren als auch eines Touchscreens ist es entsprechend unterausgerüstet. Die Lösung: Zwei Arduinos müssen her, eines kümmert sich um die Motoren, das andere um den Touchscreen.

Wie diese Boards sich genau um die Komponenten «kümmern», wird im Folgenden erklärt.

### 4.1.1 BEDIENUNG VON ZWEI MOTOREN ÜBER EIN MOTORSHIELD

Wie zuvor schon kurz erwähnt, gibt es sogenannte «Shields», welche das Standard Arduino um zahlreiche Funktionen erweitern können. Ich habe mir ein Shield speziell für die Ansteuerung von Motoren zugelegt. Es entspricht dem recht bekannten Adafruit motorshield v.1, welches hier abgebildet ist und unterscheidet sich eigentlich nur dadurch, dass keine Blume etwas rechts oben von der Mitte abgebildet ist und dass es viel billiger ist. Der Nachteil ist jedoch, dass man im Falle eines Problems keine Garantie hat sowie dass bei Kompatibilitätsproblemen nur schwer Informationen zu finden sind.

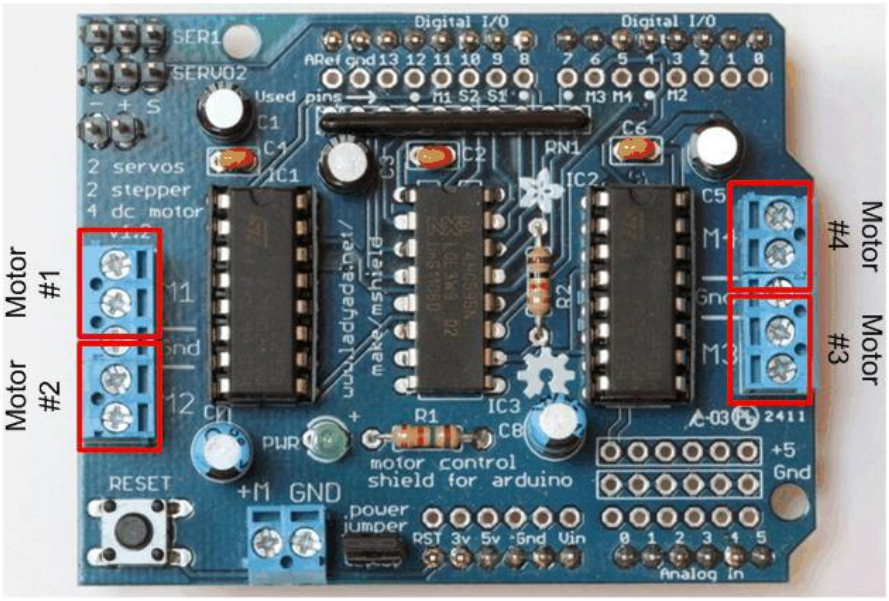



Abb. 5: Bild des Adafruit Motorshields v. 1

An den mit «Motor» beschrifteten Stellen kann man insgesamt vier Gleichstrommotoren anschliessen oder je zwei dieser Anschlussstellen für einen Schrittmotor benutzen (welcher je vier bzw. fünf Pins benötigt). Zusätzlich befinden sich oben links noch zwei Anschlüsse für

[10] Quelle: http://www.electricrcaircraftguy.com/2014/02/arduino-power-current-and-voltage.html

Servomotoren und die zwei *Schraubklemmen* unten können für einen separaten Stromanschluss verwendet werden.[11] Die Tatsache, dass das Shield auf das Arduino Uno aufgesteckt werden kann (wie in späteren Bildern noch ersichtlich wird) sowie dass eine extra Bibliothek für die Ansteuerung von Motoren mittels diesem Board verfügbar ist, macht die Bedienung mehrerer Motoren einfach und komfortabel. Die richtige Zuordnung der Pins von den Schrittmotoren zu den Motoranschlüssen stellte dabei das grösste Problem dar, denn vernünftige Datenblätter zu den Motoren suchte ich vergeblich, was die Zuordnung anfangs zu einem blossen Ratespiel reduzierte.

[11] Quelle: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-motor-shield.pdf

### 4.1.2 ANSTEUERUNG EINES TOUCHSCREENS MIT DEM ARDUINO MEGA

Etwas komplizierter wird das Ganze mit der Ansteuerung des Touchscreens, da zum einen der «Markt» für entsprechendes Equipment viel kleiner ist als bei Motoren und andererseits nochmals mehr Pins für die Datenübertragung auf den Bildschirm fällig sind. Das Pinproblem wird dadurch gelöst, dass statt dem normalen Arduino Uno ein Arduino Mega (2560) verwendet wurde, welches sich vor allem durch mehr Pins, einen schnelleren Prozessor sowie mehr Chipspeicher auszeichnet.[12] Erneut geschieht die Ansteuerung des Touchscreen über ein Shield, welches zwar diesmal nichts anderes macht, als die Pins des Touchscreens so zusammenzuführen, dass man ihn bequem aufstecken kann, es wäre sonst aber ziemlich mühsam und verwirrend, jeden *Jumper* einzeln vom Touchscreen zum entsprechenden Pin zu ziehen. Im Bild ist mein Set zu sehen, der Bildschirm hat eine Diagonale von 3,2 Zoll.


Abb. 6: Bild des bestellten Sets bestehend aus einem Arduino Mega (links), einem Shield für den Touchscreen (Mitte), einem 3,2'' Touchscreen (rechts) sowie einem USB Kabel (oben)

Für die Programmierung des Shields ist diesmal auf jedem Fall eine Bibliothek vonnöten, welche jedoch auf die Grösse und das Modell meines Touchscreens abgestimmt sein muss, was bei einem billigen No Name Produkt aus China nicht ganz einfach ist. Letztlich, auch dank der guten Dokumentation der empfehlenswerten «RinkyDink Electronics» libraries[13] sowie etwas Recherche im Internet konnte ich die benötigte Modellnummer schliesslich in Erfahrung bringen und die Ansteuerung verlief im Folgenden reibungslos.

[12] Quelle: https://www.arduino.cc/en/Products/Compare
[13] http://www.rinkydinkelectronics.com/library.php?id=65

## 4.2 I²C KOMMUNIKATION ZWISCHEN DEN ARDUINOBOARDS

Ein Problem welches sich nun jedoch noch ergibt, ist die Koordination zwischen den beiden Arduino Boards. Schliesslich soll das eine mit Hilfe des Touchscreens die Müslimengen beim Benutzer abfragen, welche dann irgendwie an das Arduino mit den Schrittmotoren weitergeleitet werden müssen, damit dieses die entsprechenden Portionen ausgeben kann. Dies habe ich mit Hilfe der I²C Datenübertragungsmethode (gesprochen: «i squared C»[14]) umgesetzt, deren Funktionsweise ich nun erläutere.

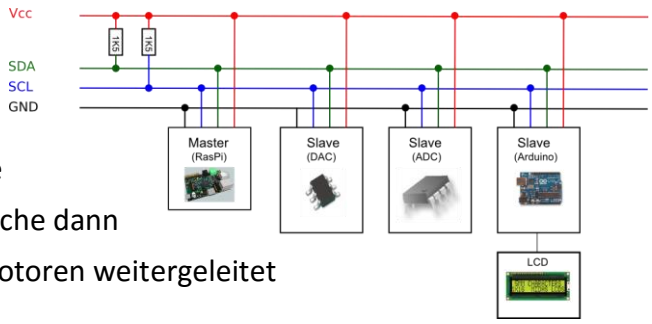



Abb. 7: Schema der Funktionsweise von I²C mit «Slaves» und «Masters»

Bei der I²C Kommunikation kann man theoretisch unendlich viele Geräte anschliessen (praktisch ist die Anzahl begrenzt, allerdings für fast alle Anwendungen mehr als ausreichend), wobei man zwischen «Master» und «Slave» Geräten unterscheidet. Als «Master» deklarierte Geräte bestimmen in der Regel die Frequenz der SCL Leitung (siehe nächster Abschnitt) und starten Übertragungen sowie geben Befehle, während «Slave» Geräte lediglich auf Anweisungen reagieren und auf Anfrage Daten senden können.

In der nebenstehenden Abbildung sieht man insgesamt vier Verbindungen, für die Datenübertragung sind eigentlich nur derer zwei benötigt. Die mit Vcc und GND beschrifteten Verbindungen stellen lediglich sicher, dass sowohl bei der für die Übertragung verwendeten Spannung (Vcc) als auch Erdung (GND) für beide Geräte gleiche Referenz-Werte gelten (damit mit «*HIGH*» und «*LOW*» beide Geräte dasselbe meinen). In meinem Fall benötige ich nur die GND Verbindung, da meine zwei kommunizierenden Geräte beide mit derselben Spannung arbeiten. Die SCL Verbindung ist die sogenannte «Clock-line», welche in regelmässigen Intervallen zwischen *LOW* und *HIGH* wechselt, während durch die SDA Verbindung die Daten übertragen werden. Die SDA Verbindung ist standardmässig, also ohne Eingreifen von einem der beiden Überträger, *HIGH*. Wie die Übertragung von Daten auf diese Weise funktioniert, möchte ich im nächsten Abschnitt aufzeigen.

Auf der nächsten Seite ist ein Schema zu sehen, wie eine solche Übertragung typischerweise abläuft. Um die Übertragung zu starten muss der Master eine «START condition» senden, dies geschieht dann, wenn er die SDA Verbindung auf *LOW* setzt, während die Clock-line sich auf *HIGH* befindet. Anschliessend muss der Master die 7-*bit* lange Adresse des angezielten

[14] Quelle : https://en.wikipedia.org/wiki/I%C2%B2C

Empfängers schicken. Das Senden dieser Adresse funktioniert identisch wie im Folgenden die Datenübertragung; der Master setzt die SDA Verbindung für jeden Ausschlag der Clock-line entweder auf *HIGH* oder *LOW* und sendet so die gewünschte Adresse, welche jedes angeschlossene Gerät ablesen kann. Dabei kann der Master den Zustand der SDA Verbindung nur ändern, wenn sich die Clock-line auf *LOW* befindet (sonst Komplikationen mit Start/Stop Kondition).

Zur Verdeutlichung ein kleines Beispiel: Der Master möchte eine Datenübertragung starten und setzt deswegen die SDA Verbindung auf *LOW,* während sich die Clock-line auf *HIGH* befindet. Um nun die Adresse des Empfängers zu senden, verändert der für die SDA Verbindung zuständige Pin des Masters seinen Output nach dem folgenden Muster: *HIGH*, *HIGH*, *LOW*, *HIGH*, *HIGH*, *LOW*, *LOW*. Alle angeschlossenen Empfänger lesen immer, während die Clock-line *HIGH* ist, den Wert der SDA Verbindung und lesen somit Folgendes: 1101100. Da zuvor eine START condition gesendet wurden, wissen die Geräte, dass es sich um eine Adresse handelt und entsprechend ob die Nachricht für sie gedacht ist oder nicht.

Nachdem die Adresse gesendet wurde, sendet der Master noch ein *Bit,* welcher angibt, ob der etwas an den Slave schicken oder von ihm lesen möchte (*LOW* = Write, *HIGH*=Read). Anschliessend muss der angesprochene Slave noch bestätigen, dass alles in Ordnung ist und er die Botschaft bekommen hat (siehe Grafik ACK(nowledge)), was er tut, indem er die SDA Verbindung während des nächsten Ausschlags der Clock-line auf *LOW* hält, dies wird «Acknowledge-bit» genannt. Ist dies geschehen, beginnt entweder der Master oder der Slave (je nachdem ob Read oder Write) mit der Datenübertragung, wobei nach jedem *Byte* der Leser mit einem «Acknowledge-bit» den Empfang bestätigen muss. Hat das sendende Gerät alles gesendet, beendet es die Übertragung mit einer «STOP condition», bei welcher er die SDA Verbindung auf *HIGH* zieht, während die Clock-line *HIGH* ist.[15]

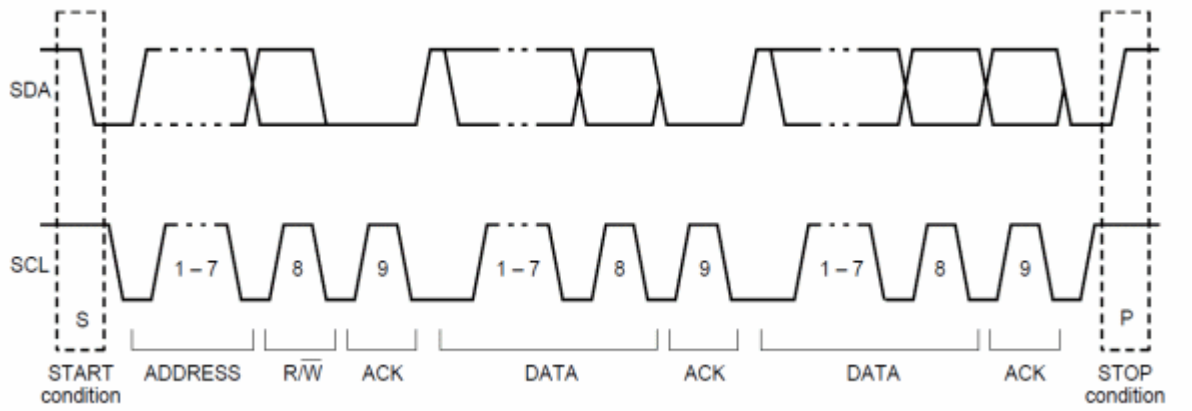



Abb. 8: Schema der Funktionsweise von einer Datenübertragung mit $I^2C$

[15] Quelle: http://i2c.info/i2c-bus-specification

## 4.3 KONZEPT DES ÖFFNUNGSMECHANISMUS

Dies war letztendlich wohl das langwierigste Problem, insgesamt drei Öffnungskonzepte boten sich an, welche hier mit ihren Pros und Kontras vorgestellt werden.

### 4.3.1 ÖFFNUNG MIT DREHRAD

Eine dieser Ideen war die eines Drehrads, welches die Zutaten portionsweise ausgibt. Die nebenstehende Skizze soll dies darstellen. Die Kraftübertragung von dem Motor auf die Öffnung kann man sich ähnlich wie bei einem Flachschraubenzieher vorstellen. In der Ausgangslage befände sich dieser waagerecht, wie auch in dieser Skizze der Fall (1). Dadurch ist es für den grossen Stepper möglich, den richtigen Zutatenbehälter (2) zum Öffnungsmechanismus zu drehen (vgl. Kap. Konzept). Einmal an Ort und Stelle würde sich der kleine Stepper dann um eine, von der ausgewählten Menge abhängige Anzahl Halbumdrehungen drehen (er muss den Vorgang ja waagerecht beenden) und so das Müsli portionsweise abgeben(3). Nach Abschluss dieses Vorgangs würde umgehend der nächste Behälter in Stellung gedreht werden.

