Carl-Reuther-Berufskolleg des Rhein-Sieg-Kreises in Hennef

TE6W Elektrotechnik

Arbeitstitel:	Dokumentation Cocktailmaschine
Fach:	IT (Selbstlernphase)
Klasse:	TE6W
Lehrer:	Hr. Hippenstiel

Gruppenarbeit vorgelegt von:

Dennis Kolvenbach
Südstraße 28
53797 Lohmar
Tel.: 0151/22349911
E-Mail: dennis@leckmeineeier.de

Niklas Krol
Guntherstraße 25
51147 Köln
Tel.: 01520/5647332
E-Mail: niklas@leckmeineeier.de

Datum der Abgabe: 13.06.2018

Inhaltsverzeichnis

1 Aufgabenstellung 4

2 Projektvorstellung 4

2.1 Funktionsbeschreibung 4

2.2 Projektumfang 5

3 Projektdurchführung 6

3.1 Mechanik-Konstruktion 6

3.2 Mechanik-Berechnungen 7

3.3 Mechanik-Fertigung 8

3.4 Elektro-Konstruktion 11

3.5 Elektronik-Testaufbau (Breadboard) 12

3.6 Elektronik-Fertigung 13

3.6.1 Treiber-Platinen 13

3.6.2 Spannungsverteilung 13

3.6.3 LCD-Ansteuerung 14

3.6.4 Grundplatte 14

3.6.5 Handbedienpult 15

3.7 Software-Entwicklung 16

3.7.1 Programmablaufplan 16

3.7.1.1 Hauptprogramm: 16

3.7.1.2 Rezept n 19

3.7.1.3 Debug 20

3.7.1.4 Positionier-Controller 21

3.7.1.5 Fahr-Controller 22

3.7.1.6 Ausgießen-Funktion 24

3.7.1.7 Hub-Controller 25

3.7.1.9 Einzelschritt Hub 26

3.7.1.10 Einzelschritt Fahren 26

3.7.2 Programmaufrufstruktur 27

3.7.3 Programmcode 29

3.7.3.1 Konstanten und Variablen-Definition und Deklaration 29

3.7.3.2 Setup-Teil 31

3.7.3.3 Haupt-Loop 32

3.7.3.4 Funktionen stepModeX() und stepModeY() 33

3.7.3.5 Funktionen manuelControlEncoder() und manuelControlButtons() 35

3.7.3.6 Einzelpulsfunktionen onePulseX() und onePulseY() 38

3.7.3.7 Funktion selectRecipe() 39
3.7.3.8 Funktion recipeDrink_n() 41
3.7.3.9 Funktionen drivePos(n), posStart() und posEnd() 42
3.7.3.10 Funktion driveX() 42
3.7.3.11 Funktion driveY() 44
3.7.3.12 Funktion pourY () 44
3.7.3.13 Funktion menuButtonCounterLCD() 45
3.7.3.14 Funktion updateLCD() 46
3.7.3.15 Funktionen text…() 47
3.7.4 Rezept-Auswahl 48
3.7.4.1 Zutatenauswahl 48
3.7.4.2 Cocktail-Rezept 1 (Screwdriver) 49
3.7.4.3 Cocktail-Rezept 2 (Fuzzy Novel) 49
3.7.4.4 Cocktail-Rezept 3 (Green Eyes) 49
3.7.4.5 Cocktail-Rezept 4 (Green Hurrican) 49
3.7.4.6 Cocktail-Rezept 5 (Green Sombrero[Shot]) 49
3.7.4.7 Cocktail-Rezept 6 (Grüne Wiese) 50
3.7.4.8 Cocktail-Rezept 7 (Grüne Wiese Plus) 50
3.7.4.9 Cocktail-Rezept 8 (Peach-O) 50
3.7.4.10 Cocktail-Rezept 9 (Tequila Sunset) 50
3.7.4.11 Cocktail-Rezept 10 (Barcardi Sunset) 50
3.8 Zeitplanung und Ablauf 51
3.9 Probleme im Projektablauf 52
3.9.1 Motortreiber zu schwach 52
3.9.2 Schrittmotoren ruckeln bei der Rampenfahrt 52
3.9.3 Serial.Print bremst die Pulspausenzeiten 52
3.9.4 Lieferprobleme Zahnräder 52
3.9.5 Falsche Länge der Haupt-Zahnriemen für die Hubfunktion 52
3.9.6 Fehlende Pull-Down-Widerstände ohne angestecktes Handbedienpult 52

4 Materialdisposition 53

5 Fazit 54

1 Aufgabenstellung

Es sollte ein sinnvolles Projekt geplant und durchgeführt werden, in dem wenigstens ein Schrittmotor (Stepper) mittels eines Arduino angesteuert wird. Dabei gab es vornehmlich keinerlei weitere Beschränkungen, bis auf die Prämisse, dass komplexere Projekte eine höhere Aussicht auf eine gute Benotung besitzen.

2 Projektvorstellung

Wir entschieden uns dazu, eine Cocktailmaschine zu bauen. Diese soll ein Glas mittels Schrittmotor unter verschiedene Portionierer verfahren und dieses dort mittels eines zweiten Motors ausheben, so dass der entsprechende Portionierer ausgelöst wird. Welche Portionierer (und damit, welche Zutaten) in welcher Reihenfolge angefahren werden, kann im Vorfeld als Rezept im Arduino-Programmcode hinterlegt werden.

Dabei wurde bewusst der Fokus nicht auf Effizienz und Schnelligkeit der Cocktailmaschine gelegt (denn hier wäre eine Maschine basierend auf Pumpenmotoren, ohne bewegte Masse deutlich effektiver), sondern ein möglichst anspruchsvoller Aufbau, der das reibungslose Zusammenspiel mehrerer Komponenten erfordert, um eine Komplexität zu erreichen, die eines Techniker-Schulprojekts angemessen ist.

2.1 Funktionsbeschreibung

Nach dem das leere Cocktailglas auf die Plattform, welche sich in der Grundstellung (Parkposition) befindet gestellt wurde, kann mittels „Select-Taster“ das gewünschte Cocktailrezept ausgewählt werden. Welches Rezept dabei gerade ausgewählt wurde, ist über das LCD ersichtlich. Im Anschluss kann der Vorgang mit einem Druck auf den „Start-Taster“ gestartet werden. Daraufhin fährt die Plattform samt Glas, getrieben über 4 Spindelantriebe zunächst vollständig in der Y-Achse nach unten, um das Glas unter den Portionierern in X-Achse bewegen zu können.

Danach fährt das Glas nacheinander unter die Portionierer, die im jeweiligen Rezept vorgegeben sind. Zum Auslösen eines Portionierers wird das Glas mit Kraft unter die Portionierer gehoben, so dass der Portionierer 3cl Flüssigkeit ausgibt. Nach dem alle, dem Rezept zugehörigen Portionierer, in der vorgegebenen Anzahl und Reihenfolge angefahren wurden, fährt das Glas wieder in die Grundstellung und es wird über das LCD die Meldung ausgegeben, dass der Cocktail fertig ist.

Dabei erfolgt die gesamte Positionierung ausschließlich über eine entsprechende Anzahl an Pulsen, die die Schrittmotoren daraufhin ausführen. Sollten dabei einmal Schritte „verloren gehen“ und die Achsen nicht mehr sauber auf dem Referenzpunkt parken, können beide Achsen mittels eines mobilen Handbedienpults in den Referenzpunkt verfahren werden. Dazu können die Achsen mittels Taster (X+, X-, Y+, Y-) „schnell“ verfahren werden. Die Feinpositionierung am Referenzpunkt erfolgt daraufhin mittels Endlos-Dreh-Encoder, welcher die Schrittmotoren in Einzelschritten fahren kann.

Des Weiteren besitzt die Cocktailmaschine ein „verstecktes“ Service-Menü im LCD, über das die Betriebszustände des Handbedienpults dargestellt werden, aber auch beide Achsen im Teilautomatikbetrieb zwischen den Einzelpositionen verfahren werden können.

2.2 Projektumfang

Das Projekt wurde vollständig in Eigenregie konstruiert, gebaut und programmiert. Es wurde dabei weder auf eine vorgefertigte Mechanik, noch eine vorgefertigte Elektronik, noch ein vorgefertigtes Programm zurückgegriffen.

Im Rahmen der Projektarbeit wurden folgende Punkte abgearbeitet:

- Mechanik-Konstruktion
  - Konstruktion des Aufbaus in 3D mit SolidWorks
  - Berechnung von Drehmomenten und Drehzahlen
- Mechanik-Fertigung
  - Fertigung von Metall-Teilen
  - Fertigung der Holz-Zuschnitte
  - Zusammenbau und Anpassung der Einzelteile
- Elektronik-Konstruktion
  - Schaltungsentwurf mit Fritzing
  - Platinen-Entwurf mittels handschriftlicher Skizzen
- Elektronik-Fertigung (Testaufbau)
  - Testaufbau der Schaltung auf Breadboards
  - Test der Einzelschaltungen mittels kleiner Programmabschnitte
- Elektronik-Fertigung (Endausbau)
  - Aufbau und Fertigung der Platinen (Lochraster)
  - Montage der Platinen auf Grundträgerplatte
  - Verkabelung des Gesamtsystems
- Software-Entwicklung
  - Funktionsentwicklung mittels PAP-Designers
  - Schreiben des Arduino-Programms
- Inbetriebnahme
  - Einrichtung und Ausrichtung der Mechanik-Komponenten
  - Bugfixen der Software und anpassen der Grenzwerte
  - Prüfen und Testen des Elektronik-Aufbaus
  - Stresstest der Maschine und Korrektur aufgetretener Fehler
- Dokumentation
  - Verschriftlichung dieser Projektdokumentation

3 Projektdurchführung

Die Projektdurchführung gliederte sich über mehrere Bereiche (siehe Abschnitt 2.2 Projektumfang), die in Eigenregie durchgeführt wurden. Im Folgenden alle Details zur Projektdurchführung, anhand der einzelnen Arbeitsschritte.

3.1 Mechanik-Konstruktion

Die Mechanik-Konstruktion wurde vollständig in SolidWorks getätigt. Dabei wurden alle benötigten Komponenten zunächst als Einzelteile konstruiert und schlussendlich zur vollständigen Maschine zusammengefügt.

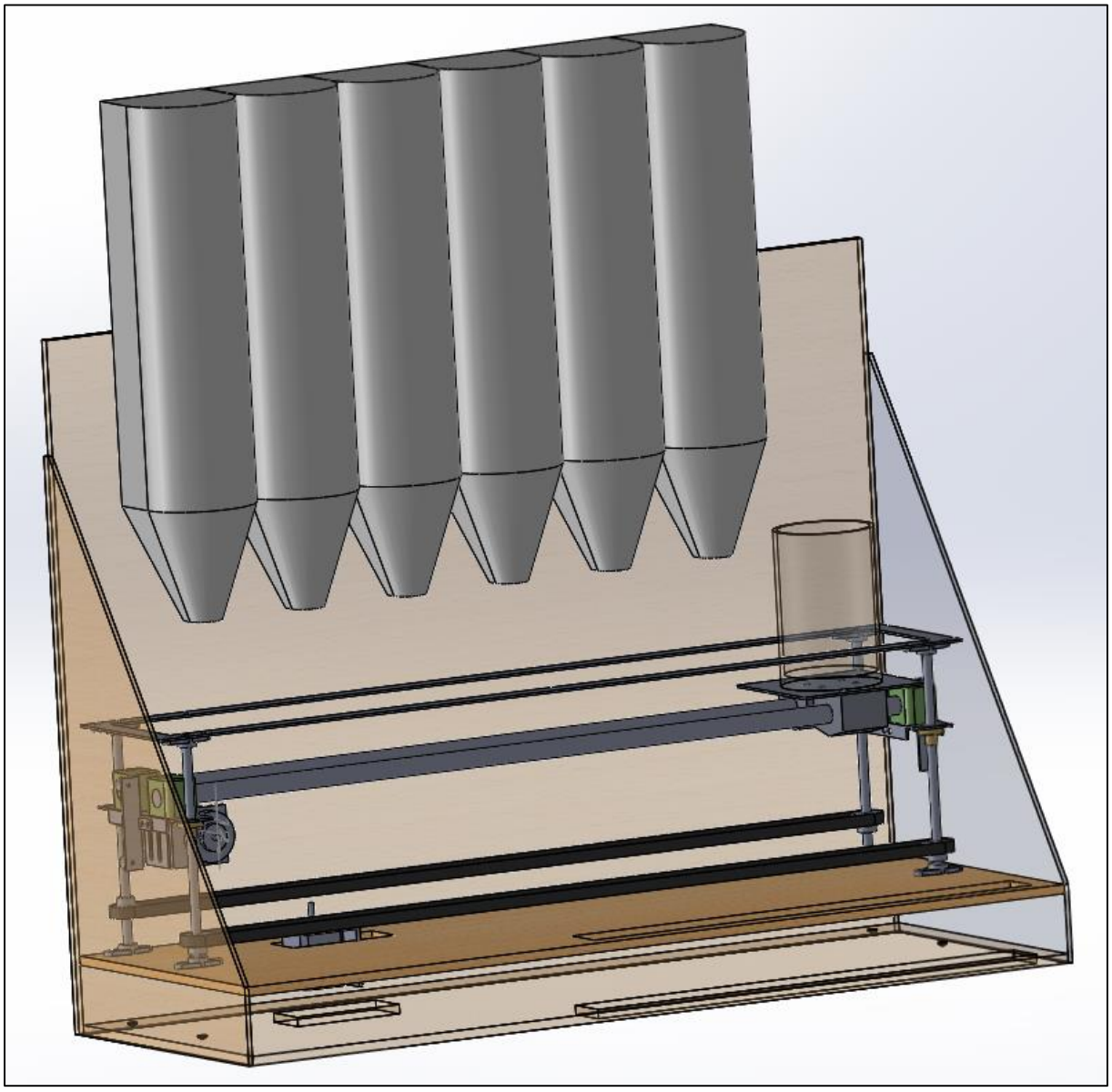

Abb. 1: Gesamtansicht[1]