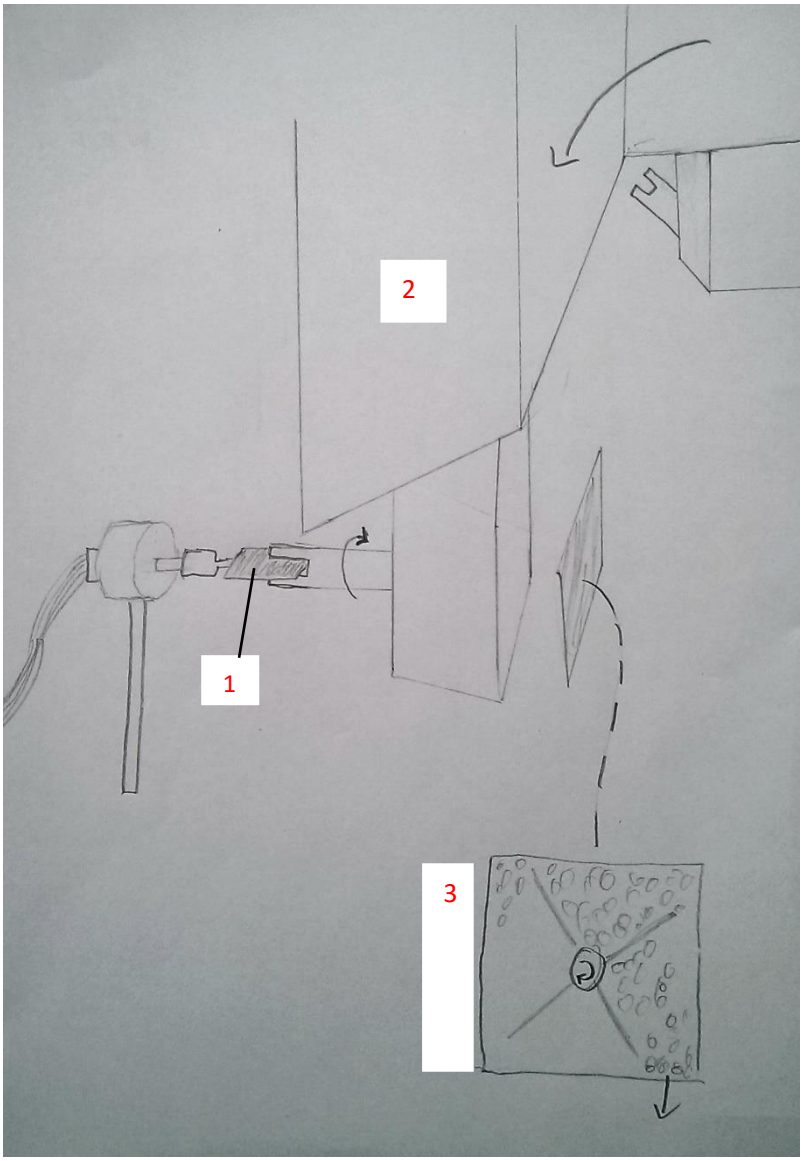



Abb. 9: Skizze des Öffnungsmechanismus mit Drehrad

Dieses Öffnungskonzept hätte den Vorteil, dass sich die ausgegebene Menge recht exakt angeben liesse, wobei sie jedoch nur in groben Schritten einstellbar wäre.

Letztendlich wurde dieses Konzept jedoch aus den folgenden Gründen verworfen:

- Es wären aufgrund der unterschiedlichen Grobkörnigkeit zutatenspezifische Drehräder vonnöten gewesen, was mit 3D-Druck geplant war. Erste Testversuche erwiesen sich diesbezüglich auch vielversprechend, da ich jedoch keinen eigenen 3D-

Drucker zuhause habe, wurde der Aufwand, für jede Zutat Testexemplare auszudrucken zu gross.

- Vor allem erwiesen sich jedoch Bedenken bezüglich der Drehkraft des kleinen Steppers und der Folgen einer möglichen Verklemmung von grobkörnigen Teilchen innerhalb des Drehrades.

### 4.3.2 ÖFFNUNG MIT EINEM SERVO ALS ZWEITEM MOTOR

Eine andere Idee, in welcher das benötigte *Drehmoment* vom kleinen Motor so klein wäre, dass sogar ein Servo verwendbar wäre, ist die folgende: An dem Servo wäre ein nach einer Stimmgabel aussehendes Objekt (1) befestigt, deren «Breitseiten» sich in der Ausgangslage wieder waagerecht befinden würden (nicht wie in Skizze!). Sobald sich der richtige Zutatenbehälter (2) in Position befände, würde sich diese Gabel um 90° drehen (wie in Skizze). Daraufhin würde sich der grosse Stepper erneut ein Stück drehen und damit eine Öffnung aufziehen (3). Nach anschliessendem, von der ausgewählten Menge abhängigem Warten, würde sich der grosse Stepper wieder zurückdrehen und so die Öffnung verschliessen, woraufhin sich die Gabel ebenfalls zurückdreht und der nächste Behälter in Position kommt. Ein Servo wäre einem Stepper insofern überlegen, dass er nur drei Anschlusspins benötigt,

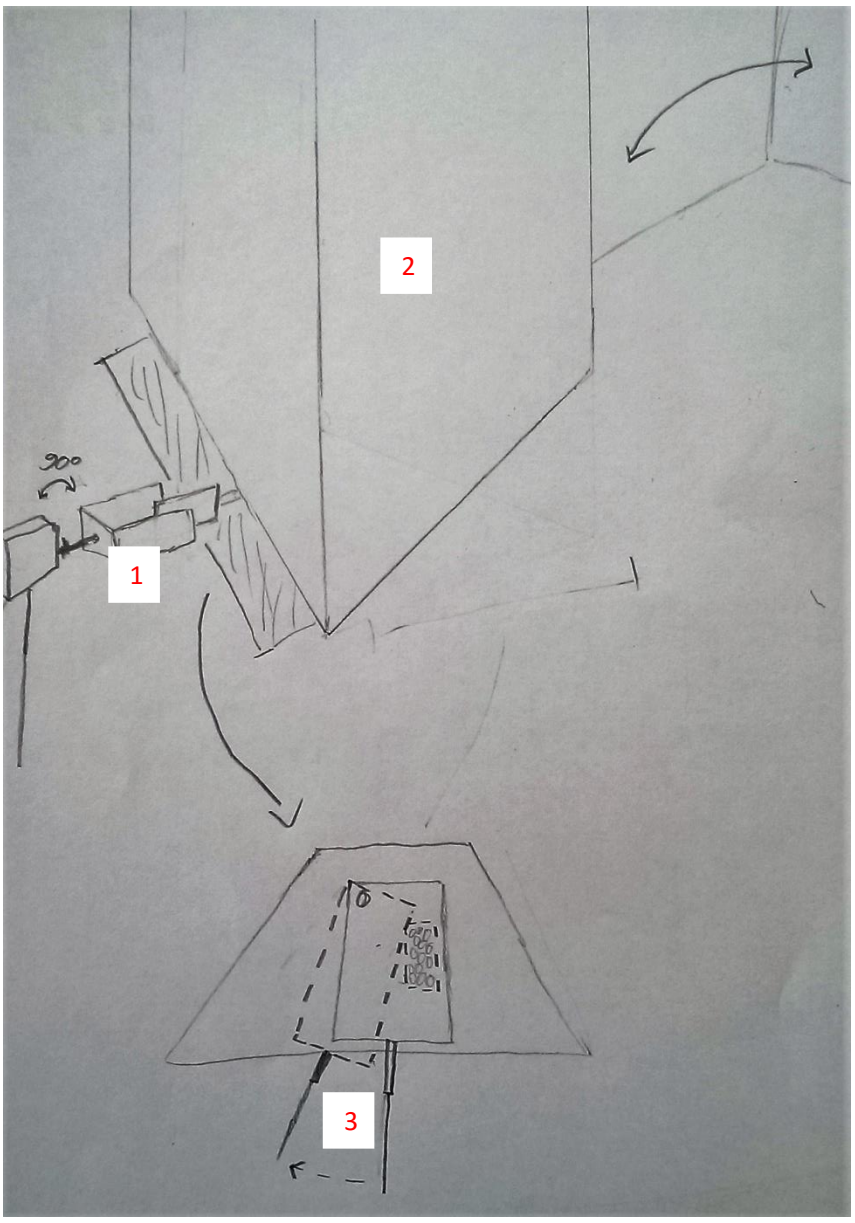



Abb. 10: Skizze des Öffnungsmechanismus mit Servo

weniger Strom zieht und einfacher zu programmieren ist. Er kann sich dafür nur um einen maximalen Winkel von 180° drehen und kaum Lasten bewegen.[16]

Auch dieses Konzept wurde verworfen, aus den untenstehenden Gründen:

- Die Bewegung wäre von den hier gezeigten die komplexeste.
- Es würde ein grosses *Drehmoment* auf dem grossen Stepper wirken, auch aus dem Grund, dass die zur Öffnung nötige Kraft an einem Punkt wirken würde, welcher mit einem grossen Hebelarm vom Schrittmotor entfernt läge. Der Motor selbst wäre zwar nicht der Flaschenhals, allerdings erlaubt das Motorshield nur Stromstärken von bis zu 0.6 A (für kurze Zeiten bis zu 1.2 A) und maximale Spannung von 25V[17] pro Motor, während der Motor eine Menge mehr Strom ziehen kann. Auf diese Problematik wird vertieft in Kapitel 4.4 eingegangen.

[16] Quelle : https://de.wikipedia.org/wiki/Servo
[17] Quelle : https://www.adafruit.com/products/81

### 4.3.3 ÖFFNUNG MIT «STÖSSEL»

Die letzte Idee, welche letztendlich auch in die Praxis umgesetzt wurde, ist die eines Stössels, welcher die Öffnung «aufstösst».

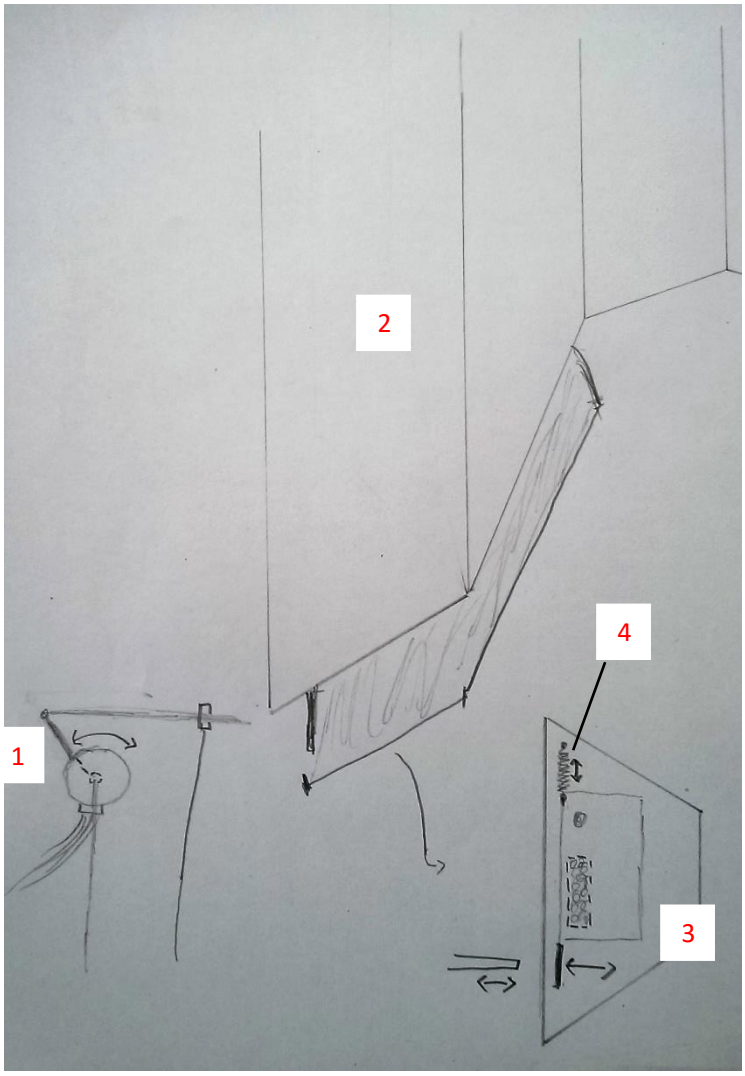


Abb. 11: Skizze des Öffnungsmechanismus mit Stössel

Um die Drehbewegung des kleinen Steppers in eine stossende umzuwandeln, war die ursprüngliche Idee, es wie hier in der Skizze (1) mit zwei Stangen nach demselben Prinzip wie bei einer Dampfmaschine zu realisieren. In der Praxis sieht die Sache natürlich wieder anders aus, aber dazu finden sie im übernächsten Kapitel mehr. Dieser Stössel soll dann bei dem, wie gehabt in Position gebrachten, Zutatenbehälter (2) einen Öffnungsmechanismus (3) aufstossen und nach Ablauf einer von der ausgewählten Menge abhängigen Zeit wieder schliessen. Der Öffnungsmechanismus funktioniert ähnlich wie im vorherigen Konzept, mit dem Unterschied, dass er sich diesmal durch eine Feder (4) oder einen Gummi von selbst wieder schliessen soll, sobald der Stössel weggezogen wird. Erneut wird nach Ablauf des Vorgangs der nächste Behälter in Position gedreht und der Vorgang wiederholt.

**Fazit:** Ob die definitive Entscheidung für diesen Vorgang die richtige Entscheidung war oder ob eine der anderen Varianten besser gewesen wäre, ist natürlich keinesfalls klar und wird in Kapitel 8 nochmal ausführlich diskutiert. Letztendlich musste ich mich aber nach bestem Gutdünken für ein Konzept entscheiden, da ich leider nicht die Ressourcen (je einzelne Motoren etc.) hatte, alle drei ausgiebig zu testen.

## 4.4 FOLGEN UND MASSNAHMEN BEZÜGLICH ZU HOHER STROMSTÄRKEN

Ein zu hoher Stromfluss war ein ständiges Sorgenkind während der Entstehung der Arbeit. In diesem Unterkapitel zeige ich mögliche Folgen einer zu hohen Stromstärke und Gegenmassnahmen, wobei ich mich hauptsächlich auf das Motorshield beziehe (für Leser mit keinerlei Vorwissen bezüglich Elektronik ist im Glossar unter *elektronische Grundlagen* ein kleiner physikalischer Exkurs zu finden).