1 Abb. 1: Gesamtansicht

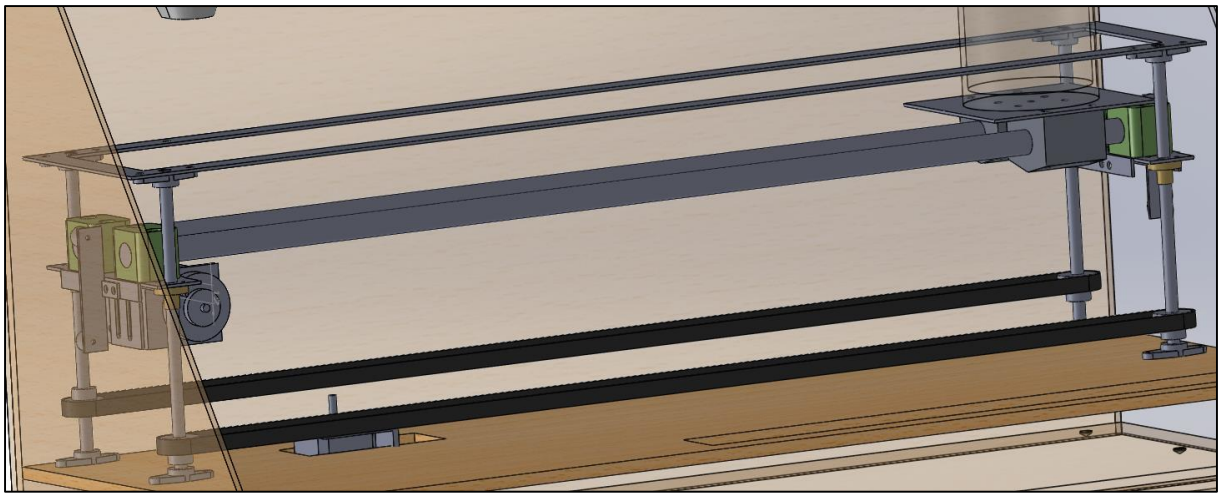

Abb. 2: Rückansicht[2]

Dabei wurde ein besonderes Augenmerk auf etwaige Störkanten gelegt, um spätere Probleme bereits im Vorfeld auszumerzen. Zudem ergaben sich aus der 3D Konstruktion alle benötigten 2D-Fertigungsunterlagen.

3.2 Mechanik-Berechnungen

Nachdem die Grundkonstruktion stand, mussten noch die erforderlichen Drehmomentberechnungen durchgeführt werden, um sicherzustellen, dass die Maschinen auch Funktioniert. Die Berechnung für das Drehmoment des Hubmotors erschließt sich wie folg:

Geg.: $m_{glas} = 300\ g\ ; m_{Flüssigkeit} = 300\ g\ ;\ m_{Verschub} = 400\ g\ ;\ m_{Liniarführung} = 2000\ g\ ; l_{Hub} = 70\ mm\ ; t_{Hub} = 5\ s\ ; g = 9{,}81 \frac{m}{s^2}\ ;\ P_{Gewindesteigung} = 2 \frac{mm}{U}$

Ges.: $m_{gesamt}\ ;\ n\ ; F_G ; F$

$$m_{gesamt} = m_{glas} + m_{Flüssigkeit} + m_{Verschub} + m_{Liniarführung}$$

$$m_{gesamt} = 300\ g + 300\ g + 400\ g + 2000 g = 3\ kg$$

$$n = \frac{l_{Hub}}{P_{Gewindesteigung}\ *\ t_{Hub}}$$

$$n = \frac{70\ mm}{2 \frac{mm}{U} * 5\ s} = 7 \frac{U}{s}$$

$$F_G = m_{gesamt} * g$$

$$F_G = 3\ kg * 9{,}81 \frac{m}{s^2} = 28{,}43\ N$$

$$F = F_G * P_{Gewindesteigung}$$

$$F = 28{,}43\ N * 2 \frac{mm}{U} = 58{,}82 Nmm = 5{,}8 Ncm$$

2 Abb. 2: Rückansicht

Eine Nema 17 kann 57 Ncm Drehmoment aufbringen und reicht somit aus. Auf Grund der berechneten Drehzahl lassen sich die Stepps pro Sekunde errechnen, so wie die benötigten Stepps um die Hubhöhe zu erreichen.

Geg.: $\frac{Steps}{U} = 200(Full)Steps = 400(Half)Steps$; $n_{Drehzahl} = 7\frac{U}{s}$; $n_{Umdrehungen} = 35\ U$

Ges.: $Steps\ pro\ Sekunde\frac{St}{s}$; $Anzahl\ Steps\ n_{Steps}$

$$\frac{St}{s} = \frac{Steps}{U} * n_{Drehzahl}$$

$$\frac{St}{s} = 400(Half)Steps * 7\frac{U}{s} = 2800\ \frac{St}{s}$$

$$n_{Steps} = \frac{Steps}{U} * n_{Umdrehungen} = 14.000\ Steps$$

$$n_{Steps} = 400(Half)Steps * 35\ U = 14.000$$

3.3 Mechanik-Fertigung

Dank der Unterstützung unseres Arbeitgebers (Fa. SIVAplan GmbH) durften wir nach Feierabend die Elektro-Werkstatt für unsere Zwecke nutzen. Dort wurden neben den Holzzuschnitten auch alle benötigten Metallteile gefertigt (zugeschnitten, gekantet, geschliffen und gebohrt). Außerdem wurden die Schaltungen weiterentwickelt und erste Testläufe der Motoren getätigt.

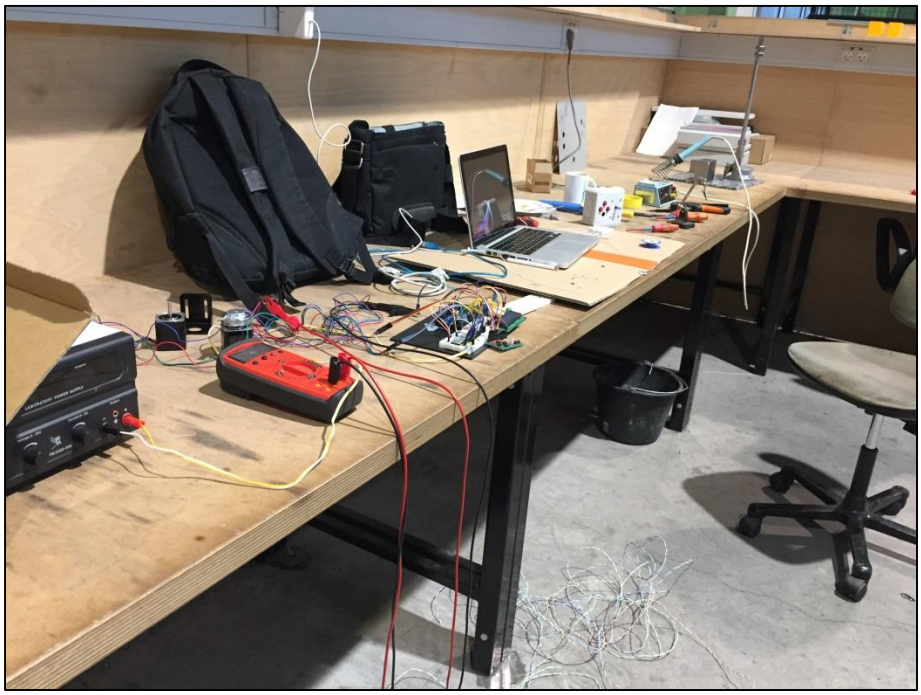

Abb. 3: Elektro-Werkstatt

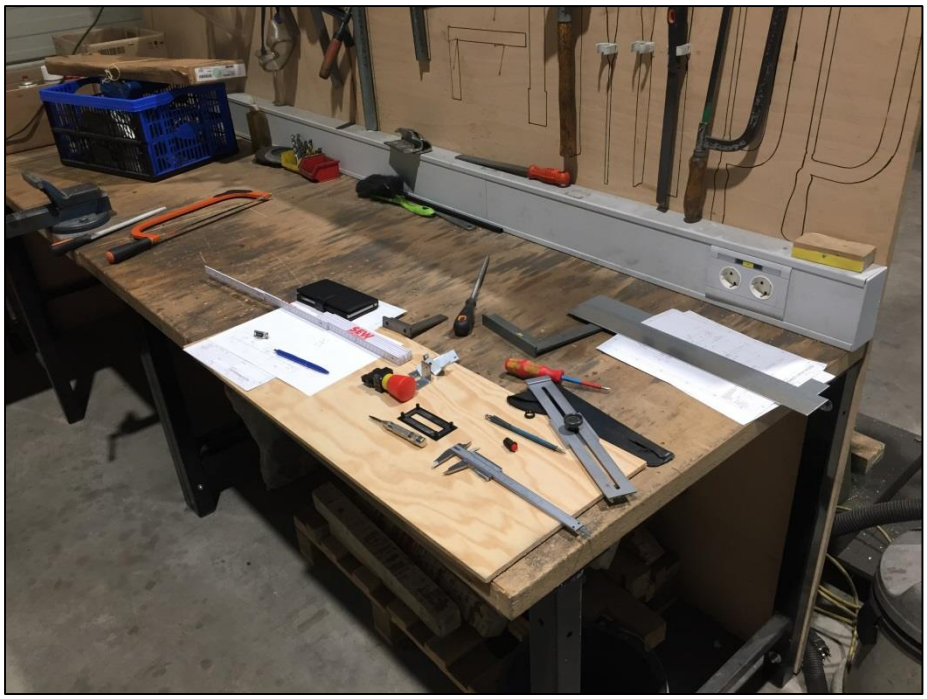

Abb. 4: Metallbearbeitung

Abb. 5: Holz-Zuschnitte

Der weitere Aufbau erfolgte teils noch in der Werkstatt der Fa. SIVAplan, teils zuhause.

Abb. 6: Cocktailmaschine bei Inbetriebnahme

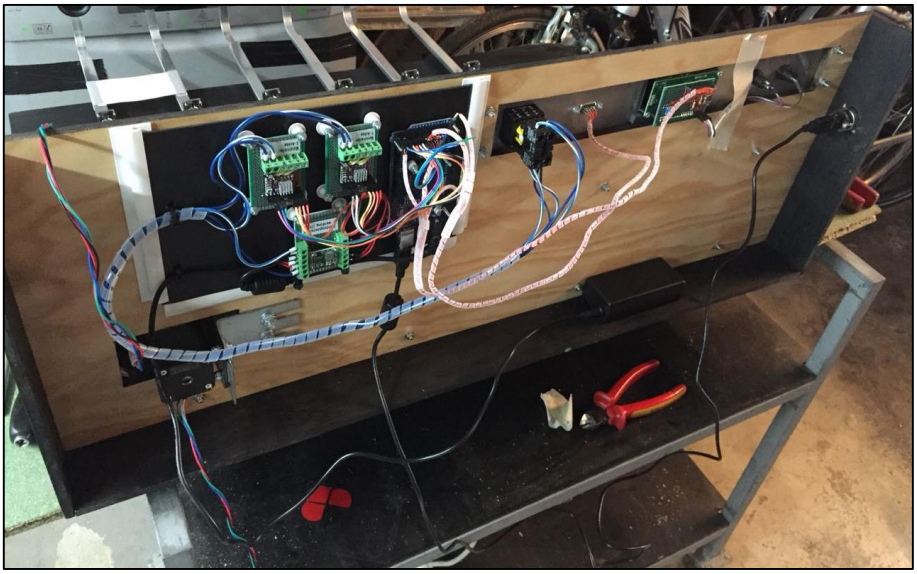

Abb. 7: Elektronik Cocktailmaschine

3.4 Elektro-Konstruktion

Nachdem der Funktionsumfang der Maschine vollständig definiert war, wurde die Schaltung basierend auf einem Breadboard-Testaufbau projektiert. Die Schaltung wurde nach erfolgreichen Testläufen auf mehrere Lochraster-Platinen aufgeteilt und diese gefertigt.

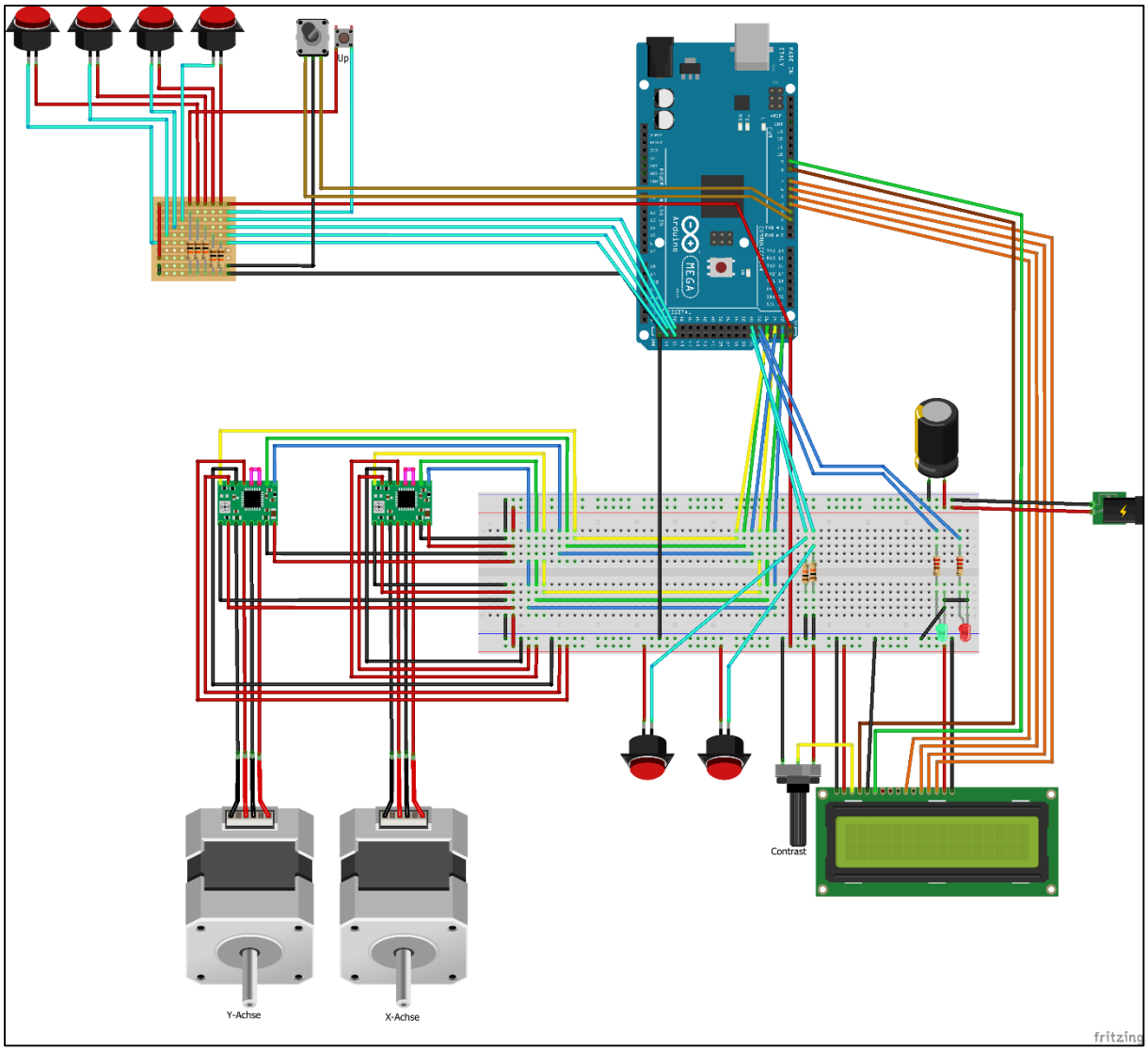

Abb. 8: Fritzing Schaltplan[3]