Grundsätzlich liessen sich die zu erwartenden Stromstärken für den Betrieb des Automaten berechnen. Dies war in der Praxis allein wegen fehlenden Angaben zu den verwendeten Motoren jedoch nicht realisierbar. Ausserdem wäre für eine Berechnung der benötigten Stromstärke (und Spannung, welche ebenfalls vom Motorshield automatisch angepasst wird), neben der Drehgeschwindigkeit und Induktivität des Motors[18], auch das am Motor wirkende *Drehmoment* wichtig.[19] [20] Die Berechnung von diesem wiederum würde unter anderem Schätzungen zu den Schwerpunkten der Zutatenbehälter sowie vor allem Tests zum Drehwiderstand des verwendeten Spannlagers unter Belastung benötigen. Dies wäre für einigermassen stimmende Präzision zu umständlich gewesen.

Auch wenn eine genaue Schätzung der benötigten Stromstärken und Spannungen nicht realisierbar war, hätte ich mir bei der Bestellung eines Motorshieldes doch etwas mehr Gedanken machen und so ein grosszügiger dimensioniertes Shield kaufen können. Wobei die diesbezüglich vorhandenen Auswahlmöglichkeiten jedoch weit weniger populär und entsprechend schlechter dokumentiert sind. Dies wirft die Frage auf, ob ich mit so einem Shield nicht stattdessen Probleme bezüglich Libraries und Kompatibilität gehabt hätte, was wieder ein Argument für die Wahl der «gebräuchlichsten» Hardware ist.

**Mögliche Folgen zu hoher Stromstärken:** Das Hauptproblem bei hohen Stromstärken ist die starke Erhitzung der Mikrocontroller (H-Bridges[21]) auf dem Motorshield. Im besten Fall kommt es bei einer Überhitzung lediglich zum Abstürzen des Motorshield, im schlimmsten

[18] Quelle : http://www.wer-weiss-was.de/t/schrittmotor-mit-welchem-shield-betreiben/7310100/2
[19] Quelle : https://de.wikipedia.org/wiki/Schrittmotor#Kenngr.C3.B6.C3.9Fen
[20] Benötigte Angaben für Berechnung mittels Leistung $P = U * I$ = Drehmoment $*$ Winkelgeschwindigkeit (Quelle: https://de.wikipedia.org/wiki/Leistung_(Physik)#Mechanische_Leistung), bei bekannter Winkelgeschwindigkeit und entsprechend bekanntem ohmschen Widerstand (Mittels Induktivität des Motors und Winkelgeschwindigkeit) des Motors sowie des wirkenden Drehmoments ($P = U*I$, $U = R*I$ -> 2x2 Gleichungssystem)
[21] Genauere Angaben: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-motor-shield.pdf

Fall kann es jedoch auch zum Durchbrennen[22] kommen, was wiederum negative Auswirkungen auf das verbundene Arduino Uno haben kann.

**Massnahmen:**

- Als eine erste Massnahme gegen Überhitzung, kommt einem wohl eine Kühlung der betroffenen Komponenten in den Sinn. Dies habe ich mit «Kühlrippen» (engl. heat sinks) realisiert. Diese werden über den zu kühlenden Objekten (mit Wärmeleitpaste) angebracht und geben die Wärme der Objekte durch ihre grosse Austauschfläche mit der Luft und gute Wärmeleitfähigkeit schnell an die Umgebung ab. Das führt besonders bei kleinen kompakten Objekten wie Mikrocontrollern zu einer starken Verbesserung der Wärmeabgabe. Diese könnte man noch zusätzlich verbessern, indem man das Motorshield, statt es direkt auf das Arduino Board zu stecken, mit *Jumpern* anbindet und auf einer wärmeleitenden Unterfläche platziert. Tut man dies nicht, erwärmt sich die kaum zirkulierende Luft im Zwischenraum zwischen Board und Shield nämlich und macht eine Wärmeabgabe nach unten nur noch bedingt möglich. Diese Situation könnte auch mit einem Luftkühler verbessert werden. Weder Umplatzierung des Shields noch Luftkühlung wurden realisiert, Ersteres aus Platzgründen sowie dem zu erwartenden Kabelsalat und Letzteres aufgrund von zu kleinem Nutzen verglichen mit dem Aufwand.
- Eine zweite Massnahme war das Stapeln von Mikrocontrollern. Das Prinzip ist eigentlich einfach, zwei identische Mikrocontroller werden so übereinander platziert, dass die gleichen Pins aufeinanderliegen. Man schaltet sozusagen die Pins von zwei oder mehr Mikrocontrollern parallel. Entsprechend reduziert sich mit jedem weiteren so aufgesteckten Mikrocontroller der Stromfluss pro Mikrocontroller, wobei der Effekt pro weiterem Mikrocontroller von Mal zu Mal abnimmt (1->2: $I$ pro Mikrocontroller -50%, 2->3: $I$ pro Mikrocontroller -33.3, 3->4: $I$ pro Mikrocontroller -25% usw.). Durch diese Massnahme liess sich die Gefahr der Überhitzung eines Mikrocontrollers nochmals gewaltig reduzieren.
- Als eine letzte Massnahme um die «Peak current» zu reduzieren, habe ich für den grösseren der zwei Motoren (welcher das ganze Gerät dreht), aus Zahnrädern eine Übersetzung gebaut. Diese reduziert das maximal an ihm wirkende *Drehmoment* und damit die maximal benötigte Stromstärke.

[22] Quelle : https://forums.adafruit.com/viewtopic.php?f=31&t=26873

# 5. ARDUINO SKETCHES

Auf den folgenden Seiten werde ich den Code erklären, so wie er letztendlich auf die Arduino Boards geladen wurde. Ich werde den Code dabei in kleine Häppchen unterteilen und je zusammenfassend erklären, was in den entsprechenden Abschnitten geschieht, wobei sich spezifischere und «nicht interpretierte» Informationen auch im Code selbst auskommentiert befinden (also hinter //).

Da ich zwei Boards benutzt habe, ist dieses Kapitel entsprechend unterteilt, im Unterkapitel «Hauptsketch» befindet sich der Code für das Arduino Mega (der Master mit Touchscreen) und im Kapitel Empfangssketch der für das Arduino Uno (der Slave mit den Motoren).

## 5.1 HAUPTSKETCH

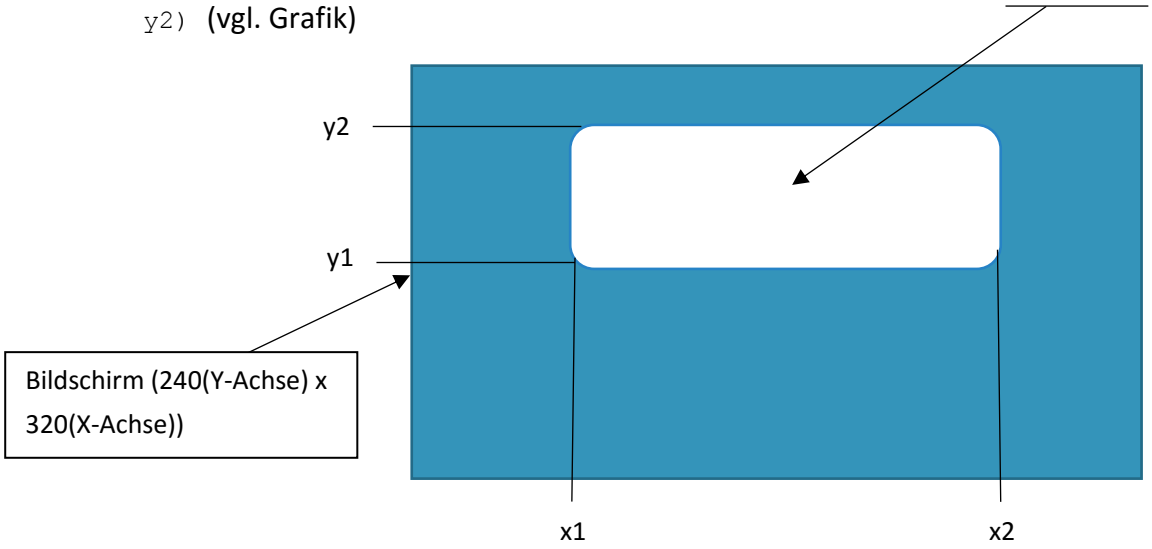
Kommen wir also zuerst zur Hauptsketch. Diese hat die Aufgabe, die Benutzeroberfläche des Touchscreens zu definieren, entsprechende Zutateneingaben entgegenzunehmen und diese an das Arduino Uno zu senden.

In diesem ersten Abschnitt des Codes passiert noch nichts Spannendes; es werden lediglich einige Voreinstellungen getroffen sowie später verwendete Variablen definiert. Die Pin Deklarationen sowie Angabe einer Modellnummer sind nötig, da die verwendete Library eine Vielzahl von Bords und Touchscreens unterstützt und somit durch die Angabe des Genannten speziell für meine verwendete Hardware konfiguriert wird.
Zu den Definitionen der Variable möchte ich nur auf «werte[]» und «currentPage» eingehen, der Rest erklärt sich am besten bei der Verwendung. Werte[] wie untenstehend definiert ist ein *Array,* welcher *Integer*s enthält, in ihm werden später die ausgewählten Zutatenmengen abgespeichert und an das Arduino Uno gesendet. Der *Character* «currentPage» wird im Folgenden als Indikator für die Seite dienen, auf welcher wir uns gerade befinden.

```
 //Die benötigten libraries
#include <Wire.h>                       //Für I²C Kommunikation
#include <UTFT.h>                       //Für Bildschirmanzeige
#include <URTouch.h>                    //Für Touchscreen spezifische
Funktionen


//Modellnummer meines Shields sowie boardspezifische Pin Deklarationen
UTFT    myGLCD(ILI9341_16,38,39,40,41);
//Boardspezifische Pin Deklarationen
URTouch  myTouch( 6, 5, 4, 3, 2);

//Die benutzten Schriftarten
extern uint8_t SmallFont[];             //kleine Schrift
extern uint8_t BigFont[];               //grosse Schrift
```

```
//Definition von Variablen
//Array der Bytes enthält (hier Zahlen von 0-256)
byte werte[6]={0,0,0,0,0,0};
int x,y;
String str0, str1,str2, str3, str4, str5; //Strings
int xA=38;                                //Integer
int xB=38;
int xC=38;
int xD=38;
int xE=38;
int xF=38;

//Für Kalorienrechner benötigte Werte und Variablen
String strkcal, strprot, strcarb, strfat = "0";

float kcalzt1pg =3.94;                    //Floats
float kcalzt2pg =3.85;
float kcalzt3pg =3.65;
float kcalzt4pg =4.86;
float kcalzt5pg =3.98;
float kcalzt6pg =3.12;

float protzt1pg =0.081;
float protzt2pg =0.057;
float protzt3pg =0.125;
float protzt4pg =.21;
float protzt5pg =.146;
float protzt6pg =.266;

float carbzt1pg =0.808;
float carbzt2pg =0.84;
float carbzt3pg =.66;
float carbzt4pg =.38;
float carbzt5pg =.62;
float carbzt6pg =.306;

float fatzt1pg =0.035;
float fatzt2pg =0.029;
float fatzt3pg =0.045;
float fatzt4pg =.31;
float fatzt5pg =.088;
float fatzt6pg =.092;

float totalkcal, totalprot, totalcarb, totalfat;


char currentPage;                         //Character
```

Im Folgenden kommt das Setup. Fast jede Arduino Sketch besteht aus einem Setup und einem Loop, wobei ersteres lediglich einmal (am Anfang oder wenn reset Taste gedrückt wurde)[23] und letzteres in unendlicher Abfolge durchläuft.[24] Neben einigen Initialisierungen im Setup definiere ich nach diesem vor allem verschiedene *Funktionen*, welche im späteren Verlauf das Vorgehen vereinfachen werden (das void indiziert dabei übrigens, dass die Funktionen keine Werte zurückgeben[25]). Die Funktionen sind durch ihre Namensgebung und die im Code gegebenen Erklärungen eigentlich alle selbsterklärend, nur drawHomeScreen sowie drawFrame seien hier kurz erläutert. DrawHomeScreen zeichnet, wie der Name schon indiziert, den Homescreen auf den Bildschirm. Dazu sei hier nur der Vollständigkeit halber erwähnt, dass alles in Verbindung mit dem dort gezeichneten Button «Informationen» hier aus dem Code entfernt ist und nicht weiter erläutert wird. Dies aus dem einfachen Grund, da die verwendeten Vorgehensweisen im Rest des Codes zu Genüge vorkommen und dass bei Berührung des Buttons lediglich ein paar Textzeilen auf den Bildschirm geschrieben werden.
Zur Funktion drawFrame: Als einzige verlangt diese Parameter bei ihrem Aufruf, der entsprechende Syntax ist im Code zu sehen und dient im Folgenden lediglich dazu, bei Berührung eines Buttons einen roten Rahmen um denselben zu malen.
Damit sie sich nicht nur durch trockenen Code lesen müssen und sich auch besser etwas darunter vorstellen können, finden sie nach der Definition der Homescreen Funktion ein Bild, welches zeigt was diese Funktion dann real macht.
Erwähnenswert ist möglicherweise noch die Koordinatenangabe bei den Funktionen, welche etwas auf den Bildschirm schreiben, da diese nicht unbedingt intuitiv ist. Sie funktioniert zum Beispiel für `drawRoundRect` folgendermassen: `myGLCD.drawRoundRect (x1, y1, x2, y2)` (vgl. Grafik)




```
//Setup welches einmal durchläuft
void setup() {
//macht das Board bereit für I²C Datenübertragung
Wire.begin();
//Einstellungen für den Touchscreen:
myGLCD.InitLCD();                    //initialisiert Bildschirm

//Löscht alles auf dem Bildschirm Angezeigte
```

[23] Quelle : https://www.arduino.cc/en/Reference/Setup
[24] Quelle : https://www.arduino.cc/en/Reference/Loop
[25] Quelle : https://de.wikipedia.org/wiki/Void_%28Schl%C3%BCsselwort%29

```
myGLCD.clrScr();
myTouch.InitTouch();                    //initialisiert Tuchscreen
//Legt Präzision für Touch-Eingabe fest
myTouch.setPrecision(PREC_MEDIUM);
drawHomeScreen();                       //Zeichnet Homescreen
currentPage = '0';                      //Zeigt an das wir im Homescreen sind
}

void drawFrame(int x1, int y1, int x2, int y2) {
  myGLCD.setColor(255, 0, 0);                          //setzt Farbe auf rot
//zeichnet Rechteckeck mit abgerundeten Ecken
  myGLCD.drawRoundRect (x1, y1, x2, y2;
}
void drawHomeScreen() {
                                                       //Titel
//Setzt Hintergrundfarbe auf schwarz
myGLCD.setBackColor(0,0,0);
myGLCD.setColor(255, 255, 255);                        //weiss
//Waehlt als Schriftgroesse "BigFont" aus
myGLCD.setFont(BigFont);
//Schreibt auf "Vollautomatisch" auf den Bildschirm
myGLCD.print("Vollautomatisch", CENTER, 10);
myGLCD.print("zum Fruestueck", CENTER, 41);
myGLCD.setFont(SmallFont);
myGLCD.print("Maturarbeit von Elias Huber" ,CENTER ,71);

//Button für die Auswahl der Zutaten
myGLCD.setColor(16, 167, 103);                         //grün
//Zeichnet ein ausgefülltes Rechteck zu den entsprechenden Koordinaten
myGLCD.fillRoundRect (35, 110, 285, 150);
myGLCD.setColor(255, 255, 255);
myGLCD.drawRoundRect (35, 110, 285, 150);
myGLCD.setFont(BigFont);
myGLCD.setBackColor(16, 167, 103);
myGLCD.print("Auswahl Zutaten", CENTER, 120);

//Button für weitere Informationen über diese die Maschine
myGLCD.setColor(16, 167, 103);
myGLCD.fillRoundRect (37,160,280,200);
myGLCD.setColor (255, 255, 255);
myGLCD.drawRoundRect (37,160,280,200);
myGLCD.setFont(BigFont);
myGLCD.setBackColor(16,167,103);
myGLCD.print("Informationen", CENTER, 170);

}
```

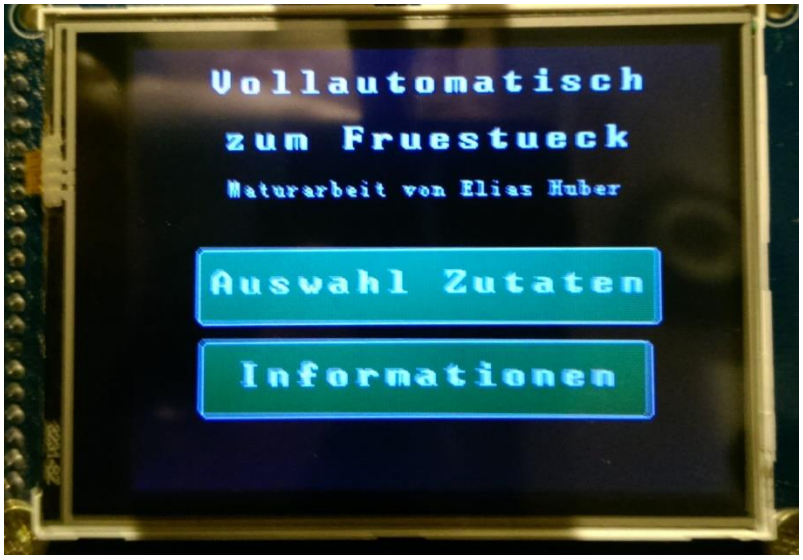



Abb. 12: Homescreen

```
//Funktion welche einen "Zurück Button" zeichnen soll
void drawZurueckButton () {
   myGLCD.setColor(16, 167, 103);
   myGLCD.fillRoundRect (10, 190, 140, 230);
   myGLCD.setColor(255, 255, 255);
   myGLCD.drawRoundRect (10, 190, 140, 230);
   myGLCD.setFont(BigFont);
   myGLCD.setBackColor(16, 167, 103);
   myGLCD.print("Zurueck", 20, 205);
  }

//Funktion welche einen "Weiter Button" zeichnen soll
void drawWeiterButton () {
   myGLCD.setColor(16, 167, 103);
   myGLCD.fillRoundRect (200, 190, 310, 230);
   myGLCD.setColor(255, 255, 255);
   myGLCD.drawRoundRect (200, 190, 310, 230);
   myGLCD.setFont(BigFont);
   myGLCD.setBackColor(16, 167, 103);
   myGLCD.print("Weiter", 210, 205);
  }

//Zeichnet ersten Auswahlscreen (vgl. drawHomeScreen)
void auswahl1() {
  drawWeiterButton();
  drawZurueckButton();
  myGLCD.setBackColor(0, 0, 0);
                                          //Titel
  myGLCD.print("Auswahl", CENTER, 10);
                                          //Zutatenbeschriftung 1
  myGLCD.setFont(SmallFont);
  myGLCD.setBackColor(0,0,0);
  myGLCD.print("Gepuffter Reis", 10, 60);
                                          //Zutatenbeschriftung 2
 myGLCD.print("Schoko Crispies", 10,105);
                                          //Zutatenbeschriftung 3
 myGLCD.print("Gepuffter Qinoa", 10,150);

}
//Zeichnet zweiten Auswahlscreen
void auswahl2() {
                                          //zeichnet "Misch Button"
  myGLCD.setColor(16, 167, 103);
  myGLCD.fillRoundRect (200, 190, 310, 230);
  myGLCD.setColor(255, 255, 255);
  myGLCD.drawRoundRect (200, 190, 310, 230);
  myGLCD.setFont(BigFont);
  myGLCD.setBackColor(16, 167, 103);
  myGLCD.print("Misch!", 210, 205);
  drawZurueckButton();

  myGLCD.setBackColor(0, 0, 0);
                                          //Zutatenbeschriftung 4
  myGLCD.print("Auswahl", CENTER, 10);
  myGLCD.setFont(SmallFont);
  myGLCD.print("Chia Samen", 10,60);
                                          //Zutatenbeschriftung 5

  myGLCD.print("Amaranth", 10,105);
                                          //Zutatenbeschriftung 6
  myGLCD.print("Weizenkeime", 10,150);
}
```

```
//Kalorienanzeige am Schluss

void drawKalorien() {
//"Fertig Button"
  myGLCD.setColor(16, 167, 103);
  myGLCD.fillRoundRect (200, 190, 310, 230);
  myGLCD.setColor(255, 255, 255);
  myGLCD.drawRoundRect (200, 190, 310, 230);
  myGLCD.setFont(BigFont);
  myGLCD.setBackColor(16, 167, 103);
  myGLCD.print("Fertig", 210, 205);
//Titel
  myGLCD.setBackColor(0, 0, 0);
  myGLCD.print("Deine Kalorien:", LEFT, 10);

//Beschriftung der Nährwerte-Slider
  myGLCD.setFont(SmallFont);
  myGLCD.print("Proteine", 10,60);
  myGLCD.print("Kohlenhydrate", 10,105);
  myGLCD.print("Fett", 10,150);

}
```

Damit wären wir auch schon beim Loop, dem Hauptteil der Arduinosketch. Da dieser ja im Gegensatz zum Setup in einer Endlosschleife läuft, können wir hier bewirken, dass das Arduino auf Einflüsse von aussen reagiert. Entsprechend sieht man auch gleich am Anfang eine if-Bedingung, deren enthaltener Code nur aktiv wird, wenn der Indikator für die Seite («currentPage») den richtigen Wert hat. Nach diesem Schema ist im Folgenden der ganze Loop aufgebaut, eine Bedingung für die Seitenzahl, die den entsprechenden Code enthält. Die Seite auf dem Bildschirm kann also nur geändert werden, wenn innerhalb des aktiven Codes der Wert des *Character*s currentPage geändert wird. In der ersten Schleife geschieht dies dann, wenn der Bildschirm an der Stelle des «Auswahl Buttons» berührt wird.

```
//"Loop" welcher im Gegenteil zum "Setup" nicht einmal, sondern unendlich
mal abläuft
void loop() {
 if (currentPage == '0') {                  //Wenn wir im Homescreen sind:
    if (myTouch.dataAvailable()) {          //Touchscreen betätigt wird:
      myTouch.read();                       //Liest ab wo gedrückt wurde...
//und speichert X Koordinate unter der Variable x ab...
      x=myTouch.getX();
//sowie die Y Koordinate unter der Variable y
      y=myTouch.getY();
//Wenn die gedrückten Koordinaten auf dem "Auswahl Button" liegen:
      if ((x>=35) && (x<=285) && (y>=110) && (y<=150)) {
//Zeichnet einen Rahmen um den "Auswahl Button"
        drawFrame(35, 110, 285, 150);
        currentPage = '1';               //Nimmt Wert für Auswahlseite 1 an
        myGLCD.clrScr();
        auswahl1;
     }

    }
   }

//Wenn wir auf der ersten Auswahlseite sind:
 if (currentPage == '1') {
    if (myTouch.dataAvailable()) {
        myTouch.read();
        x=myTouch.getX();
        y=myTouch.getY();
//Wenn "Weiter Button" gedrückt wird:
        if ((x>=200) && (x<=310) && (y>=190) && (y<=230)) {
          drawFrame(200, 190, 310, 230);
          //Nimmt Wert für Auswahlseite 2 an
          currentPage = '2';
          myGLCD.clrScr();
//Ruft zuvor definierte Funktion auf welche die Auswahlseite 2 zeichnet
          auswahl2();
          }

//Wenn "Zurück Button" gedrückt wird:
          if ((x>=10) && (x<=140) && (y>=190) && (y<=230)) {
          drawFrame(10, 190, 140, 230);
//Nimmt Wert für Homescreen an
          currentPage = '0';
          myGLCD.clrScr();
          drawHomeScreen();
          }
```

Im nachfolgenden Teil des Codes der ersten Auswahlseite werden nun die Auswahlslider gezeichnet und eingelesen. Da sich Slider waagerecht fast über die gesamte Bildschirmbreite erstrecken sollen, wird bei der If-Bedingung diesmal nur nach der gedrückten Höhe (Y Koordinate) gefragt. Dann muss jedoch noch sichergestellt werden, dass die berührten X-Koordinaten, wenn sie sich ausserhalb der Slider befinden den richtigen Wert bekommen. Natürlich hätte man die X-Koordinaten auch von Anfang an mit in die If-Bedingung nehmen können, dies hätte jedoch den Nachteil, dass es extrem schwer wäre, den höchstmöglichen Wert oder Null auszuwählen, da man dann genau den richtigen Pixel auswählen müsste. Im unten gezeigten Fall muss man nur auf der richtigen Höhe ganz links oder ganz rechts berühren, um gar nichts oder alles auszuwählen und kann somit insbesondere die Slider wieder gut auf Null zurücksetzen, wenn man sich vertippt hat.

```
//Auswahlslider A
//Wenn Bildschirm auf "Höhe" der Slider gedrückt wird:
        if( (y>=80) && (y<=106)) {
//Speichert X Koordinate unter der Variable xA ab
          xA=x;

//Falls xA kleiner oder gleich 38:
          if (xA<=38) {
            xA=38;
        }

//Falls xA grösser oder gleich 303
          if (xA>=303){
            xA=303;
        }

//Auswahlslider B
        if( (y>=120) && (y<=156)) {
          xB=x;
          if (xB<=38) {
            xB=38;
      }
          if (xB>=303){
            xB=303;
      }
    }

//Auswahlslider C
        if( (y>=160) && (y<=186)) {
          xC=x;
          if (xC<=38) {
            xC=38;
      }
          if (xC>=303){
            xC=303;
      }
    }
    }



//Nimmt den Wert xA, der sich auf einer Skala von 38 bis 310 befindet und
speichert ihn unter der Variable xAC als den Wert, den er auf einer Skala
von 0 bis 100 hätte
  int xAC = map(xA,38,303,0,100);
```

```
  int xBC = map(xB,38,303,0,100);
  int xCC = map(xC,38,303,0,50); //Auf Skala 0-50
//Speichert ihn in dem ersten Platz des Werte Arrays ab
  werte[0]= xAC;
//Speichert ihn in dem zweiten Platz...
  werte[1]= xBC;
  werte[2]= xCC;

myGLCD.setFont(SmallFont);
myGLCD.setBackColor(0,0,0);

//zeichnet Slider A
  myGLCD.setColor(255, 255, 255);
  myGLCD.fillRect(xA,89,(xA+4),97);              //Positionsanzeiger
  myGLCD.setColor(16, 167, 103);
//"ausgewählte Fläche" im Slider mit grüner Farbe füllen
  myGLCD.fillRect(31, 89, (xA-1), 97)
  myGLCD.setColor(255, 255, 255);
//"nicht ausgewählte Fläche" im Slider mit weisser Farbe füllen
  myGLCD.fillRect((xA+5), 89, 309, 97);
//speichert den Integer im Werte Array als String ab, da die print Funktion
//hier keine Integers akzeptiert
  str0= werte[0];
  myGLCD.print(str0, 200,60);

//zeichnet Slider B
  myGLCD.setColor(255, 255, 255);
  myGLCD.fillRect(xB,129,(xB+4),137);
  myGLCD.setColor(16, 167, 103);
  myGLCD.fillRect(31, 129, (xB-1), 137);
  myGLCD.setColor(255, 255, 255);
  myGLCD.fillRect((xB+5), 129, 309, 137);
  str1= werte[1];
  myGLCD.print(str1, 200,105);

//zeichnet Slider C
  myGLCD.setColor(255, 255, 255);
  myGLCD.fillRect(xC,169,(xC+4),177);
  myGLCD.setColor(16, 167, 103);
  myGLCD.fillRect(31, 169, (xC-1), 177);
  myGLCD.setColor(255, 255, 255);
  myGLCD.fillRect((xC+5), 169, 309, 177);
  str2= werte[2];
  myGLCD.print(str2, 200,150);
 }
```