Der hier abgebildete Breadboard-Aufbauplan stellt dabei den ersten Entwurf der Schaltung dar. Im Laufe des Projekts wurde dieser an vereinzelten Stellen noch angepasst. So wurde sich beispielsweise im Nachhinein gegen eine Statusanzeige auf LED-Basis entschieden, sondern stattdessen das LCD genutzt.
Außerdem wurde die Schrittmodi-Umschaltung nicht wie hier dargestellt Hardware-Verdrahtet ausgeführt, sondern über Digital-Pins am Arduino Software-Umschaltbar.

[3] Abb. 8: Fritzing Schaltplan

3.5 Elektronik-Testaufbau (Breadboard)

Nachdem die Schaltung mittels Fritzing vollständig geplant wurde, wurde die Schaltung auf Breadboards aufgebaut, um sowohl die Software-Entwicklung voranzutreiben, als auch sicher zu sein, dass die Schaltung funktioniert.

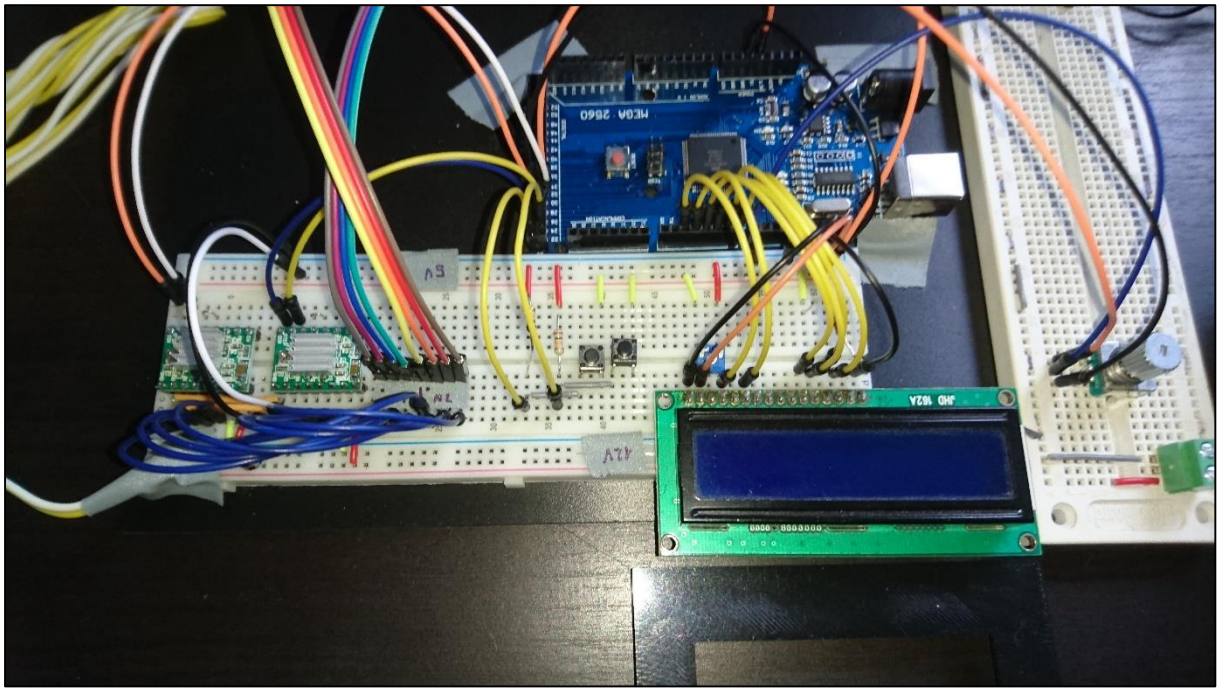

Abb. 9: Breadboard-Aufbau

Dabei fiel auf, dass die zunächst eingeplanten A4988 Treiber (wie im Bild zu sehen) nicht leistungsstark genug für die gewünschten Funktionen waren, woraufhin auf die DRV8825-Treiber gewechselt wurde.

3.6 Elektronik-Fertigung

Damit die auf dem Breadboard getesteten Schaltungen auch ihren Weg in die fertige Cocktailmaschine finden und dabei sowohl optisch ansprechend als auch transport- und erschütterungssicher eingebaut werden können, wurden diese auf Lochrasterplatinen gelötet.

3.6.1 Treiber-Platinen

Als Motor-Treiber dienen im Endausbau die DRV8825-Treiber. Diese wurden zum besseren Handling auf Lochrasterplatinen gelötet. Dabei wurde jeweils ein vom Hersteller empfohlener Puffer-Kondensator mit verbaut. Der Motor-Anschluss erfolgt genau wie die 12 V Spannungsversorgung der Motoren über Schraubklemmen und größerem Querschnitt, um dem Temperaturanstieg der Motorleitungen bei Dauerbetrieb gerecht zu werden. Die Steueranschlüsse wurden auf Stiftsockel-Leisten ausgeführt um später eine einfachere Verdrahtung zum Arduino zu ermöglichen.

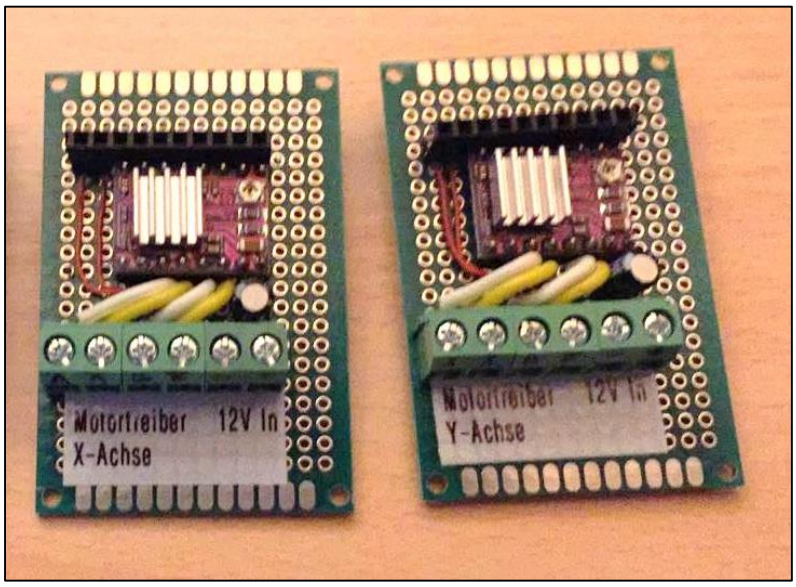

Abb. 10: Motortreiber

3.6.2 Spannungsverteilung

Damit die Cocktailmaschine nur eine Spannungsversorgung bekommt, erfolgt diese über ein Stecker-Netzteil (230V AC -> 12 V DC), welches 5 A liefern kann. Um die 12 V nun auf die Motortreiber zu verteilen, sowie den Arduino mit 5 V Spannung zu versorgen, wurde eine Spannungsverteilungsplatine gebaut. Diese besitzt einen Spannungswandler von 12 V DC auf 5 V DC, sowie genügend Klemmen um beide Spannungen innerhalb der Cocktailmaschine verteilen zu können.

Um eine größtmögliche Sicherheit bieten zu können, wurde die Cocktailmaschine auch mit einem Not-Aus-Taster ausgestattet, welche die Verbindung der 12 V-Versorgungsspannung von der Spannungsverteiler-Platine zu den Treiber-Platinen kappt und damit die Maschine in einen sicheren Zustand versetzt.

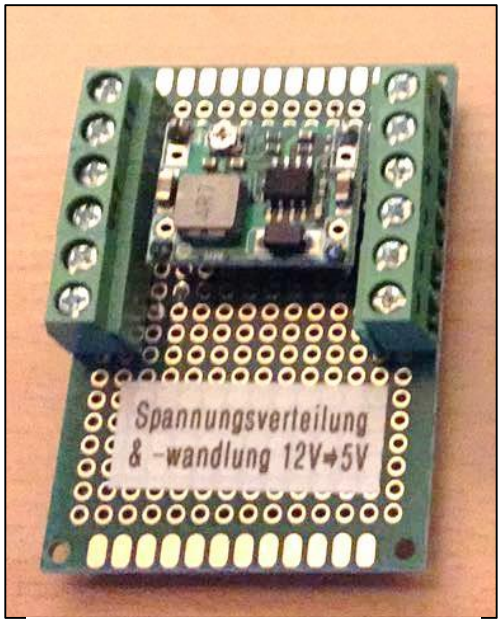

Abb. 11: Spannungsverteilung

3.6.3 LCD-Ansteuerung

Zur Aufnahme des LCD wurde ebenfalls eine Platine gebaut. Diese beinhaltet neben dem Trimmer für den Kontrast des LCD und dem Vorwiderstand für die Hintergrundbeleuchtung des LCD auch zwei Pull-Down-Widerstandsschaltungen um die Start & Select-Taster direkt mit aufnehmen zu können und alles nur mit einer Flachleitung auf die Arduino-Pins verdrahten zu müssen.

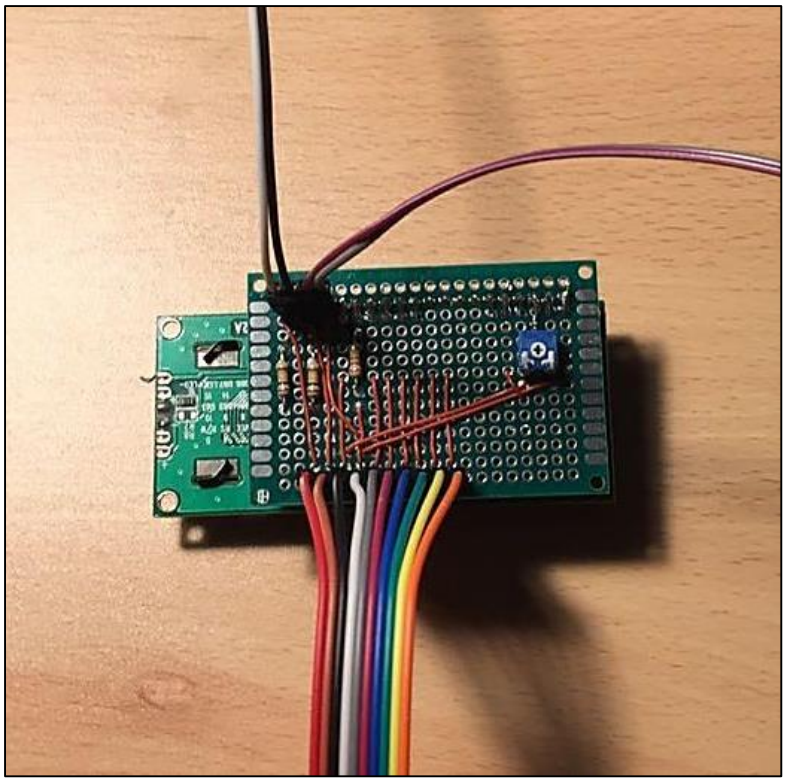

Abb. 12: LCD-Platine

3.6.4 Grundplatte

Die Kunststoff-Grundplatte dient der Aufnahme der folgenden Platinen:

- Arduino Mega
- Spannungsverteilungsplatine
- 2x Treiberplatinen für Motoren

Durch die Montage dieser Platinen auf der Grundplatte und die Führungsschienen zur Aufnahme der Grundplatte, ist es später möglich ohne weiteres die Platinen im verdrahteten Zustand aus der Maschine zu entnehmen.