Abb. 13: Auswahlscreen

Die nun folgende zweite Auswahlseite (für weitere Zutaten) ist grösstenteils identisch mit der ersten, mit dem Hauptunterschied, dass diesmal bei Berührung des «Fertig Buttons» der Werte *Array* an das Arduino Uno geschickt wird. Auf den Übertragungsprozess werde ich im Kapitel Empfangssketch noch genauer eingehen.

```
if (currentPage == '2') {
    if (myTouch.dataAvailable()) {
        myTouch.read();
        x=myTouch.getX();
        y=myTouch.getY();
        if ((x>=200) && (x<=310) && (y>=190) && (y<=230)) {
         drawFrame(200, 190, 310, 230);
```

```
//Beginnt Übertragung zu Gerät mit Adresse 9
          Wire.beginTransmission(9);
          Wire.write(werte[0]);       //sendet den Inhalt von werte[0]
          Wire.endTransmission();     //beendet Übertragung
          Wire.beginTransmission(9);
          Wire.write(werte[1]);
          Wire.endTransmission();
          Wire.beginTransmission(9);
          Wire.write(werte[2]);
          Wire.endTransmission();
          Wire.beginTransmission(9);
          Wire.write(werte[3]);
          Wire.endTransmission();
          Wire.beginTransmission(9);
          Wire.write(werte[4]);
          Wire.endTransmission();
          Wire.beginTransmission(9);
          Wire.write(werte[5]);
          Wire.endTransmission();

//Slider werden auf Null zurück gesetzt
          int xA=38;
          int xB=38;
          int xC=38;
          int xD=38;
          int xE=38;
          int xF=38;
//Kalorienanzeige wird aufgerufen
         currentPage = '3';
         myGLCD.clrScr();
         drawKalorien();
        }
//"Zurück Button"
     if ((x>=10) && (x<=140) && (y>=190) && (y<=230)) {
          drawFrame(10, 190, 140, 230);
          currentPage = '1';
          myGLCD.clrScr();
          auswahl1();
          }



        if( (y>=80) && (y<=106)) {
          xD=x;
          if (xD<=38) {
            xD=38;
        }
          if (xD>=303){
            xD=303;
        }
        }

        if( (y>=120) && (y<=156)) {
            xE=x;
          if (xE<=38) {
            xE=38;
      }
            if (xE>=303){
              xE=303;
      }
    }
        if( (y>=160) && (y<=186)) {
```

```
            xF=x;
          if (xF<=38) {
            xF=38;
      }
          if (xF>=303){
            xF=303;
      }
    }
    }
//Skalierung
  int xDC = map(xD,38,303,0,10);
  int xEC = map(xE,38,303,0,20);
  int xFC = map(xF,38,303,0,40);

  werte[3]= xDC;
  werte[4]= xEC;
  werte[5]= xFC;

  myGLCD.setFont(SmallFont);
  myGLCD.setBackColor(0,0,0);


//Slider D
  myGLCD.setColor(255, 255, 255);
  myGLCD.fillRect(xD,89,(xD+4),97); // Positioner
  myGLCD.setColor(16, 167, 103);
  myGLCD.fillRect(31, 89, (xD-1), 97);
  myGLCD.setColor(255, 255, 255);
  myGLCD.fillRect((xD+5), 89, 309, 97);
  str3= werte[3];
  myGLCD.print(str3, 200,60);

//Slider E
  myGLCD.setColor(255, 255, 255);
  myGLCD.fillRect(xE,129,(xE+4),137);
  myGLCD.setColor(16, 167, 103);
  myGLCD.fillRect(31, 129, (xE-1), 137);
  myGLCD.setColor(255, 255, 255);
  myGLCD.fillRect((xE+5), 129, 309, 137);
  str4= werte[4];
  myGLCD.print(str4, 200,105);

 //Slider F
  myGLCD.setColor(255, 255, 255);
  myGLCD.fillRect(xF,169,(xF+4),177);
  myGLCD.setColor(16, 167, 103);
  myGLCD.fillRect(31, 169, (xF-1), 177);
  myGLCD.setColor(255, 255, 255);
  myGLCD.fillRect((xF+5), 169, 309, 177);
  str5= werte[5];
  myGLCD.print(str5, 200,150);

          }
```

Im nun folgenden, letzten Teil der Hauptsketch wird der Kalorienrechner gezeichnet. Dabei passiert nichts Besonderes oder Kompliziertes, es werden lediglich aus den ausgewählten Mengen sowie den Nährwerten/g der einzelnen Zutaten die Gesamtnährwerte berechnet. Diese werden (aufgrund eines weiter hinten erklärten Bugs) auf den bereits mehrfach benutzten Slidern grafisch dargestellt.

```
if (currentPage == '3') {
//Zusammenzählen der jeweiligen Nährwerte
//Kilokalorien
     totalkcal = kcalzt1pg*werte[0] + kcalzt2pg*werte[1] +
kcalzt3pg*werte[2] + kcalzt4pg*werte[3] + kcalzt5pg*werte[4] +
kcalzt6pg*werte[5];
//Eiweiss
     totalprot = protzt1pg*werte[0] + protzt2pg*werte[1] +
protzt3pg*werte[2] + protzt4pg*werte[3] + protzt5pg*werte[4] +
protzt6pg*werte[5];

//Kohlenhydrate
     totalcarb = carbzt1pg*werte[0] + carbzt2pg*werte[1] +
carbzt3pg*werte[2] + carbzt4pg*werte[3] + carbzt5pg*werte[4] +
carbzt6pg*werte[5];
//Fett
     totalfat = fatzt1pg*werte[0] + fatzt2pg*werte[1] + fatzt3pg*werte[2] +
fatzt4pg*werte[3] + fatzt5pg*werte[4] + fatzt6pg*werte[5];

//Anzahl Kilokalorien hinter Titel schreiben
     myGLCD.setFont(BigFont);
     myGLCD.setBackColor(0, 0, 0);
     strkcal = totalkcal;
     myGLCD.print(strkcal,250, 10);

//Skalierung diesmal umgekehrt, von den Werten zu den Koordinaten
     int protslider = map(totalprot,0,150,38,303);
     int carbslider = map(totalcarb,0,150,38,303);
     int fatslider = map(totalfat,0,150,38,303);


     myGLCD.setFont(SmallFont);
     myGLCD.setBackColor(0,0,0);
//Slider für Eiweiss
     myGLCD.setColor(255, 255, 255);
     myGLCD.fillRect(protslider,89,(protslider+4),97);
     myGLCD.setColor(16, 167, 103);
     myGLCD.fillRect(31, 89, (protslider-1), 97);
     myGLCD.setColor(255, 255, 255);
     myGLCD.fillRect((protslider+5), 89, 309, 97);
     strprot= totalprot;
     myGLCD.print(strprot, 200,60);
  //Slider für Kohlenhydrate
     myGLCD.setColor(255, 255, 255);
     myGLCD.fillRect(carbslider,129,(carbslider+4),137);
     myGLCD.setColor(16, 167, 103);
     myGLCD.fillRect(31, 129, (carbslider-1), 137);
     myGLCD.setColor(255, 255, 255);
     myGLCD.fillRect((carbslider+5), 129, 309, 137);
     strcarb= totalcarb;
     myGLCD.print(strcarb, 200,105);
  //Slider für Fett
     myGLCD.setColor(255, 255, 255);
     myGLCD.fillRect(fatslider,169,(fatslider+4),177);
     myGLCD.setColor(16, 167, 103);
     myGLCD.fillRect(31, 169, (fatslider-1), 177);
     myGLCD.setColor(255, 255, 255);
     myGLCD.fillRect((fatslider+5), 169, 309, 177);
     strfat= totalfat;
     myGLCD.print(strfat, 200,150);
```

```
      if (myTouch.dataAvailable()) {
         myTouch.read();
         x=myTouch.getX();
         y=myTouch.getY();
//Wenn "Fertig Button" gedrückt
         if ((x>=200) && (x<=310) && (y>=190) && (y<=230)) {
           drawFrame(200, 190, 310, 230);
           currentPage = '0';
           myGLCD.clrScr();
           drawHomeScreen();


         }
       }
}

}
```

**Bugs:** Leider läuft, obwohl der obige Code meiner Meinung nach Sinn ergibt, nicht alles ganz wie es sollte. Grundlegend werden zwar alle an die Sketch gestellten Anforderungen erfüllt und sie ist voll funktionstüchtig, es gibt jedoch einige grafische Schönheitsfehler. Zum einen bleiben, nach Berühren des «Zurück Buttons» der Auswahlseite 1 und Aufrufen des Homescreens, die Auswahlslider über dem Homescreen angezeigt, obwohl der Bildschirm mit `myGLCD.clrScr()` zuvor eigentlich komplett geleert werden hätte müssen.

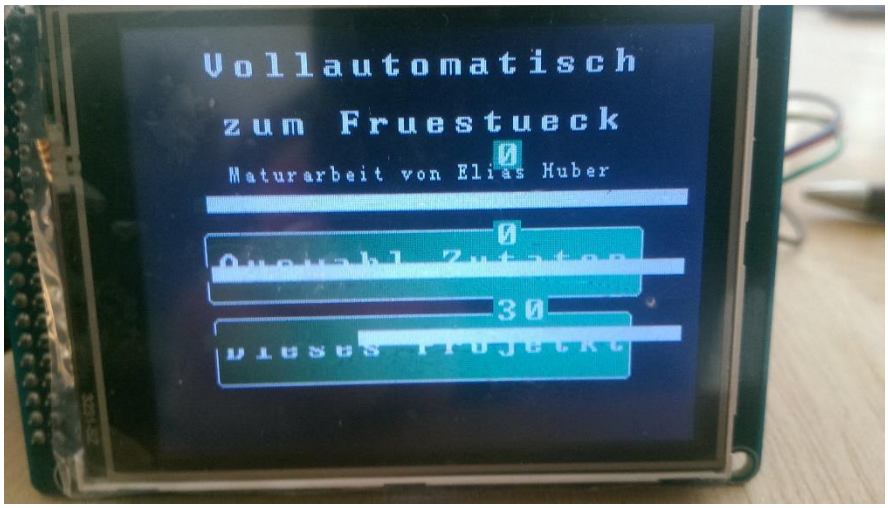



Abb. 14: Verbuggter Homescreen nach Betätigung des «Zurück Buttons»

Um dies zu beheben wurden folgende Versuche unternommen:

- Die Position von `myGLCD.clrScr()` wurde variiert, so habe ich zum Beispiel versucht sie erst mit der «drawHomescreen» Funktion aufzurufen.
- Mit Hilfe von (dem im nächsten Unterkapitel vorgestellten) **Serial**.print wurde getestet, ob die Balken zu einem späteren Zeitpunkt nochmals gezeichnet werden -> sie werden es nicht.
- Auch habe ich versucht, mit dem Zeichnen eines grossen schwarzen Rechtecks über die ganze Bildschirmfläche in der «drawHomescreen» Funktion die unerwünschten Slider optisch nicht mehr sichtbar zu machen. Dies hat ebenfalls nicht funktioniert, da die verbuggten Slider alles Neugezeichnete überschreiben.

Auch gibt es bei der Wertanzeige der Auswahlslider auf der zweiten Auswahlseite ein paar Schönheitsfehler, sofern auf der ersten Auswahlseite Mengen ausgewählt wurden. So stehen ausgewählte Werte teilweise auch auf der zweiten Auswahlseite (werden jedoch richtig gesendet).

Bezüglich der Ursache dieser Fehler tappe ich noch im Dunkeln, da ich jedoch im Code keine Fehler finden kann, nehme ich an, dass es sich um Bugs in der `UTFT` oder `URTouch` Library handelt, da diese ja auch nur ein Einmannprojekt sind und Fehler daher, möglicherweise auch aufgrund von Kompabilitätsproblemen mit meiner Hardware, vorkommen. Vielleicht noch wahrscheinlicher ist es möglicherweise aber, dass ich einfach irgendeinen dummen Fehler gemacht habe. (Sollte dies der Fall sein, würde ich mich freuen, wenn sie mich unter elixaver@gmail.com kontaktieren und darauf aufmerksam machen könnten.)

## 5.2 EMPFANGSSKETCH

Damit wären wir nun bei der Empfangssketch angelangt, welche lediglich die vom Master gesendeten Daten entgegennehmen und anschliessend die Motoren entsprechend ansteuern soll.

Der erste Teil enthält wieder lediglich ein paar Variablendefinitionen sowie Voreinstellungen.

```
#include <AFMotor.h>   //spezifische Funktionen für das Motorshield
#include <Wire.h>
float a,b,c,d,e,f;     //Zahle mit Nachkommastellen
int x=0;
byte werte[6];
AF_Stepper motor(48, 1); //Benennung der 2 Motoren und Zuordnung zu den
AF_Stepper motor2(200, 2);//Ports des Motorshields
```

Das Setup ist da schon interessanter, wieder wird die $I^2C$ Datenübertragung gestartet, nur diesmal als Slave. Ausserdem wird mit «onReceive» ein sogenannter Handler, in diesem Fall eine, von uns anschliessend deklarierte Funktion aufgerufen, wenn immer Daten empfangen werden. Zusätzlich wir auch die serielle Datenübertragung mit dem Computer gestartet. Diese hat letztendlich keine Bedeutung mehr für das Funktionieren der Sketch, war im Entwicklungsprozess jedoch eine grosse Hilfe, weshalb ich den dazugehörigen Code stehengelassen habe. Der Parameter 9600 ist die Übertragungsgeschwindigkeit (auch Baud-Rate), für welche man bei einem normalen USB Anschluss normalerweise auf 9600 bps (*Bits* pro Sekunde) festgelegt wird.

```
void setup() {
Wire.begin(9); //Beginnt I²C Datenübertragung als Slave mit Adresse 9
//Ruft sobald Daten empfangen werden die Funktion "receiveEvent" auf
Wire.onReceive(receiveEvent);
//Beginnt serielle Übertragung zum Computer mit 9600 bps
Serial.begin(9600);
motor2.setSpeed(5); //Der Motor motor2 wird mit einer Geschwindigkeit von
//5 drehen
}
```

Die bei Empfang aufgerufene Funktion hat nun die Aufgabe, die gesendeten Daten abzuspeichern. Es ist zu sehen, dass dabei ein *Integer* entgegengenommen wird. Dieser enthält die Anzahl der gesendeten Bytes und ist in diesem Fall von keiner weiteren Bedeutung. Der Inhalt dieser Variable beträgt hier sowieso immer nur Eins, da, wie in der Hauptsketch zu sehen war, nur je ein Byte auf einmal gesendet wird.
Im Folgenden werden die erhaltenen Daten abgespeichert, dies mit Hilfe einer Variable x, welche den aktuellen Arrayplatz indiziert. Zur Erinnerung, in der Hauptsketch werden der Reihe nach die Werte der Arrayplätze 0, dann 1, 2 usw. gesendet. Da x am Anfang Null ist, wird der erste gesendete Wert wieder im ersten Arrayplatz abgespeichert, wonach x um Eins vergrössert wird, sodass der zweite Wert im zweiten Platz gespeichert wird. Nachdem alle sechs Werte empfangen wurden, wird x wieder auf Null gesetzt, womit das Ganze erneut beginnen kann. Nach jeder Übertragung wird der übertragene Wert auf den Bildschirm (auf eine neue Linie -> print**ln**) gedruckt, sofern ein Computer angeschlossen ist. Dadurch konnte einfach überprüft werden, ob die Daten auch richtig übertragen und abgespeichert wurden. Verwirrend könnte hier die If-Abfrage sein, weshalb ich auf sie genauer eingehen möchte. Die Funktion sizeof() misst die Anzahl enthaltener Bytes ihres Parameters. Im ersten Fall

(`sizeof(werte)`), sind dies sechs Bytes (jeder Arrayplatz einen). Im zweiten (`sizeof(werte[0])`) einer. Unsere Rechnung lautet also 6/1 und gibt Sechs. Klingt sinnvoll, da wir ja sechs Arrayplätze nacheinander ausgeben möchten. Nun fragt man sich vielleicht, was, neben der schöneren Optik, der Grund ist es so kompliziert zu schreiben, statt von Anfang an eine Sechs. Der Vorteil dieser Notation besteht darin, dass ich nichts an der If-Bedingung ändern muss, sollte ich mich entscheiden, die Anzahl der auswählbaren Zutaten zu erhöhen (oder Zahlen über 256 zu verwenden, welche nicht mehr in einem Byte Platz haben).

```
//Unsere Funktion, welche aufgerufen wird sobald Daten empfangen werden,
//übernimmt einen Integer von onReceive
void receiveEvent(int bytes){

  werte[x] = Wire.read();
//Ist x im Bereich des Arrays?
  if (x < (sizeof(werte) / sizeof(werte[0]))){
    //schreibt den entsprechenden Platz des Werte-Arrays auf den Bildschirm
    Serial.println(werte[x]);
    x++;
  }
  else {
    x=0;
    }
 }
```

Im Loop werden diese Daten nun verwendet, um die Motoren entsprechend anzusteuern. Es soll nur etwas passieren, wenn der Werte *Array* auch Werte enthält, daher die erste If-Bedingung. Anschliessend wird überprüft, ob von den jeweiligen Zutaten etwas ausgewählt wurde. Ist dies der Fall dreht der kleine Stepper («motor») die Öffnung auf und wartet bei offener Öffnung eine von der ausgewählten Menge abhängige Zeit, welche aufgrund von Erfahrungswerten und der beim Öffnungsvorgang austretenden Menge berechnet wird (das Minus ist für die Menge, welche beim Öffnungsvorgang austritt). Nach Ablauf dieser Zeit wird die Öffnung wieder verschlossen und der grosse Stepper («motor2») dreht den nächsten Behälter in Position, worauf der Vorgang wiederholt wird.
Der kleine Motor benötigt übrigens so grosse Schrittzahlen und «Double-Steps» (Doppelschritte), da er ein eigenes Getriebe eingebaut hat und so auf eine Umdrehung weit über tausend Schritte benötigt. Der grosse Motor (motor2) braucht für eine Umdrehung 200 «Single-Steps». 200/6 sind entsprechend die Schritte zwischen zwei Zutatenbehältern. Die zwei weiteren Faktoren (40/14 und 51/26) entsprechen dem Zahnradverhältnis eines hinzugefügten Getriebes, welches das maximal vom Motor benötigte *Drehmoment* reduziert. In manchen Fällen ist noch ein zusätzlicher Summand/Subtrahend zu sehen. Dieser resultiert aus leichten Ungleichheiten bei der Öffnungsplatzierung sowie Rundungsungenauigkeiten bei den vorherigen Brüchen (der Motor kann nur ganze Schritte machen).

```
void loop() {
 if((werte[0]+werte[1]+werte[2]+werte[3]+werte[4]+werte[5])>0) {
//Wenn von der ersten Zutat etwas ausgewählt
if (werte[0]>0){
 motor.setSpeed(350);
 //drehe motor um 1000 "Double-Steps" vorwärts
  motor.step(1000, FORWARD, DOUBLE);
 //Berechnung der Öffnungszeit
  a=(werte[0]-13)*1000/21;
```

```
  if (a<0) {
    a=0;
    }
  delay(a); //wartet a ms
//drehe motor um 1000 "Double-Steps" rückwärts
  motor.step(1000, BACKWARD, DOUBLE);
  }
//drehe motor2 um 200/6*40/14*51/26+10 "Single-Steps" rückwärts
  motor2.step(200/6*40/14*51/26+10, BACKWARD, SINGLE);

//Wenn von der zweiten Zutat etwas ausgewählt
 if (werte[1]>0){
  motor.setSpeed(350);
  motor.step(1100, FORWARD, DOUBLE);
  b=(werte[1]-40)*1000/26.4;
  if (b<0) {
    b=0;
    }
  delay(b);
  motor.step(1100, BACKWARD, DOUBLE);
  }

  motor2.step(200/6*40/14*51/26, BACKWARD, SINGLE);

//Wenn von der dritten Zutat etwas ausgewählt
 if (werte[2]>0){
  motor.setSpeed(300);
  motor.step(1000, FORWARD, DOUBLE);
  c=(werte[2]-1)*1000/3.6;
  if (c<0) {
    c=0;
    }
  delay(c);
  motor.step(1000, BACKWARD, DOUBLE);
  }

  motor2.step(200/6*40/14*51/26+5, BACKWARD, SINGLE);

//Wenn von der vierten Zutat etwas ausgewählt
 if (werte[3]>0){
  motor.setSpeed(300);
  motor.step(800, FORWARD, DOUBLE);
  d=(werte[3]-1)*1000/2;
  if (d<0) {
    d=0;
    }
  delay(d);
  motor.step(800, BACKWARD, DOUBLE);
  }

motor2.step(200/6*40/14*51/26+5, BACKWARD, SINGLE);

//Wenn von der fünften Zutat etwas ausgewählt
 if (werte[4]>0){
  motor.setSpeed(300);
  motor.step(950, FORWARD, DOUBLE);
  e=(werte[4]-1)*1000/4.75;
  if (e<0) {
    e=0;
    }
  delay(e);
  motor.step(950, BACKWARD, DOUBLE);
```

```
  }

motor2.step(200/6*40/14*51/26, BACKWARD, SINGLE);

//Wenn von der sechsten Zutat etwas ausgewählt
 if (werte[5]>0){
  motor.setSpeed(300);
  motor.step(1050, FORWARD, DOUBLE);
  f=(werte[5]-5)*1000/11.54;
  if (f<0) {
    f=0;
    }
  delay(f);
  motor.step(1050, BACKWARD, DOUBLE);
 }

motor2.step(200/6*40/14*51/26-5, BACKWARD, SINGLE);

 werte[0] = 0; //Setzt alle Plätze im Werte Array auf Null
 werte[1] = 0;
 werte[2] = 0;
 werte[3] = 0;
 werte[4] = 0;
 werte[5] = 0;
 }
 }
```

# 6. PRAXISTEIL

In diesem Kapitel wird die praktische Realisierung der zuvor behandelten Theorie gezeigt, wobei ich das Kapitel nach den verschiedenen Komponenten unterteilt habe.

## 6.1 HERSTELLEN DER INNENBEHÄLTER

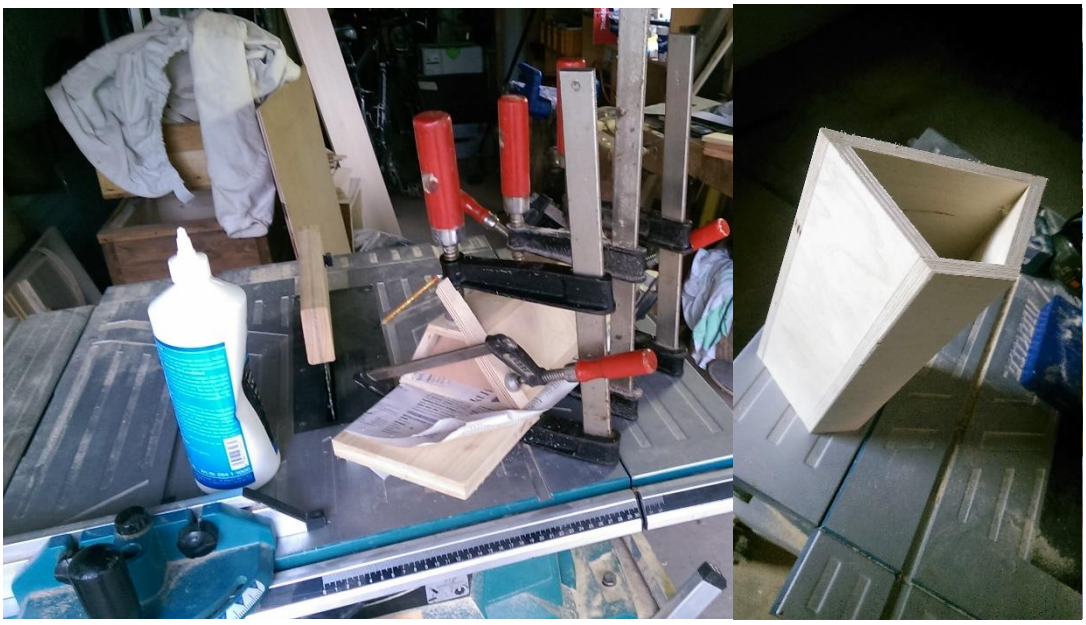


Abb. 15: Verleimen eines Inhaltsbehälters, links neben ihm das Blatt der Kreissäge, rechts daneben ein fertiger Behälter

Zur Erinnerung, die Behälter sollen an einen sechseckigen Drehturm angehängt werden und haben entsprechend einen trapezförmigen Grundriss. Sie haben in der Endfassung eine Wandstärke von einem Zentimeter und die einzelnen Teile sind verleimt. Die Wandstärke ist mit Sicherheit etwas sehr grosszügig bemessen, aber dort ging es nach dem Motto lieber zu viel als zu wenig, ausserdem sind die Leimflächen so grösser. Nach dem Erstellen einer Skizze und Festlegen auf die Masse, war der erste Schritt das Abschleifen der späteren Innenseiten der Behälter. Daraufhin folgte das Zuschneiden der Holzteile mithilfe einer Kreissäge mit verstellbarem Winkel.

Das nachfolgende Leimen stellte sich aufgrund der unhandlichen Trapezform der Behälter als komplizierter heraus als gedacht, war mithilfe von Innenformen und vielen Schraubzwingen aber machbar.

Anschliessend wurden auch die Aussenseiten geschliffen und gehobelt. Letztlich wurden die Formen aus hygienischen Gründen noch von innen mit einer Leinöltinktur bestrichen, woraufhin die Bearbeitung der Rohform abgeschlossen war. Das Anbringen der Öffnungen sowie die Fertigstellung des ganzen Behälters werden im Unterkapitel 6.5 erläutert.

## 6.2 HERSTELLEN DER ÖFFNUNGEN

Die Öffnungen sind letztendlich anders geworden als anfänglich geplant. Im Kapitel 4.3.3 war zu lesen, dass die Öffnungsplatten an einem Drehpunkt befestigt sind und bei Druck aufschwingen. Wie im Bild zu sehen, sieht die finale Version so aus, dass sich die ganze Öffnungsplatte auf Schienen gerade nach hinten bewegt. Dies hat den Vorteil, dass keine schräge Kraft auf den Stössel wirkt und ausserdem die Öffnungsgrösse im Verhältnis zur Stosslänge grösser ist. Als Material habe ich relativ dünnes Blech genommen, welches sich recht gut mit einer entsprechenden Blechschere schneiden liess. Die anschliessende Faltung und Entgratung mit der Schleifmaschine stellte sich mal wieder als komplizierter heraus als gedacht, vor allem die Bewerkstelligung des reibungsfreien Laufs der Öffnungsplatte durch die Schienen bereitete einige Probleme. Die Schienen habe ich anschliessend auf der Bodenplatte aufgenagelt, wo auch noch die Befestigungen für die Gummis sowie Stopper (der Öffnungsplatte) angebracht wurden. Ich habe mich für Gummis statt Federn entschieden, da diese nicht so schnell an Spannkraft verlieren und ausserdem einen gleichmässigeren Zug gewährleisten. Die Öffnung selbst wurde gebohrt, bzw. für gröbere Zutaten mit der Stichsäge gesägt. Auf der Innenseite wurden die Kanten noch mit einer Oberfräse abgeschrägt, um einen besseren Zutatenfluss zu gewähren.

Abb. 16: Öffnungsmechanismus, links daneben Schleifmaschine

## 6.3 HERSTELLEN DES DREHTURMS

Der Drehturm ist im Prinzip ein Prisma mit sechseckigem Grundriss, das auf einer Drehscheibe steht.


Abb. 17: Turm beim Leimen, rechts fertiger Turm

Für die Herstellung des Turms habe ich wie zuvor bei den Zutatenbehältern, zuerst die Seitenflächen ausgesägt und anschliessend verleimt. Dabei musste ich feststellen, dass das Zusammenleimen eines sechseckigen Prismas noch garstiger zu bewerkstelligen ist, als das eines mit Trapezgrundfläche. Dank der bei den Zutatenbehältern zuvor gesammelten Erfahrungen, einer Menge Schraubzwingen und sorgfältiger Nachbearbeitung entstand am Ende aber doch noch ein stabiler Turm.

Für die Drehplatte ist ein Spannlager verwendet, an welchem man einerseits an einem gewölbten Aussenring (1) und mit Schrauben (2) an einem Innenring Platten befestigen kann, die anschliessend gegeneinander drehbar sind.


Abb. 18: Spannlager

Die mit dem Aussenring verbundene Platte habe ich im Folgenden fest an einer Grundplatte fixiert, während der Turm sich auf einer mit dem Innenring verbunden Platte drehbar befindet (1 unten, 2 oben). Durch den Innenring dreht dann eine, an dem grossen Stepper über ein Getriebe verbundene Achse ((1), Abb. Oben) den Turm mitsamt angehängten Zutatenbehältern. Für die Achse habe ich einen Metallstab genommen, auf den ein Gewinde gedreht wurde. Er ist durch ein Loch in der Bodenplatte des Turms gesteckt und mit zwei Muttern gegenseitig festgedreht.

## 6.4 HERSTELLEN DES STÖSSELS

In der nebenstehenden Abbildung ist ein Bild des Stossmechanismus zu sehen. Dieser wurde nicht, wie anfangs geplant, als Schwungrad mit angebrachter Stange umgesetzt (siehe Kap 4.3). Dies aus dem Grund, dass bei einem Schwungrad die Ausstossrichtung des Stössels (bezüglich der Höhe) variiert. Das liesse sich bei fester Stosslänge zwar durch eine Vergrösserung des Schwungrades oder Verlängerung der angebrachten Stange minimieren. Es bestände im ersten Fall jedoch der Nachteil, dass das auf den kleinen Schrittmotor wirkende *Drehmoment* aufgrund des längeren Hebelarms vergrössern würde. Dies würde zu einer grösseren maximalen Stromstärke führen, deren mögliche Konsequenzen bereits in Kapitel 4.4 erläutert wurden. Die Variante mit einer Verlängerung der Stossstange hätte den Nachteil einer weiteren Verlängerung der Apparatur. Da sich auch entsprechende Testversuche als schwierig erwiesen, wurde nach einer anderen Möglichkeit gesucht, womit wir wieder beim nebenstehenden Bild und dessen Erläuterung angelangt sind.