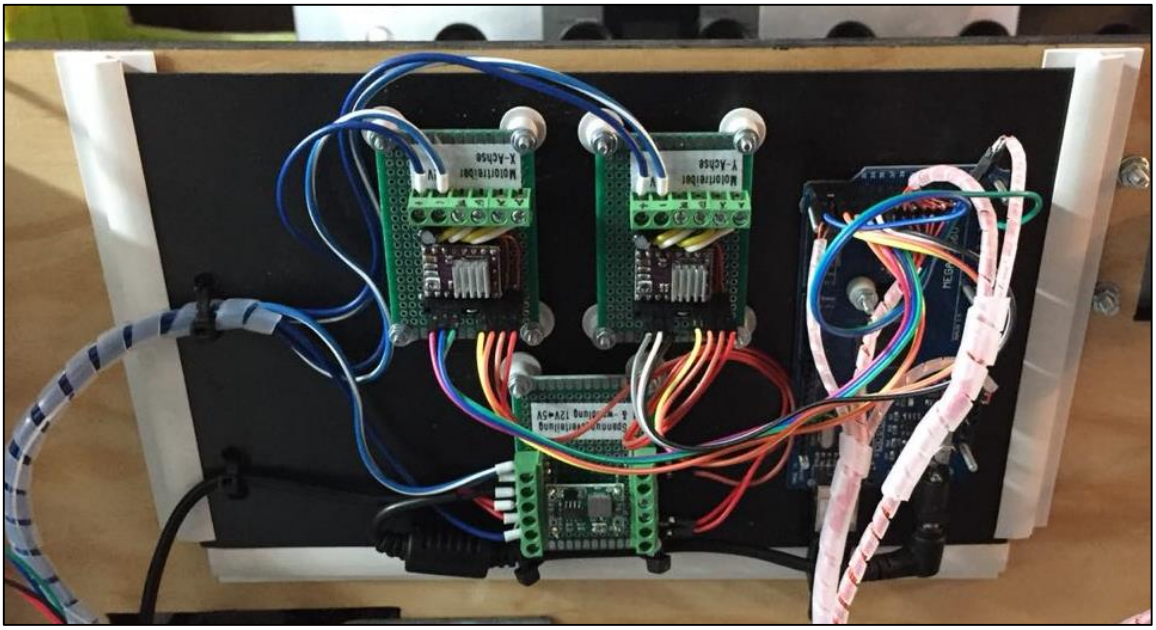

Abb. 13: Elektronik Cocktailmaschine

3.6.5 Handbedienpult

Um die beiden Achsen der Cocktailmaschine referenzieren zu können wurde sich dazu entschlossen, eine Handbedienung beider Achsen einzubauen. Damit diese Taster aber nicht das Gesamtbild des MMI (Mensch-Maschinen-Interface) der Cocktailmaschine stören und den Benutzer unnötig verwirren, wurde das Handbedienpult in ein optional-ansteckbares externes Gehäuse gebaut. Dieses externe Gehäuse kann mittels D-Sub-Stecker mit der Cocktailmaschine verbunden werden und diese daraufhin bedienen. Auf der Oberfläche des Handbedienpults befindet sich neben den Tastern zur Schnellpositionierung beider Achsen auch ein 2-Stelliger (eingedrückt und ausgefahren) Endlosencoder, der die Feinpositionierung beider Achsen im Einzelschrittbetrieb erlaubt.

Abb. 14: Handbedienpult

3.7 Software-Entwicklung

Die Software stammt nahezu vollständig aus Eigenentwicklung und nutzt lediglich die Bibliothek für die Ansteuerung des LCD. Zur besseren Organisation bei der Programmerstellung wurde die Software möglichst modular aufgebaut und in mehrere Dateien gesplittet, die später vom Compiler beim Kompilieren und übertragen auf den Arduino zu einer Datei zusammengesetzt werden.

Zur Entwicklung der Software wurden zunächst die Grundfunktionen mittels eines Programmablaufplans ermittelt.

3.7.1 Programmablaufplan

Der Programmablaufplan wurde mit dem PAP-Designer erstellt und soll die Grundfunktionalität der Software darstellen. Bereits in diesem Stadium wurde die Struktur der Software festgelegt und die Funktionen ausgegliedert.

Hinweis:

Zur besseren Lesbarkeit wurden die PAPs teilweise in zwei Teile zerschnitten.

3.7.1.1 Hauptprogramm:

Im Hauptprogramm ist die Benutzerführung geregelt. Von hier aus werden die Unterprogramme

- *Rezept n*
- *Debug*

aufgerufen. Wobei Rezept n mehrfach existiert und nur exemplarisch für ein Cocktailrezept steht.

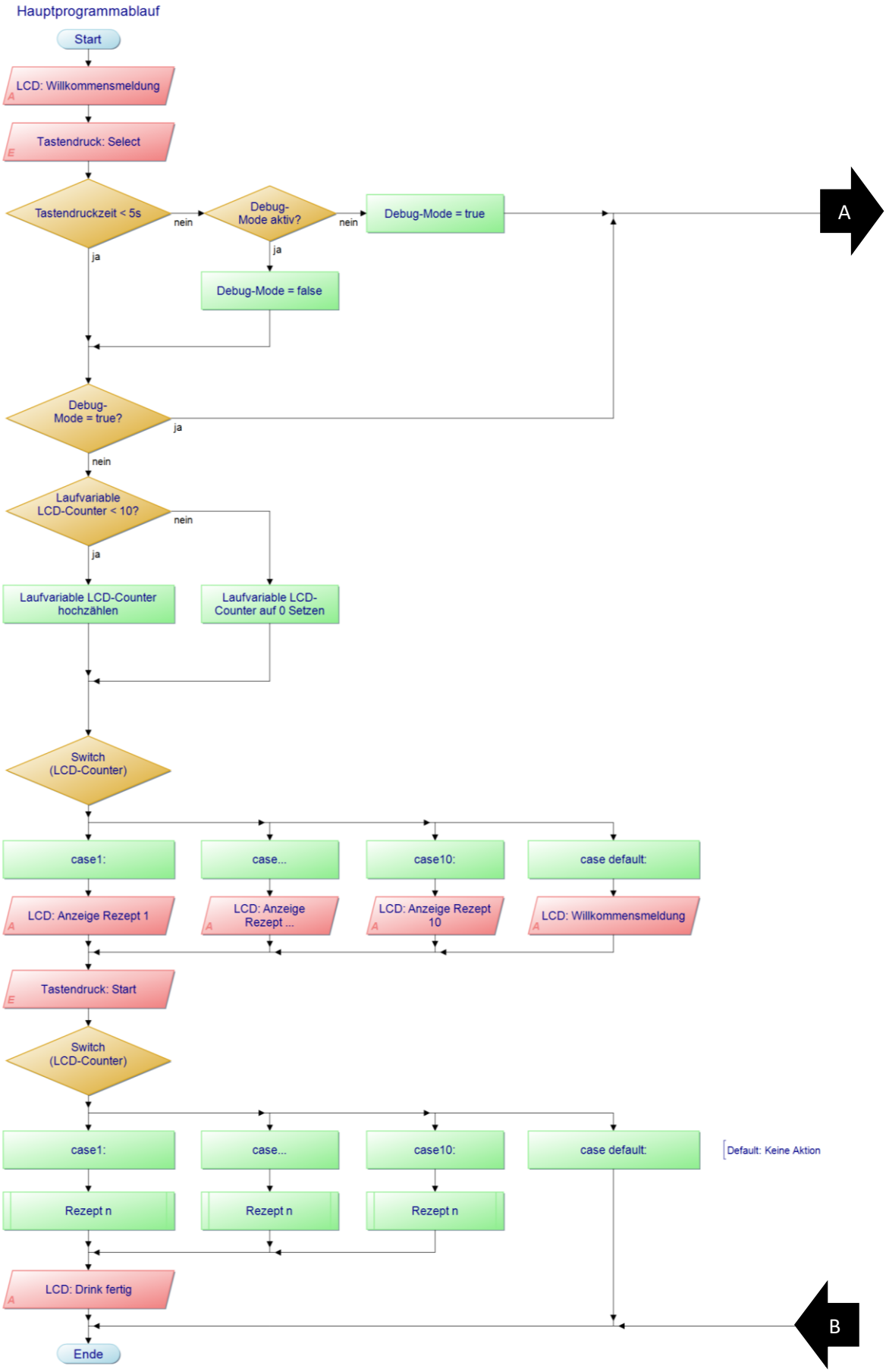

Abb. 15:PAP Hauptprogramm (Teil 1 von 2)

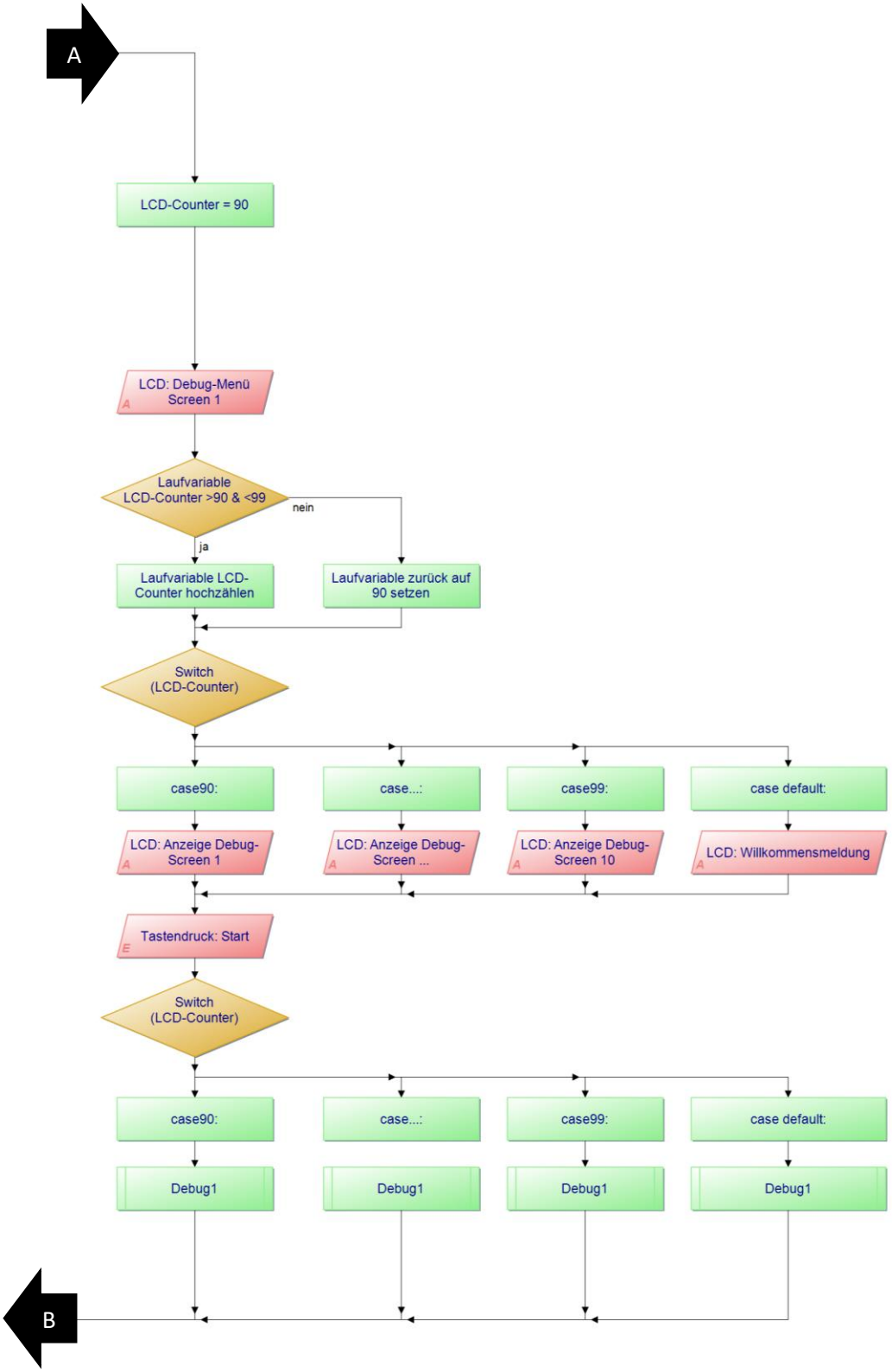

Abb. 16: PAP Hauptprogramm (Teil 2 von 2)

3.7.1.2 Rezept n

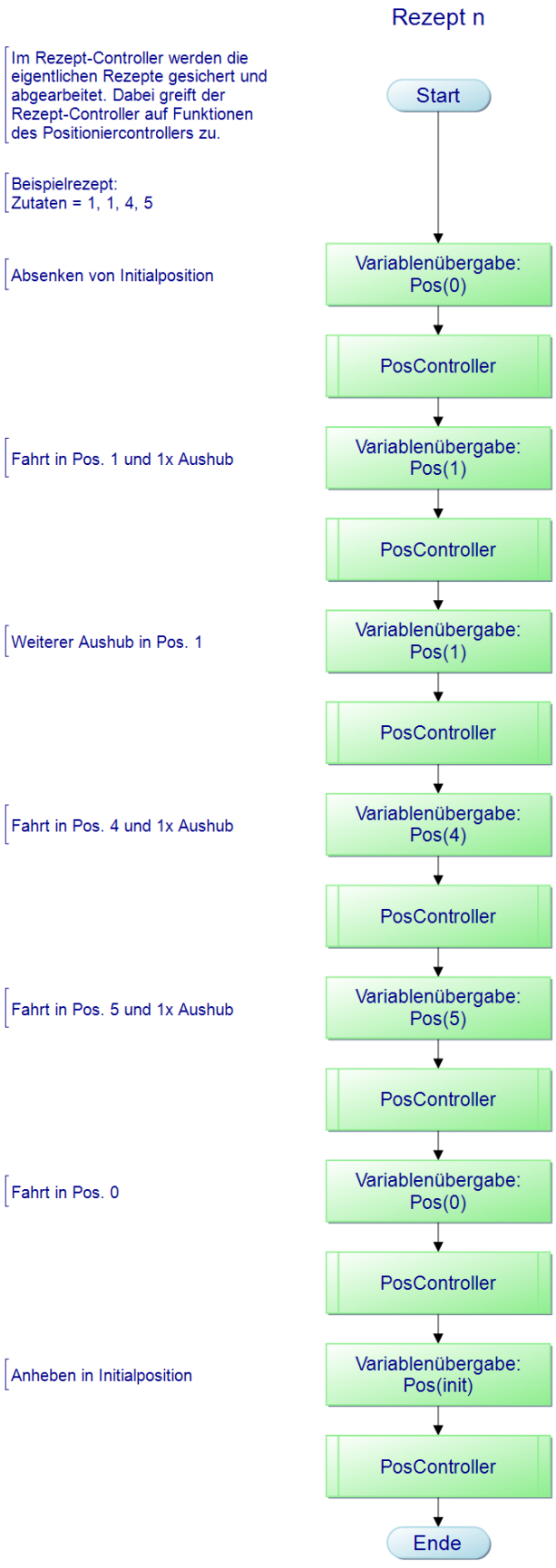

Abb. 17: PAP Rezepte

3.7.1.3 Debug

In der Debug-Funktion werden, je nach aktuell aufgerufenem Menüeintrag Einzelfahrten ausgelöst.