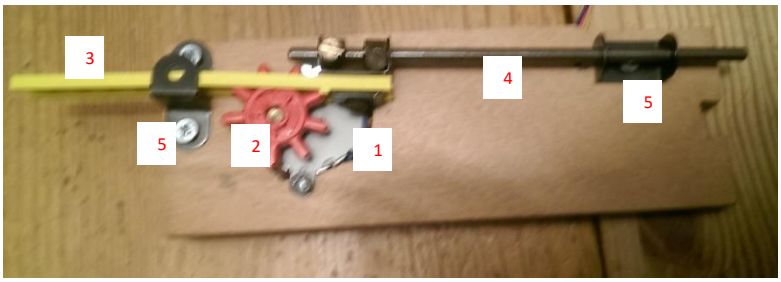



Abb. 19: Stossmechanismus

Auf einer Holzplatte ist der kleine Stepper(1) befestigt, welcher ein Zahnrad(2) dreht. Auf diesem Zahnrad ist eine Art Leiter(3) (Danke LEGO!) befestigt, deren Sprossenabstand mit den Abständen zwischen den Zähnen des Zahnrades übereinstimmt. An dieser Leiter wiederum ist ein Metallstab(4) befestigt, welcher die Öffnung aufstösst. Stab und Leiter werden durch Befestigungen(5) so in der Bahn gehalten, dass sie bei Drehung des Zahnrades durch den Motor gerade nach vorne geschoben werden.

## 6.5 FERTIGSTELLEN DES PROJEKTS

In diesem Teil erkläre ich, wie die Einzelteile letztlich zusammengefügt wurden. Ausserdem ist hier der Grundsockel der Maschine erläutert.

Abb. 20: Verleimen der Grundplatte mit dem Sockel

Der Grundsockel hat ebenfalls eine sechseckige Form, wobei ich an einer Seite ein Rechteck stehengelassen habe, auf welchem später die Müslischale abgestellt und der Touchscreen platziert wird. Auf die Platte ist der Sockel für die «Drehplatten» aufgeleimt, welcher die Form eines halben sechseckigen Prismas hat.

Auf diesem Sockel wurden dann das Spannlager, mit den daran befestigten, gegeneinander drehbaren, zwei Platten, angebracht. Auf die obere dieser Platten wurde der nun drehbare Turm installiert. Auf der unteren, fest mit dem Sockel verbundenen Platte hingegen, ist an der Öffnungsstelle der Stossmechanismus befestigt. An dieser Stelle ist ausserdem ein Loch in die Platte gesägt, durch welches die Zutaten hindurch fallen können. Damit stand das grobe Gerüst bereits, nun fehlten nur noch die Behälter, die Anbindung des grossen Steppers sowie die Hardware.


Abb. 21: Fertiger Zutatenbehälter

Für die Fertigstellung der Zutatenbehälter mussten einerseits die Öffnungsmechanismen angebracht werden, andererseits die Befestigungen, um sie an den Drehturm anzubringen. Bezüglich ersterem habe ich die Öffnungen angeschraubt und allfällige Kanten abgeschliffen. Für das Anbringen an den Drehturm habe ich aus Blech ein «Haken»(1) gebogen und dann mit zwei Schrauben befestigt.

Für die Verbindung zwischen dem grossen Stepper und der am Drehturm befestigten Stange waren erneut Planänderungen nötig. Ursprünglich war vorgesehen, den Stepper direkt mit dieser Stange fest zu verbinden. Es stellte sich dann jedoch heraus, dass, sobald die am Drehturm angebrachten Zutatenbehälter gefüllt werden, sich der Drehwiderstand des Spannlagers erhöhte, da die Gummidichtungen desselben auf der Holzbefestigung schliffen. Nun reichte in diesem Fall das *Drehmoment* des Steppers nicht mehr aus, den ganzen, vollbeladenen Turm zu drehen. Ein stärkerer Motor kam jedoch nicht in Frage, da sich der vorhandene bezüglich Stromstärke und Spannung bereits an der oberen Grenze des für das Motorshield Möglichen befand. Daher fiel die Entscheidung mittels Zahnrädern ein Getriebe

auf die Bodenplatte zu improvisieren, welches das auf den Motor wirkende *Drehmoment* reduziert.

Wie im nebenstehenden Bild zu sehen, wurden dafür vier Zahnräder verwendet. Ein kleines, sich auf der Motorstange befindendes Zahnrad versetzt ein verhältnismässig grösseres in Drehung. Dieses befindet sich an einer Achse, an welcher ein weiteres, wiederum kleines (rotes) Zahnrad befestigt ist welches erneut ein grösseres Zahnrad dreht, das direkt an der Achse des Drehturms befestigt ist. Die Drehzahl **i** ($\frac{Zähne\ Antrieb}{Zähne\ Abtrieb}$) beträgt an den messingfarbigen Zahnrädern ca. 0.51 und bei den oberen Zahnrädern 0.35. Die Gesamtübersetzung $\mathbf{i_{ges}}$ ist damit $0.51 * 0.35 \approx 0.18$. Entsprechend ist das vom Motor benötigte *Drehmoment*, unter Vernachlässigung von Reibung im Getriebe, $M_{Motor} \approx 0.18 * M_{Drehturm}$ , kurz: Die nun vom Motor benötigte «Kraft» (*Drehmoment*), um **sich** gleichschnell wie zuvor zu drehen, beträgt nur noch ca. $\frac{1}{5}$ der ohne Getriebe notwendigen «Kraft».[26] Allerdings muss, im Verhältnis zum Fall ohne Getriebe, auch die Drehgeschwindigkeit des Steppers verfünffacht werden, um eine gleichschnelle Drehgeschwindigkeit des Drehturms zu ermöglichen. Überlegungen zum Beschleunigungsvorgang sowie zu Auswirkungen von höherer Drehgeschwindigkeit des Motors (z.B. auf benötigte Spannung) seien hiermit, genauso wie solche zu Reibung im Getriebe, angemerkt, werden jedoch nicht weiter vertieft.

Abb. 22: Getriebe für den grossen Stepper

Der Stepper wird von an der Grundplatte angeschraubten Metallverstrebungen fest in Position gehalten.

[26] Quelle : https://de.wikipedia.org/wiki/%C3%9Cbersetzung_(Technik)

Nach dem Einbau des Getriebes ging die Verkabelung und der Einbau der Hardware ausnahmsweise einmal problemlos vonstatten. Der Touchscreen ist in einen mit der Stichsäge gesägten Rahmen geschraubt, welcher dann über ein Scharnier(1) mit der Bodenplatte befestigt wurde. Dies hat den Vorteil, dass sich der Touchscreen nach vorne umklappen lässt, sollten weitere Arbeiten vorgenommen oder ein sehr grosse Müslischale verwendet werden.

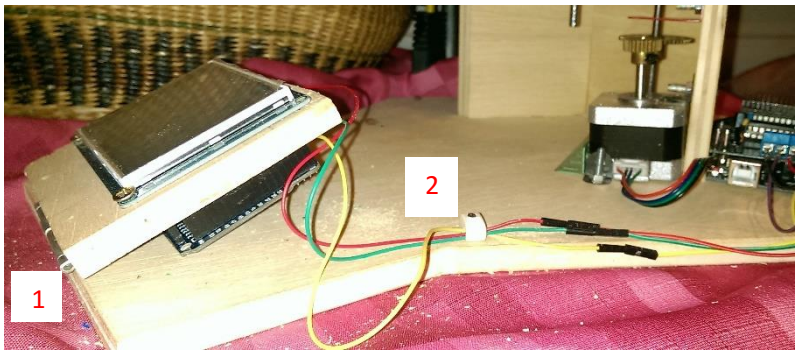


Abb. 23: Platzierung der Hardware und Verkabelung der Komponenten

Das Arduino Uno mit dem Motorshield habe ich direkt neben dem grossen Stepper angeschraubt. Für die Verkabelung sind weisse Kabelhalter(2) verwendet, dabei habe ich absichtlich keine *Jumper* fest verlötet, um leichtes Auseinandernehmen zu ermöglichen. Lediglich für das Kürzen von Kabeln kam der Lötkolben zum Einsatz.

In der nebenstehenden Abbildung ist die fertige Müslimaschine in ihrer vollen Pracht zu sehen. Lediglich ein optisches Detail wurde hier noch nicht erläutert; der Titel der Arbeit sowie Nummerierungen für die richtige Zutatenreihenfolge. Für die Herstellung der Beschriftungen wurde ein 3D-Drucker verwendet. Ursprünglich für die Herstellung der Drehräder (Kap. 4.3.1) geplant, welche leider nicht verwirklicht wurden, hat diese faszinierende Technologie schliesslich doch noch Eingang in meine Arbeit gefunden, wenn auch nur für die optische Vervollständigung.

Wenn sie die Müslimaschine einmal in voller Aktion sehen wollen, finden sie unter folgendem Youtubelink ein Video:

https://youtu.be/zGifQDpxGSU

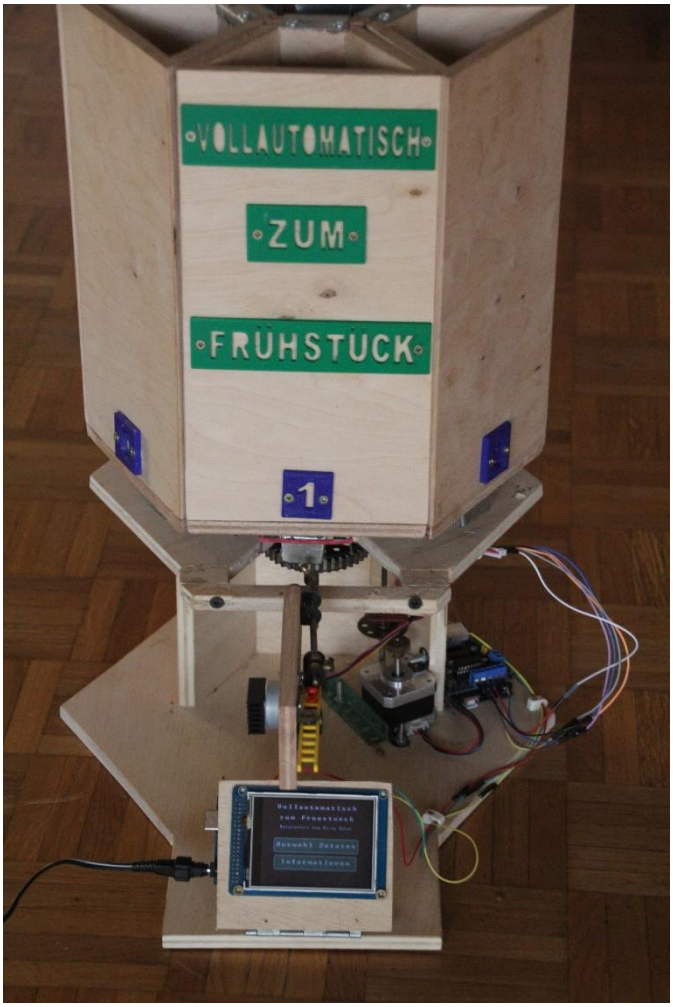


Abb. 24: Die fertige Müslimaschine

**7. Fazit**

An dieser Stelle möchte ich einen Blick zurück auf die intensive Zeit während der Entstehung dieser Arbeit und die gewonnenen Erfahrungen werfen.

Angefangen mit einer ungenauen Idee, eine automatische «Müslimischmaschine» zu bauen und einem Grundlagenbuch über Arduino[27], ist nun nach vielen Stunden ein funktionierendes Produkt entstanden. Dabei lief sicher nicht alles so, wie es in der Theorie geplant war und viele Entscheidungen und Pläne erwiesen sich in der Praxis als schwerer umsetzbar als gedacht, wurden geändert und wieder verworfen. Würde ich mich mit den dabei gewonnenen Erfahrungen erneut an den Bau einer Müslimaschine machen, so würde ich wahrscheinlich manches anders angehen. Doch die selbständige Einarbeitung in die mir neue Welt von Arduino sowie die intensive handwerkliche Betätigung, waren für mich, sobald die Sache einmal ins Rollen kam, extrem interessant und bereichernd. Dabei gab es durchaus Hürden und Tiefpunkte. Momente in denen unvorhergesehene Probleme die Arbeit vieler Stunden vergebens machten oder solche, in denen es schien, dass obschon alles richtiggemacht wurde, Dinge aus mysteriösen Gründen nicht funktionierten und in denen ernsthafte Zweifel am Erfolg der Arbeit aufkamen. Rückblickend waren dies zwar die schwierigsten Momente, aber keinesfalls vergeudete Zeit, da sie ebenso lehrreich und stärkend waren wie die Erfolgserlebnisse.

Das Endprodukt ist sicher nicht perfekt und im nächsten Kapitel werden zahlreiche Verbesserungsmöglichkeiten angesprochen, trotzdem denke ich das die gestellten Zielsetzungen, eine funktionierende Müslimaschine aus bescheidenen Mitteln zu bauen, voll erfüllt wurden. Bezüglich der Zeitersparnis gibt es allerdings noch Verbesserungspotential, dafür läuft der Zubereitungsvorgang nach Eingabe aber vollautomatisch ab.

Ich bin überzeugt, dass ich auch in Zukunft mit positiven Erinnerungen auf die Zeit und die gesammelten Erfahrungen während dem Bau meiner Müslimaschine zurückblicken werde, mit der ich sicher nicht das Rad neu erfunden habe, auf welche ich aber in jeder Hinsicht stolz bin!

[27] McComb, Gordon : *Arduino Robot Bonanza*. McGraw-Hill

# 8. VERBESSERUNGS- UND ERWEITERUNGSMÖGLICHKEITEN

Im Folgenden werde ich einige denkbare Verbesserungen und Erweiterungen der Müslimaschine aufführen.

## 8.1 MÖGLICHE VERBESSERUNGEN UND ÄNDERUNGEN AM BESTEHENDEN GERÄT

Die hier aufgeführten Änderungsvorschläge wären in absehbarer Zeit an der aktuellen Maschine umsetzbar.

Der grösste Schwachpunkt des jetzigen Gerätes ist der Öffnungsmechanismus. Ein Aufschieben der Öffnung und anschliessendes «Herausströmen» der Zutaten, schien mir, auch angesichts der Tatsache, dass ich dieses Konzept bereits von einer Getreidemühle kannte, anfangs der beste der in Kapitel 4.3 vorgestellten Öffnungsmechanismen. Im Nachhinein war diese Entscheidung wohl nicht die beste. Mit kleinen Zutaten, wie den Körnern einer Getreidemühle funktioniert der Mechanismus zwar einwandfrei, mit grösseren, weniger dichten Zutaten zeigen sich jedoch einige Probleme. Im jetzigen Automaten können zum Beispiel keine Haferflocken verwendet werden, wobei sich diese eigentlich hervorragend zum Mischen eignen würden. So kommt es bei diesen relativ schnell zu Verklemmungen, was einen ungleichmässigen Zutatenfluss und entsprechend ungenaue Kalorienangaben zur Folge hat. Auch sehr zuckrige Zutaten haben sich als ungeeignet herausgestellt, da diese bei zu langem Stehen, besonders bei warmen Zimmertemperaturen, verkleben. Ein weiterer Nachteil ist, dass sich der Stössel direkt im Ausflussstrahl der Zutaten befindet, wodurch dieser gestreut wird und eine grosse Schale für das Auffangen benötigt wird.