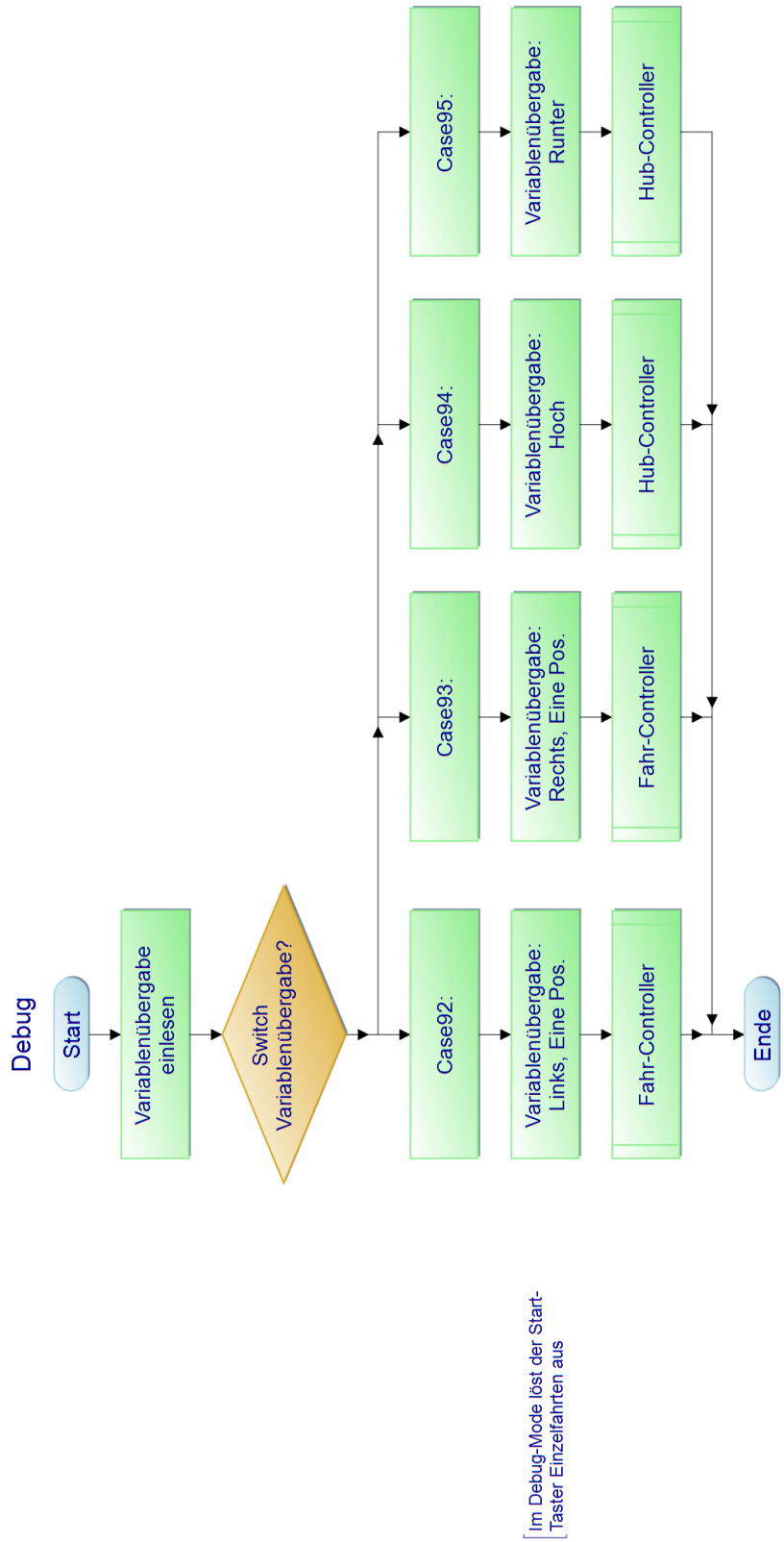

Abb. 18: PAP Debug-Menü

3.7.1.4 Positionier-Controller

Der Positioniercontroller merkt sich die aktuelle Position und ermöglicht das Absolute anfahren (mithilfe der Fahr-Funktion) von Positionen, was die Rezeptverwaltung deutlich vereinfacht.

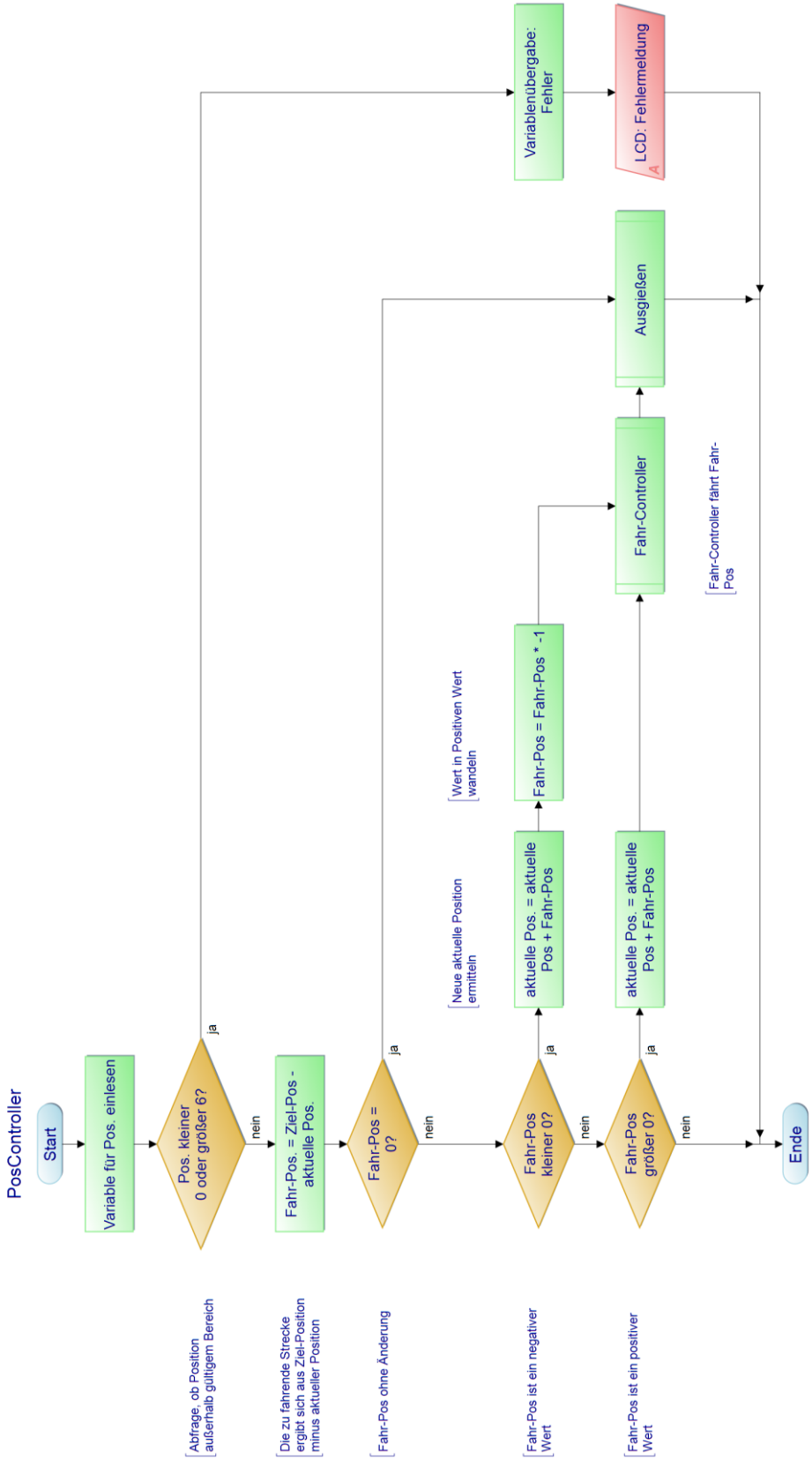

Abb. 19:PAP Positioniercontroller

3.7.1.5 Fahr-Controller

Der Fahr-Controller steuert das eigentliche verfahren in X-Achse. Dabei bekommt er die Anzahl der zu fahrenden Positionen, die Richtung und einen Korrekturwert übergeben und fährt daraufhin eine Rampenfahrt zur Zielposition.

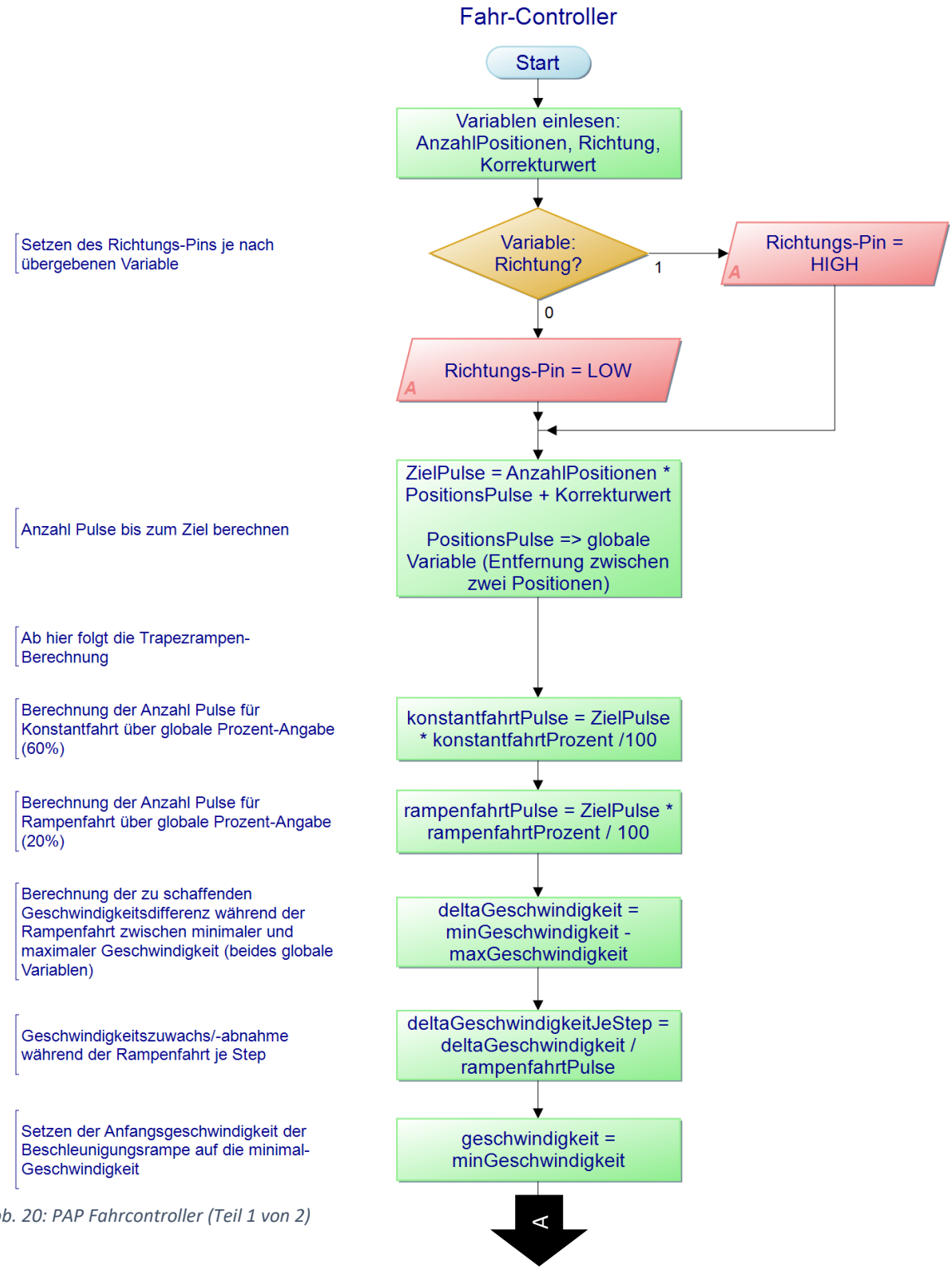

Abb. 20: PAP Fahrcontroller (Teil 1 von 2)

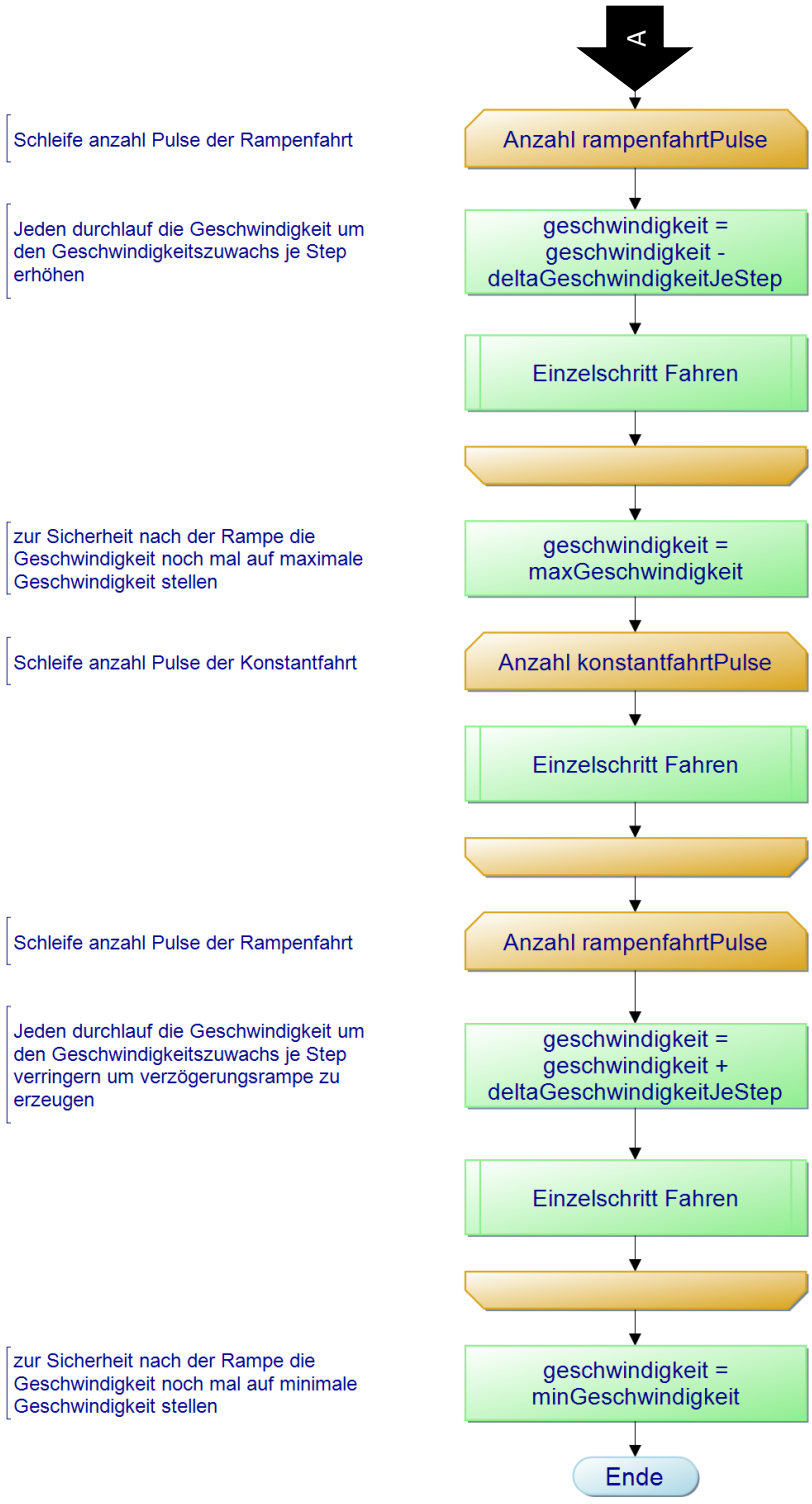

Abb. 21: PAP Fahrcontroller (Teil 2 von 2)

3.7.1.6 Ausgießen-Funktion

Die Ausgießen-Funktion fährt automatisch einen Ausgieß-Zyklus und erleichtert das häufige aufrufen.

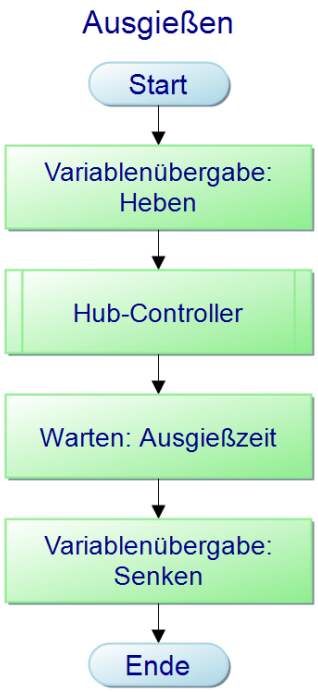

Abb. 22: PAP Ausgießfunktion

3.7.1.7 Hub-Controller

Der Hub-Controller ermöglicht das Verfahren in Y-Achse, durch Angabe von Richtung und Korrekturwert.

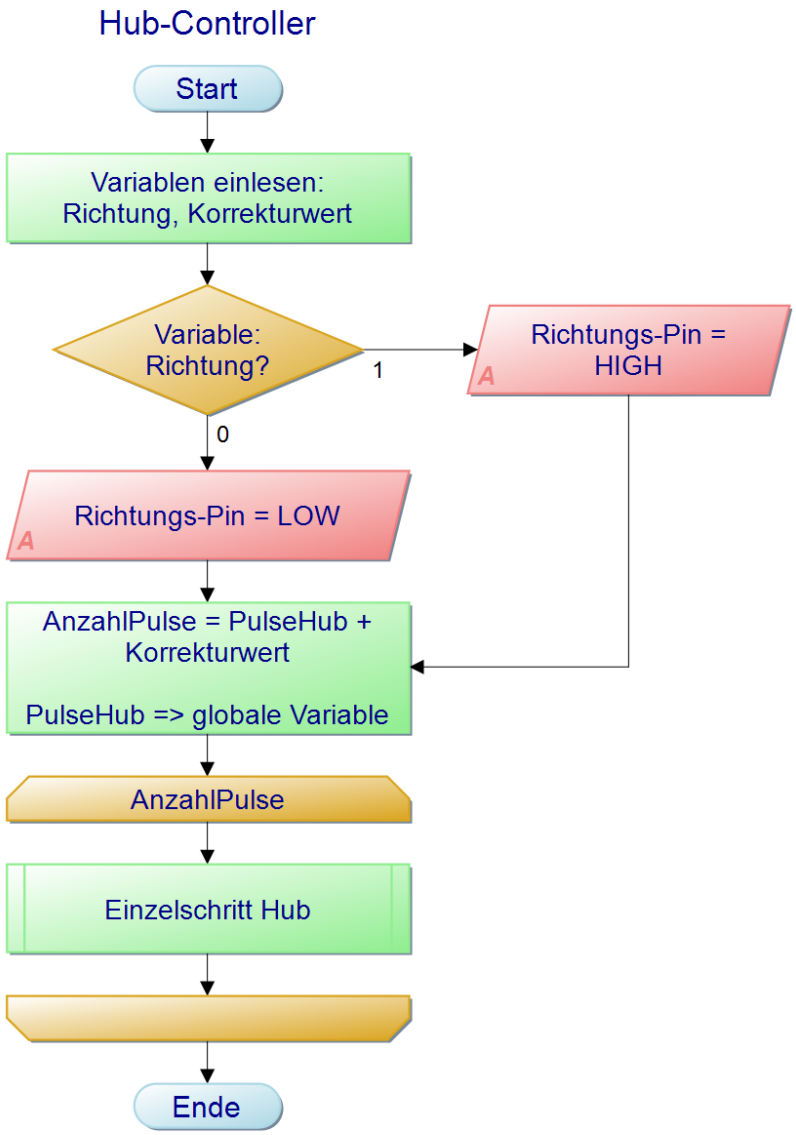

Abb. 23: PAP Hubcontroller

3.7.1.9 Einzelschritt Hub

In den Einzelschritt-Funktionen wird der Puls für die Schrittmotoransteuerung generiert. Dabei ist die Delayzeit variabel und wird über die Fahr- bzw. Hub-Controller gesteuert.

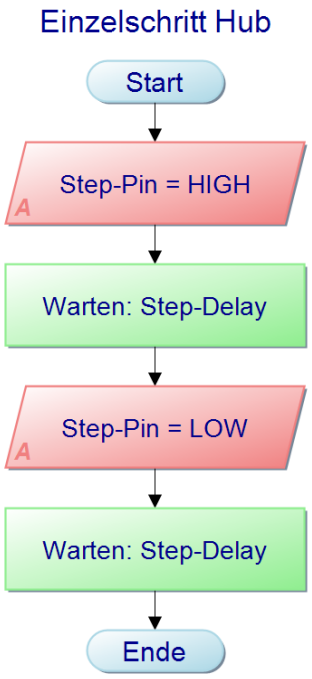

Abb. 24: PAP Einzelschritt Hub

3.7.1.10 Einzelschritt Fahren

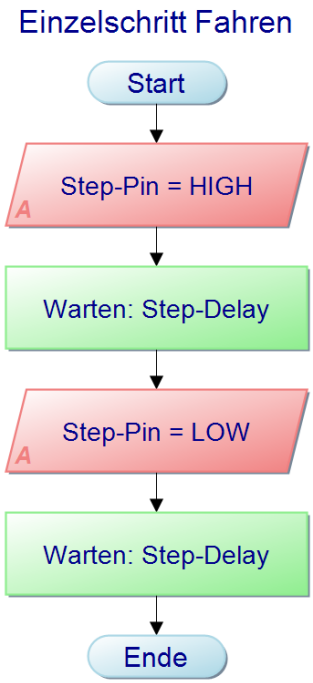

Abb. 25: PSP Einzelschritt Fahren

3.7.2 Programmaufrufstruktur

Um das Programm möglichst modular gestalten zu können und damit spätere Erweiterungen möglichst einfach ermöglichen zu können, werden im Haupt-Loop nur alle Eingänge eingelesen und alle Unterfunktionen aufgerufen. Die einzelnen Unterfunktionen besitzen dabei teilweise noch weitere Unterfunktionen, wie man an der Programmablaufplanung bereits sehen konnte. Zum besseren Verständnis, wie die einzelnen Funktionen zusammenhängen, nachfolgend die grafische Ableitung der Programmaufrufstruktur:

Hinweise:

- *Zur besseren Lesbarkeit wurde die Grafik vertikal in zwei Teile zerschnitten.*
- *Da das Projekt evtl. später im Internet veröffentlich werden soll, wurde der komplette Programmcode auf Englisch kommentiert und alle Variablen und Funktionen englisch benannt.*

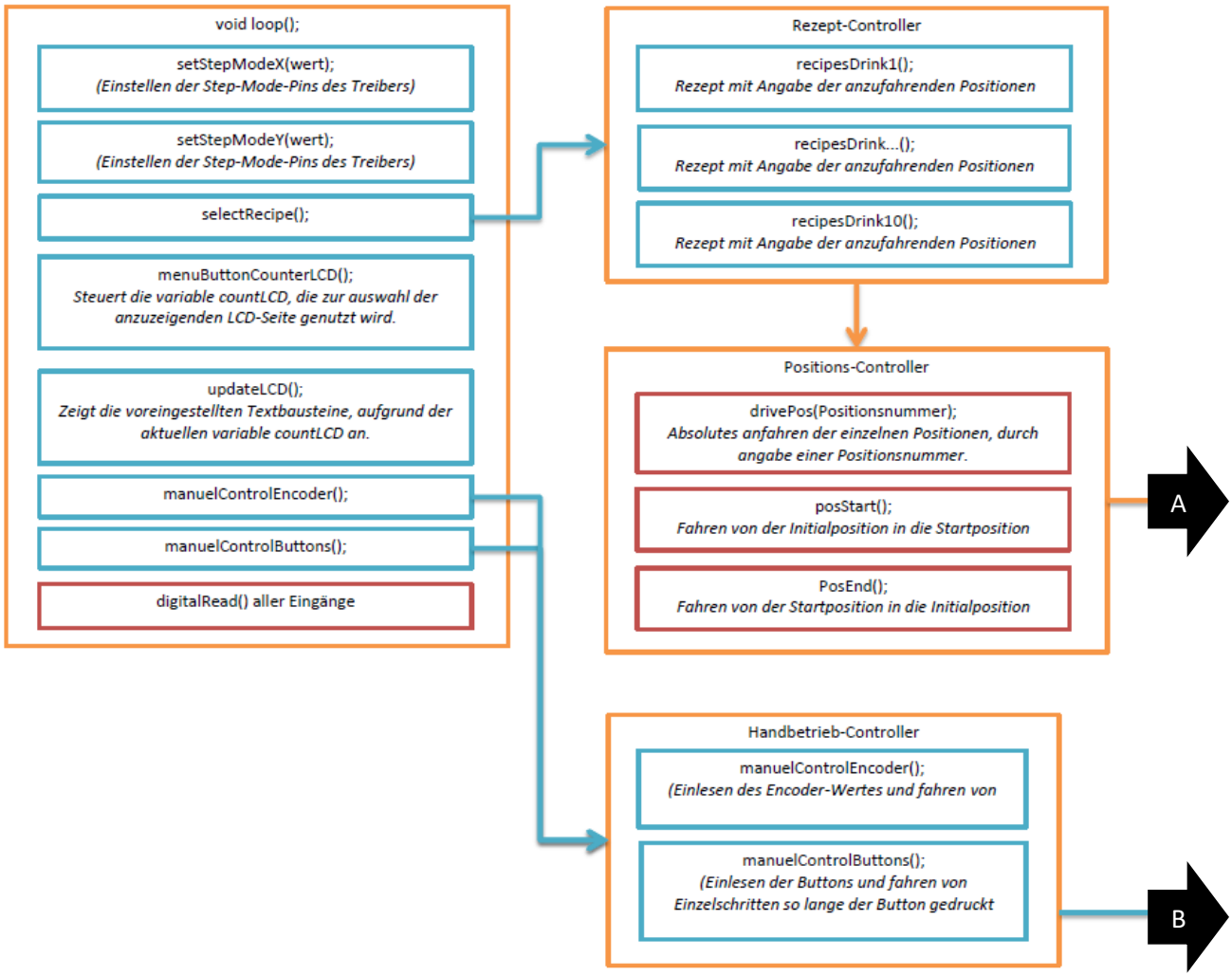

Abb. 26: Programmstruktur (Teil 1 von 2)