Eine bessere Alternative wäre vermutlich die «Öffnung mit Drehrad» gewesen (Kap. 4.3.1), welche eine genauere Abgabe, auch von grösseren Zutaten ermöglichen würde. Der Umbau dafür wäre zwar aufwändig, aber ohne komplette Neukonstruktion möglich, da die aktuellen Öffnungen an den Zutatenbehältern einfach abgeschraubt und durch neue ersetzt werden könnten. Auch der Stössel müsste natürlich entfernt und durch einen neuen Mechanismus ersetzt werden, was auch die Anschaffung eines stärkeren Motors erfordern würde.

Ein weiterer denkbarer Zusatz wäre die Möglichkeit einer Auswahl von Beispielkombinationen, welche sich der Benutzer mischen lassen kann. Dies wäre sehr einfach zu implementieren, die Tauglichkeit bei nur sechs Zutaten ist hingegen fragwürdig.

## 8.2 NOTWENDIGE ÄNDERUNGEN FÜR KOMMERZIELLEN VERTRIEB

Sollte die Müslimaschine jemals an Kunden gebracht werden, müssten natürlich einige Anpassungen getroffen werden. Dabei ist zu unterscheiden, ob die Maschine für den Heimgebrauch oder für den Einsatz in Restaurants und Hotels am Frühstücksbüffet gedacht wäre. Ersteres macht meiner Meinung nach wenig Sinn, aufgrund des benötigten Platzes und der geringen Nutzung im Verhältnis zum recht hohen Anschaffungspreis. Ausserdem haben wohl die wenigsten Leute das Bedürfnis, jeden Morgen ein individuell zusammengemischtes Müsli zu geniessen. Als Bestandteil des Morgenbüffets, in einem Hotel oder Restaurant könnte ich mir kommerzielle Nutzung aber definitiv vorstellen. Im Folgenden habe ich einige Änderungen erklärt, welche nötig oder zumindest sinnvoll wären.

Es würde für den kommerziellen Gebrauch wenig Sinn machen, den gesamten Automaten aus Holz zu fertigen. Einzelne Bestandteile, wie zum Beispiel die Zutatenbehälter könnten um einiges einfacher aus Kunststoff in grossen Mengen produziert werden. Dies würde auch einen geringeren Platzaufwand benötigen, da die Wandstärke reduziert werden könnte.

Ein weiterer Punkt wäre die Hygiene; um den Automaten öffentlich einzusetzen, müssten natürlich die Hygienevorschriften des eidgenössischen Departements des Innern[28] beachtet und das Gerät bewilligt werden.

Auch die Zutatenmenge ist für den Einsatz in Büffets im aktuellen Modell noch etwas mager. In diesem Zusammenhang sollte sicher überdacht werden, ob es bezüglich der Idee eines Betriebes mit nur zwei Motoren, zumindest aus rationaler Sicht nicht bessere Möglichkeiten gäbe. Natürlich ist es aus Demonstrationszwecken ziemlich cool, wenn sich die ganze Maschine mit dreht. Auch in Betracht der Kosten ist es effizient, da zusätzliche Motoren auch für die Ansteuerung weitere Hardware benötigen würden. Andererseits ist der gesamte Drehvorgang im Verhältnis zum Rest der Maschine sehr fehleranfällig und beschränkt in der Praxis die maximale Zutatenmenge. Es wäre stattdessen zum Beispiel ein separates Öffnen jedes Zutatenbehälters und eine spätere Zusammenführung der Zutaten denkbar, was auch ein Vermischen der Zutaten ermöglichen würde.

Ebenfalls ist es nicht sehr kosteneffizient, für die Ansteuerung des Motors und des Touchscreens je ein separates Arduinoboard mit eigenem Stromanschluss und je vielen ungenutzten Extrafunktionen zu nehmen. So könnte sicher die Funktion des Arduino Uno mit dem Motorshield deutlich günstiger ersetzt werden. Das Problem bestand ja darin, dass auf dem Arduino Mega zu wenig Pins für die Ansteuerung von Touchscreen und Motoren

[28] https://www.admin.ch/opc/de/classified-compilation/20050160/201401010000/817.024.1.pdf

vorhanden sind. Mit der in Kapitel 4.2 vorgestellten $I^2C$ Methode, wäre es jedoch möglich, Motortreiber und Touchscreen vom Arduino Mega aus anzusteuern. Dabei sorgen erstere mithilfe der empfangen Informationen lediglich für die richtige Stromversorgung der Motoren und wären um einiges weniger material- und kostenintensiv als ein ganzes Arduino Uno mit einem vielseitig einsetzbaren Motorshield. Dies würde auch das Hinzufügen beliebig vieler weiterer Motoren deutlich vereinfachen.

# 9. DANKSAGUNG

In diesem Kapitel möchte ich mich bei den Menschen bedanken, welche mich während der Entstehung der Arbeit in unterschiedlichster Form unterstützt haben.

An erster Stelle möchte ich hier meiner Familie danken. Die Arbeit stellte mich über einige Zeit unter ziemlichen Stress und die praktischen Tests und Basteleien blockierten oft die halbe Wohnung, was sicher nicht immer einfach war. Besonders erwähnen möchte ich hier meinen Vater, ohne sein gewaltiges Vorwissen als ehemaliger Schreiner wäre mir der praktische Teil in dieser Form unmöglich gewesen. Er stand mir während der gesamten Bauphase stets mit Rat und Tat zur Seite. Seine kreativen Lösungsvorschläge sorgten oft für eine viel einfachere und elegantere Lösung von praktischen Problemen. Auch die Nutzung der umfangreichen Werkstatt, trotz zerbrochenen Bohrern und einer gigantischen Unordnung, war zentral für das Gelingen des Projekts. Auch meiner Mutter möchte ich danken, welche meine Arbeit stets geduldig Korrektur las und mich auf ungeschickte Formulierungen hinwies.

Bedanken möchte ich mich auch bei Walter Vogelsanger und Günter Schmidtner für die Realisierung des 3D-Drucks. Auch wenn dieses Element letztlich, anders als ursprünglich geplant, nur zur optischen Verfeinerung diente, war die Auseinandersetzung mit 3D-Druck eine eindrückliche Erfahrung. Herr Schmidtner nahm sich die Zeit, mich in den an der Kanti befindlichen 3D-Drucker einzuführen und ermöglichte ein komfortables und teilweise recht kurzfristiges Drucken.

Ebenfalls ein grosses Dankeschön geht an Christian und Elisabeth Schwerin, welche mich dazu motivierten, mehr zu versuchen, als nur eine reine Berichterstattung zu schreiben. Auch gab mir Christian einfallsreiche Vorschläge für die Beseitigung von Problemen in den Arduinosketches.

Ausserdem bedanke ich mich natürlich bei meinem Betreuer Rainer Steiger sowie meinem Koreferenten Rafael Riederer für die Zusammenarbeit von Anfang an.

# 10. GLOSSAR

Die vorkommenden kursiv geschriebenen Begriffe sind hier kurz erklärt:

- **digital (Signal):** Ein digitales Signal kann laut Wikipedia als eine Folge von Zahlen aus einem Wertebereich angesehen werden.[29] Bei den Digital Pins des Arduinos besteht der Wertebereich nur aus zwei Zahlen, Null oder Eins.

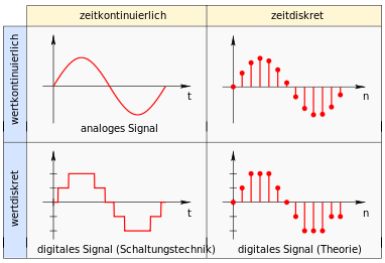



Abb. 23: Schema von analogen und digitalen Signalen

- **analog (Signal):** Bei analogen Signalen wird in der Regel die Spannung als physikalische Einheit herbeigezogen[30], die Arduino Analog Pins arbeiten mit einer Spannung von 0 bis 5V. Liegt also eine Spannung von 0V vor und man fragt den Wert des Pins ab, bekommt man Null. Als höchsten Wert, also bei 5V, erhält man (bei den meisten Arduino Chips) 1023.[31]

Abb. 24: Schraubklemme

- **Schraubklemmen:** siehe Abb. 24
- **Jumper:** Ein Kabel welches einfach in die Pins des Arduinos oder auch Steckbretter gesteckt werden kann.
- **HIGH/LOW:** Oft verwendet statt Null und Eins in Bezug zu den Arduino Pins
- **Drehmoment:** Drehmoment ist das Äquivalent von Kraft im Zusammenhang mit Drehbewegungen. Die Notwendigkeit einer neuen Einheit kann man einsehen, indem man sich überlegt, dass ein Unterschied besteht, ob die an einer Drehscheibe wirkende (Tangential-) Kraft nahe des Drehpunktes, oder weit von diesem entfernt wirkt. Entsprechend ist der Betrag des Drehmoments $M$ definiert als Kraft $F$ mal Hebelarm $\mathrm{l}$ (und der Vektor $\boldsymbol{M} = \mathbf{r}\times\mathbf{F}$ , wobei $\mathbf{r}$ der Ortsvektor zum Punkt an dem die Kraft wirkt, mit dem Drehpunkt als Koordinatenursprung und $\mathbf{F}$ der Vektor der wirkenden Kraft).[32]

  Für diese Arbeit ist vor allem die Tatsache wichtig, dass die Grösse des Drehmoments sowohl von der Grösse der wirkenden Kraft, als auch von der Länge des Hebelarms abhängig ist.

  **Elektronische Grundlagen:** Um hiermit nicht mehrere Seiten zu füllen möchte ich in diesem Zusammenhang nur auf vier Dinge kurz eingehen: Stromstärke, Spannung, ohmscher Widerstand und Spulen. Die Stromstärke beschreibt die Anzahl Elektronen

[29] Quelle : https://de.wikipedia.org/wiki/Digitalsignal
[30] Quelle : https://de.wikipedia.org/wiki/Analogsignal
[31] Quelle : https://www.arduino.cc/en/Tutorial/AnalogInputPins
[32] Quelle : Paul A. Tipler/Gene Mosca : *Tipler*. Springer-Verlag, Berlin, 2015

welche pro Zeiteinheit den Querschnitt des Leiters passieren. Sie ist dafür verantwortlich, dass sich ein Leiter erwärmt, je mehr desto höher die Stromstärke(bei gleichem Widerstand).[33] Die Spannung zwischen zwei Stellen A und B in einem Stromkreis bezeichnet die Energie, welche nötig ist, ein geladenes Teilchen von A nach B zu bewegen.[34] Vereinfacht kann man sich die Spannung als Druck und die Stromstärke als Wasservolumen pro Zeiteinheit in einem Wasserkreislauf vorstellen. Der ohmsche Widerstand zeigt an, wie gut ein Leiter Strom leitet, je geringer der ohmsche Widerstand desto besser die Leitfähigkeit.[35] Stromstärke ($\mathrm{I}$), Spannung ($\mathrm{U}$) und Widerstand ($\mathrm{R}$) stehen in einem Stromkreislauf in folgendem Zusammenhang: $\mathrm{U} = \mathrm{R} * \mathrm{I}$

Zu Spulen sei hier nur gesagt, dass sie zentrale Bestandteile von Elektromotoren sind, Magnetfelder erzeugen und dafür verantwortlich sind, dass in Elektromotoren der Widerstand (bzw. Gegenspannung) abhängig von der Drehgeschwindigkeit ist.[36]

- **Bit:** In einem Bit ist der Informationsgehalt von zwei Entscheidungsmöglichkeiten enthalten, er hat entweder den Wert 0 oder 1, oder wie im Umgang mit Arduino gern verwendet: *LOW* oder *HIGH*. («ein» *digitales* Signal)[37]
- **Byte:** 8 Bit[38]
- **Array:** Einen Array (auch Liste oder Feld genannt) kann man sich, zumindest für den Gebrauch in dieser Arbeit, wie einen Schrank mit mehreren Schubladen vorstellen. Der Schrank hat einen Namen (in unserem Fall «werte») und die Schubladen sind nummeriert, wobei zu beachten ist, dass mit Null als erster Nummer angefangen wird.[39] (Syntax siehe Kap. Aruino Sketches)
- **Integer(int):** Ein Datentyp, der ganzzahlige Werte enthält.[40] (Syntax siehe Kap. Aruino Sketches)
- **Character(char):** Ein Datentyp, deren Wert maximal einen Byte Speicher braucht, also in der Regel ein Zeichen (z.B. Buchstabe o. Zahl)[41](Syntax siehe Kap. Aruino Sketches)
- **String(str):** Der Datentyp String enthält eine Zeichenkette, welche in Anführungszeichen (strings) gesetzt wird.[42]

[33] Quelle : https://de.wikipedia.org/wiki/Elektrische_Stromst%C3%A4rke
[34] Quelle : https://de.wikipedia.org/wiki/Elektrische_Spannung
[35] Quelle : https://de.wikipedia.org/wiki/Elektrischer_Widerstand#Ohmscher_Widerstand
[36] Quelle : https://de.wikipedia.org/wiki/Gleichstrommaschine
[37] Quelle: https://de.wikipedia.org/wiki/Bit
[38] Quelle: https://de.wikipedia.org/wiki/Byte
[39] Quelle: https://de.wikipedia.org/wiki/Feld_(Datentyp)
[40] Quelle: https://de.wikipedia.org/wiki/Integer_(Datentyp)
[41] Quelle: https://de.wikipedia.org/wiki/Char_(Datentyp)
[42] Quelle : http://www.info-wsf.de/index.php/Datentyp_String

- **Float(float):** Sogenannte Gleitkommazahl, hat im Gegensatz zum *Integer* (eine begrenzte Anzahl) Nachkommastellen.[43]

[43] Quelle : https://de.wikipedia.org/wiki/Gleitkommazahl

# 11. REDLICHKEITSERKLÄRUNG

Ich erkläre hiermit, dass ich die vorliegende Arbeit eigenständig verfasst habe. Dort wo Hilfestellungen oder Quellen Eingang in die Arbeit gefunden haben, habe ich diese korrekt und vollständig bezeichnet.

Datum: 05.12.2016 Unterschrift: _____

# 12. ABBILDUNGSVERZEICHNIS