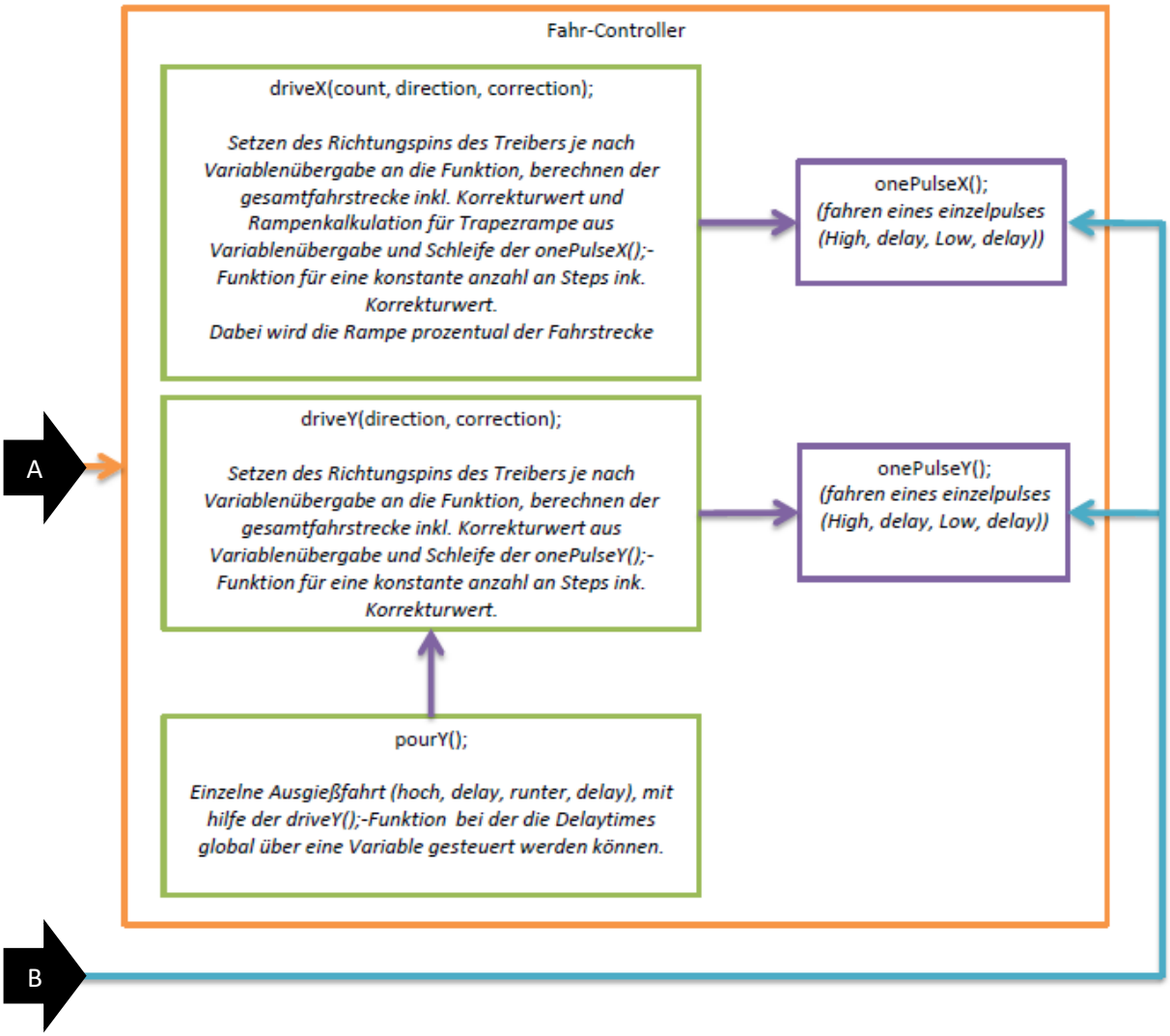

Abb. 27: Programmstruktur (Teil 1 von 2)

3.7.3 Programmcode

Hier der vollständige Programmcode, samt Erklärung.

3.7.3.1 Konstanten und Variablen-Definition und Deklaration

Vor dem Haupt-loop und vor dem Setup-Teil werden zunächst alle benötigten Variablen und Konstanten definiert und deklariert.

```
#include <LiquidCrystal.h>  //LCD-Library
LiquidCrystal theDisplay(34,36,38,40,42,44); //define LCD-Pins

//Constants
  // drive-Pins
    const int stepPinX = 26;  //pin for Stepping X-Axis
    const int dirPinX = 27;   //pin for Direction X-Axis
    const int stepPinY = 24;  //pin for Stepping Y-Axis
    const int dirPinY = 25;   //pin for Direction Y-Axis
  //drive-Mode-Pins
    const int m1PinX = 28;  //Drive-Mode Pin M1 X-Axis
    const int m2PinX = 30;  //Drive-Mode Pin M2 X-Axis
    const int m3PinX = 32;  //Drive-Mode Pin M3 X-Axis
    const int m1PinY = 29;  //Drive-Mode Pin M1 Y-Axis
    const int m2PinY = 31;  //Drive-Mode Pin M2 Y-Axis
    const int m3PinY = 33;  //Drive-Mode Pin M3 Y-Axis
  //Menu Button-Pins
    const int btn1Pin = 23;  //pin for button 1
    const int btn2Pin = 22;  //pin for button 2
  //Manuel-Mode-Pins
    const int btnYMPin = 47;  //pin for button Y-
    const int btnYPPin = 43;  //pin for button Y+
    const int btnXMPin = 45;  //pin for button X-
    const int btnXPPin = 41;  //pin for button X+
    const int encPin1 = 37;   //encoder Pin 1
    const int encPin2 = 35;   //encoder Pin 2
    const int encModePin = 39;//encoder Push Mode
  //debug-led
    const int debugLED = 13;  //pin for onboard LED

  //Value definitions
    const int textDelayFinish = 4000;     //Displaytime for Finish-Text
    const int timeDelayPour = 1800;       //Delaytime to Pour Drink
    const int timeDelaySwitchAxis = 300;  //Delaytime for change Axis
    const long speedPulseMinX = 5000;     //Delaytime in µs for minimum Speed (=
max delay) [5.000µs = 5 ms]
    const long speedPulseMaxX = 2000;     //Delaytime in µs for maximum Speed (=min
delay) [2.000µs = 2 ms]
    const int pulseY = 9000;              // number of pulses for full Y (9000)
    const int rampPercentX = 20;          //how many percent of the full move to
dest. is one ramp (acc or dec) [rampPercent * 2 + drivePercent = 100%]
    const int drivePercentX = 60;         //how many percent of the full move to
dest. is normal drive at full speed
    const int posPulseX = 335;            //number of pulses between two X-
positions (one full Turn = 200 Pulses; 335 @ 1/2-Step-Mode)
    const long speedPulseXManuel = 1000;
    const long speedPulseYManuel = 1000;
```

```
//Variables (values will be overwritten during program)
  //dynamic working-values
    long speedPulseX = 0;       //Delaytime in microsec (µs) for the steps
    long speedPulseY = 200;     //Delaytime in microsec (µs) [init = 200]
    int menuCounter = 0;        //variable for Menu Counter
    int busyOperate = 0;        //variable which is set 1 if the maschine is work-
ing on a Drink
    int pulseYcorr = 0;         //number of Pulses for Y-Drive with correction-
Value

  //init values for steppers
    int stepModeX = 0;          //step-mode init-value
    int stepModeY = 0;          //step-mode init-value
  //calculated Values for driving-curve
    long destinationPulseX = 0; //number of pulses for full travel to destination
    long speedPulseDeltaX = 0;  //calculated difference between min and max Pulse
    long speedPulsePerStep = 0; //calculated delaytime per Step
    long rampPulseX = 0;        //how many pulses for one ramp equals
dest.Pulse*rampPercent/100
    long drivePulseX = 0;       //how many pulses for normal drive equals
dest.Pulse * drivePercent/100
  //values for position-controller
    int actPos = 0;             //counting variable for actual position
    int tarPos = 0;             //target position variable
    int drvPos = 0;             //relative pos. calculated from actPos and tarPos

  //init values for Buttons etc.
    int btn1State = 0;            //variable for Button 1 actual state
    long btn1holdMil;             //millis btn1 is pressed
    long btn1holdSec;             //Seconds btn1 is pressed (calculated from mil-
lis)
    long btn1holdSecPrev;         //Seconds btn1 was pressed in prev. check
    byte btn1prevCheck;           //Switch to remember if btn was pressed in prev.
check
    unsigned long btn1firstTime;  //how long till btn1 was pressed first Time
    bool lcdDebugMode = false;    //Switch to remember if lcd is in debug-mode //
default: LOW = no debug mode

    int btn2State = 0;          //variable for Button 2
    int btn2StateOld = 0;       //variable for prev. Button-State
    int btnYMState = 0;         //Button Y-
    int btnYPState = 0;         //Button Y+
    int btnXMState = 0;         //Button X-
    int btnXPState = 0;         //Button X+
    int btnXYCounter = 0;       //variable for button manuel mode
  //encoder
    int encMode = 0;            //Encoder Mode (X-Axis or Y-Axis)
    int encIn = LOW;            //Encoder Input X&Y-Axis
    int encInOld = LOW;         //Encoder prev. Input X&Y-Axis
    int encValueX = 0;          //Encoder Value X-Axis
    int encValueY = 0;          //Encoder Value Y-Axis
```

3.7.3.2 Setup-Teil

Im Setup-Teil werden die Pins definiert und das LCD definiert.

```
void setup() {
  //Pin definitions
    //Drive-Pins
      pinMode(stepPinX,OUTPUT);
      pinMode(dirPinX,OUTPUT);
      pinMode(stepPinY,OUTPUT);
      pinMode(dirPinY,OUTPUT);
    //Drive-Mode-Pins
      pinMode(m1PinX, OUTPUT);
      pinMode(m2PinX, OUTPUT);
      pinMode(m3PinX, OUTPUT);
      pinMode(m1PinY, OUTPUT);
      pinMode(m2PinY, OUTPUT);
      pinMode(m3PinY, OUTPUT);
    //Menu-Buttons
      pinMode(btn1Pin,INPUT);
      pinMode(btn2Pin,INPUT);
    //Manuel-Mode-Pins
      pinMode(btnYMPin,INPUT);
      pinMode(btnYPPin,INPUT);
      pinMode(btnXMPin,INPUT);
      pinMode(btnXPPin,INPUT);
      pinMode(encPin1, INPUT_PULLUP);
      pinMode(encPin2, INPUT_PULLUP);
      pinMode(encModePin, INPUT);
    //debug-led
      pinMode(debugLED,OUTPUT); //Debug LED-Pin
    //LCD-init
      theDisplay.begin(16,2);
}
```

3.7.3.3 Haupt-Loop

Im Haupt-Loop werden nur die Unterfunktionen aufgerufen, die externen Bedienelemente eingelesen und deren Werte in Variablen geschrieben.

```
void loop() {
//Setting the drive-Parameter
    setStepModeX(1); //(1)  //0= fullstep;  1= 1/2-step;   2= 1/4-step;    3= 1/8-
step;    4=1/16-step;    5= 1/32-Step
    setStepModeY(1); //(1)
//Manuel-Control
    manuelControlEncoder();
    manuelControlButtons();
//Recipe Selection
    selectRecipe();
//LCD inhalt updaten
    menuButtonCounterLCD();
    updateLCD();
//Button Readings
    btn1State = digitalRead(btn1Pin);
    btn2State = digitalRead(btn2Pin);
    btnYMState = digitalRead(btnYMPin);
    btnYPState = digitalRead(btnYPPin);
    btnXMState = digitalRead(btnXMPin);
    btnXPState = digitalRead(btnXPPin);
//Encoder Readings
    encMode = digitalRead(encModePin);
    encIn = digitalRead(encPin1);
}
```

Erklärung

- *setStepModeX(1);* stellt den Treiber für den Schrittmotor der X-Achse auf 1/2-Step-Betrieb
- *setStepModeY(1);* stellt den Treiber für den Schrittmotor der Y-Achse auf 1/2-Step-Betrieb
- *manuelControlEncoder();* Ruft die Unterfunktion für den Encoder des Handbedienpults auf
- *manuelControlButtons();* Ruft die Unterfunktion für die Buttons des Handbedienpults auf
- *selectRecipe();* Ruft die Unterfunktion zum Starten von Rezeptabfahrten auf
- *menuButtonCounterLCD();* Ruft die Unterfunktion zum Steuern der variable auf, die den Inhalt des LCD festlegt
- *updateLCD();* Ruft die Unterfunktion auf, die den Inhalt des LCD anhand der zählvariable aus *menuButtonCounterLCD* anzeigt auf.
- *btn…State* liest den Status der Taster-Eingänge in entsprechende Variablen ein.
- *enc…* liest den Status der Encoder-Eingänge in entsprechende Variablen ein

3.7.3.4 Funktionen stepModeX() und stepModeY()

Diese Funktionen ermöglichen das komfortable, softwareseitige Umstellen der Step-Modes auf den Treiberplatinen.

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
// Setting the Step-Mode for Stepper X-Axis                           //
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //

void setStepModeX(int stepModeX){
  switch(stepModeX){
    case 1: // 1/2-Step-Mode
      digitalWrite(m1PinX, HIGH);
      digitalWrite(m2PinX, LOW);
      digitalWrite(m3PinX, LOW);
      break;
    case 2: //1/4-Step-Mode
      digitalWrite(m1PinX, LOW);
      digitalWrite(m2PinX, HIGH);
      digitalWrite(m3PinX, LOW);
      break;
    case 3: //1/8-Step-Mode
      digitalWrite(m1PinX, HIGH);
      digitalWrite(m2PinX, HIGH);
      digitalWrite(m3PinX, LOW);
      break;
    case 4: //1/16-Step-Mode
      digitalWrite(m1PinX, LOW);
      digitalWrite(m2PinX, LOW);
      digitalWrite(m3PinX, HIGH);
      break;
    case 5: //1/32-Step-Mode
      digitalWrite(m1PinX, HIGH);
      digitalWrite(m2PinX, HIGH);
      digitalWrite(m3PinX, HIGH);
      break;
    default:  //default (0) = fullstep-Mode
      digitalWrite(m1PinX, LOW);
      digitalWrite(m2PinX, LOW);
      digitalWrite(m3PinX, LOW);
      break;
  }
}
```

```
// Setting the Step-Mode for Stepper Y-Axis
void setStepModeY(int stepModeY){
  switch(stepModeY){
    case 1: // 1/2-Step-Mode
      digitalWrite(m1PinY, HIGH);
      digitalWrite(m2PinY, LOW);
      digitalWrite(m3PinY, LOW);
      break;
    case 2: // 1/4-Step-Mode
      digitalWrite(m1PinY, LOW);
      digitalWrite(m2PinY, HIGH);
      digitalWrite(m3PinY, LOW);
      break;
    case 3: // 1/8-Step-Mode
      digitalWrite(m1PinY, HIGH);
      digitalWrite(m2PinY, HIGH);
      digitalWrite(m3PinY, LOW);
      break;
    case 4: // 1/16-Step-Mode
      digitalWrite(m1PinY, LOW);
      digitalWrite(m2PinY, LOW);
      digitalWrite(m3PinY, HIGH);
      break;
    case 5: // 1/32-Step-Mode
      digitalWrite(m1PinY, HIGH);
      digitalWrite(m2PinY, HIGH);
      digitalWrite(m3PinY, HIGH);
      break;
    default:  //default (0) = fullstep-Mode
      digitalWrite(m1PinY, LOW);
      digitalWrite(m2PinY, LOW);
      digitalWrite(m3PinY, LOW);
      break;
  }
}
```

Erklärung

- Durch eine Parameterübergabe an die Funktion, wird per Switch/Case der entsprechende Step-Mode ausgewählt und die Step-Mode-Pins, die auf die Treiberplatine verdrahtet sind, entsprechend einem vordefinierten Bitmuster geschaltet.

3.7.3.5 Funktionen manuelControlEncoder() und manuelControlButtons()

Diese Funktionen dienen der Ansteuerung der Schrittmotoren durch das Handbedienpult.

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
// Manuel-Mode-Controller                                         //
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //

void manuelControlButtons(){  //Button-Mode
  if(btnYMState == HIGH and btnYPState == LOW){
    digitalWrite(dirPinY, LOW); //down
    onePulseY();
    }
  if(btnYPState == HIGH and btnYMState == LOW){
    digitalWrite(dirPinY, HIGH); //up
    onePulseY();
    }
  if(btnXMState == HIGH and btnYPState == LOW){
    digitalWrite(dirPinX, HIGH);  //right
    onePulseX();
    }
  if(btnXPState == HIGH and btnXMState == LOW){
    digitalWrite(dirPinX, LOW); //left
    onePulseX();
    }
}
```

Erklärung

- Es wird geprüft, welcher Button auf dem Handbedienpult gedrückt wurde und daraufhin der Richtungs-Pin der Treiberplatine entsprechend geschaltet und danach die entsprechende *onePulse*-Funktion so lange ausgeführt, wie der Button gedruckt gehalten wird
- Die Richtungen der Buttons sind so gegeneinander verriegelt, das ein gleichzeitiges ansteuern in beide Richtungen nicht funktioniert

```
void manuelControlEncoder(){  //Encoder-Mode
  if (encMode == HIGH){ //Encoder Down = X-Axis
    if ((encIn == HIGH)&&(encInOld == LOW)){
      if (digitalRead(encPin2) == HIGH){
        encValueX++;
        digitalWrite(dirPinX,LOW);
        onePulseX();

      }else{
        encValueX--;
        digitalWrite(dirPinX,HIGH);
        onePulseX();
      }
    }
```

Erklärung

- Zunächst wird geprüft, ob der sich Encoder in eingedruckter oder ausgefahrener Stellung befindet. Bei eingefahrener Stellung, gilt der oben stehende Code für die X-Achse, in ausgefahrener Stellung gilt der nachfolgende Code für die Y-Achse.
- Danach wird geprüft, in welche Richtung der Encoder gedreht wurde, in dem der Wert des letzten Durchlaufs mit dem aktuellen Wert verglichen wird.
- Entsprechend der Drehrichtung, wird dann der Richtungs-Pin der Motor-Treiber geschaltet
- Daraufhin wird die Funktion für einen Puls zu fahren aufgerufen
- Demnach fährt der Motor je Encoder-Puls auch einen Step

```
}else{    //Encoder Up = Y-Axis
    if ((encIn == HIGH)&&(encInOld == LOW)){
      if (digitalRead(encPin2) == HIGH){
        encValueY++;
        digitalWrite(dirPinY,LOW);
            onePulseY();  //do 8 Pulses (could also be written as a for-loop)
            onePulseY();
            onePulseY();
            onePulseY();
            onePulseY();
            onePulseY();
            onePulseY();
            onePulseY();
      }else{
        encValueY--;
        digitalWrite(dirPinY,HIGH);
            onePulseY();  //do 8 Pulses (could also be written as a for-loop)
            onePulseY();
            onePulseY();
            onePulseY();
            onePulseY();
            onePulseY();
            onePulseY();
            onePulseY();
      }
    }
  }
  encInOld = encIn;
}
```

Erklärung

- Zunächst wird wieder geprüft, ob der sich Encoder in eingedruckter oder ausgefahrener Stellung befindet. Bei eingefahrener Stellung, gilt der oben stehende Code für die X-Achse, in ausgefahrener Stellung gilt dieser Code für die Y-Achse.
- Danach wird geprüft, in welche Richtung der Encoder gedreht wurde, in dem der Wert des letzten Durchlaufs mit dem aktuellen Wert verglichen wird.
- Entsprechend der Drehrichtung, wird dann der Richtungs-Pin der Motor-Treiber geschaltet
- Daraufhin wird die Funktion für einen Puls zu fahren 8x aufgerufen, da die Hubfunktion durch die Spindeln eine deutlich höhere Übersetzung besitzt
    - Das mehrfach-Aufrufen der Einzelpuls-Funktion könnte auch über eine Schleife erfolgen.

3.7.3.6 *Einzelpulsfunktionen onePulseX() und onePulseY()*

Über die Einzelpulsfunktionen werden die Schrittmotoren im eigentlichen Sinne über die Treiberplatinen angesteuert.

```
// Simple Function for one Pulse X-Axis
void onePulseX(){
    digitalWrite(stepPinX,HIGH);
    delayMicroseconds(speedPulseX);
    digitalWrite(stepPinX,LOW);
    delayMicroseconds(speedPulseX);
}
// Simple Function for one Pulse Y-Axis
void onePulseY(){
    digitalWrite(stepPinY,HIGH);
    delayMicroseconds(speedPulseY);
    digitalWrite(stepPinY,LOW);
    delayMicroseconds(speedPulseY);
}
```

Erklärung

- Die beiden onePulse-Funktionen für die X- und Y-Achse sind identisch aufgebaut
    1. Step-Pin auf HIGH
    2. Delaytime warten
    3. Step-Pin auf LOW
    4. Delaytime warten
- Es gibt je Achse eine globale Variable, die den Delay und damit die Geschwindigkeit der Schrittmotoren steuert

3.7.3.7 Funktion selectRecipe()

Über die Funktion *selectRecipe()*wird je nach aktuell auf dem LCD angezeigtem Rezept, das Unterprogramm zum Abfahren des Entsprechenden Rezepts gestartet.

```
// Start Recipe if selected and button select pressed
//Bottle-Config:  A = Orangejuice; B = Peach-liquer; C = Barcardi; D = Tequila; E = Wodka; F = Blue Curacao
void selectRecipe(){
  if (btn2State != btn2StateOld){
    if (btn2State == HIGH){
      if (busyOperate != 1){
        switch(menuCounter){
            case 1:
              recipesDrink1();  //Screwdriver (1x Wodka[E]; 6x Orangejuice[A])
              break;
            case 2:
              recipesDrink2();  //Fuzzy Novel (1x Wodka[E]; 5x Orangejuice[A]; 1x Peach-liquer[B])
              break;
            case 3:
              recipesDrink3();  //Green Eyes (1x Wodka[E]; 5x Orangejuice[A]; 1x BlueCuracao[F])
              break;
            case 4:
              recipesDrink4();  //Green Hurrican (1x Wodka[E]; 1x Barcardi[C]; 4x Orangejuice[A]; 1x BlueCuracao[F])
              break;
            case 5:
              recipesDrink5();  //Green Sombrero (klein) (1x Peach-liquer[B] 2x Tequila[D]; 1x BlueCuracao[F])
              break;
            case 6:
              recipesDrink6();  //Grüne Wiese (2x BlueCuracao[F]; 5x Orangejuice[A])
              break;
            case 7:
              recipesDrink7();  //Grüne Wiese Plus (1x BlueCuracao[F]; 5x Orangejuice[A]; 1x Barcardi[C])
              break;
            case 8:
              recipesDrink8();  //Peach-O (5x Orangejuice[A]; 2x Peach-liquer[B])
              break;
            case 9:
              recipesDrink9();  //Tequila Sunset (1x Tequila[D]; 5x Orangejuice[A]; 1x BlueCuracao[F])
              break;
            case 10:
              recipesDrink10(); //Barcardi Sunset (1x Barcardi[E]; 5x Orangejuice[A]; 1x BlueCuracao[F])
              break;
```

```
            case 92:
               recipesDebugYM(); //Testprogram for Y-Axis Down
               break;
             case 93:
               recipesDebugYP(); //Testprogram for Y-Axis Up
               break;
             case 94:
               recipesDebugXM(); //Testprogram for X-Axis right
               break;
             case 95:
               recipesDebugXP(); //Testprogram for X-Axis left
               break;
             default:
               break;
         }
       }
     }
   }
   btn2StateOld = btn2State;
}
```

Erklärung

- Zuerst wird eine doppelte Auslösung des Tasters durch eine Verriegelung verhindert
- Dann wird geschaut, ob die Maschine gerade ein Rezept abarbeitet und ein Restart während einer aktuellen Abarbeitung verhindert
- Abschließend wird über die laufvariable des LCD-Menüs per Switch-Case die zu fahrende Aktion aufgerufen
- Neben den eigentlichen Rezepten sind hier auch die Einzelplatz-Fahrprogramme hinterlegt

3.7.3.8 Funktion recipeDrink_n()

Die Funktion *recipesDrink_n()* stellt exemplarisch das Rezept eines Cocktails dar. Die einzelnen Rezepte sind prinzipiell identisch aufgebaut und durchnummeriert. Z.B. *recipeDrink4()* für das 4. Cocktailrezept.

```
void recipesDrink4(){
  busyOperate = 1;
  textWaitLCD();
  //Green Hurrican (1x Wodka[E]; 1x Barcardi[C]; 4x Orangejuice[A]; 1x BlueCura-
cao[F])Order:C-E-F-A-A-A-A
      posStart();
      drivePos(3);
      drivePos(5);
      drivePos(6);
      drivePos(1);
      drivePos(1);
      drivePos(1);
      drivePos(1);
      drivePos(0);
      posEnd();
  busyOperate = 0;
  textFinishLCD();
  menuCounter = 0;
}
```

Erklärung

- Zuerst wird über die Variable *busyOperate* eine Restart-Verriegelung gesetzt.
- Danach die Glas-Plattform in Startposition gesenkt
- Danach über die *drivePos(n)* absolut unter die entsprechenden Plätze verfahren und ausgehoben
    - Findet keine Platz-Änderung statt, wird an gleicher Stelle erneut ausgehoben.
- Schlussendlich wird die Plattform wieder in die Initialposition angehoben
- Abschließend wird noch die Restart-Verriegelung zurückgenommen, auf dem LCD der „Drink fertig“-Text angezeigt und danach das LCD-Menü wieder in den Begrüßungsbildschirm versetzt

3.7.3.9 Funktionen drivePos(n), posStart() und posEnd()

Diese Funktionen sind Bestandteil des Positioniercontrollers, der ein absolutes Anfahren der einzelnen Positionen erlaubt.

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
// Position Controller (Translates Positions to Driving-Commands)   //
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //

void drivePos(int posNo){

  if (posNo <= 6 and posNo >= 0){
    tarPos = posNo;
    drvPos = tarPos - actPos;
    if(drvPos == 0){
      delay(timeDelaySwitchAxis);
      if(tarPos != 0){
        pourY();
      }
    }
    else if(drvPos < 0){
      actPos = actPos + drvPos;
      drvPos = drvPos * -1;
      driveX(drvPos,0,0);
      delay(timeDelaySwitchAxis);
      if(tarPos != 0){
        pourY();
      }
    }
    else if(drvPos > 0){
      actPos = actPos + drvPos;
      driveX(drvPos,1,0);
      delay(timeDelaySwitchAxis);
      if(tarPos != 0){
        pourY();
      }
    }
  }else{
    textFailLCD(); //show error-message
    delay(5000);   //for 5s
  }
}
```

Erklärung

- Die Funktion *posStart()* verfährt die Plattform von der Initialposition herunter in die Startposition
- Die Funktion *posEnd()* verfährt die Plattform von der Startposition hoch in die Initialposition
- Die Funktion *drivePos(n)* fährt die Plattform unter die Position, die der Funktion übergeben wurde
    - Dabei wird zunächst geprüft ob die Eingabe eine gültige Positionsnummer ist
    - Dann geschaut, ob die Positionsnummer links, rechts oder am aktuellen Standort liegt
    - Abschließend wird die Plattform entsprechend verfahren und die neue aktuelle Position gemerkt

3.7.3.10 Funktion driveX()

Die Funktion *driveX()* kann die Plattform relativ zur aktuellen Position verfahren. Dabei werden der funktion die nachfolgenden Werte übergeben:

- Anzahl zu fahrende Positionen
- Richtung
- Korrekturwert

Dabei wird die Plattform immer mittels Trapezrampe verfahren!

```
// Function for driveX
void driveX(int countX, int dir, int corrPulseX){
  if(dir == 0) {digitalWrite(dirPinX,LOW);};  //rotations direction (low = left;
high = right)
  if(dir == 1) {digitalWrite(dirPinX,HIGH);};

  //Calculate Position
  destinationPulseX = countX * posPulseX + corrPulseX;     //calculate number of
pulses for full move to destination
  drivePulseX = destinationPulseX * drivePercentX / 100;   //calculate number of
pulses for drive at full speed
  rampPulseX = destinationPulseX * rampPercentX / 100;     //calculate number of
pulses for one ramp (used for both ramps)

  //Calculate Speed
  speedPulseDeltaX = speedPulseMinX - speedPulseMaxX;     //calculate delayTime
delta
  speedPulsePerStep = speedPulseDeltaX / rampPulseX;       //how many delaytime per
Step in the Ramp

  //drive the Stepper with acc-Ramp, const drive and dec-Ramp
  speedPulseX = speedPulseMinX;           //init with min Speed
  for(int x1 = 0; x1 < rampPulseX; x1++){                //acceleration ramp (for
the number of Pulses: decrease the delayTime each loop)
    speedPulseX = speedPulseX - speedPulsePerStep;
    onePulseX();
  }
  speedPulseX = speedPulseMaxX;         //set to max speed (should be after
acc.Ramp)
  for(int x2 = 0; x2 < drivePulseX; x2++) {        //drive with max speed
    onePulseX();
  }
  for(int x3 = 0; x3 < rampPulseX; x3++){                //deceleration ramp (for
the number of Pulses: increase the delayTime each loop)
    speedPulseX = speedPulseX + speedPulsePerStep;
    onePulseX();
  }
  speedPulseX = speedPulseMinX;           //reset to min Speed
}
```

Erklärung

- Zunächst wird die Richtung aus der Variablenübergabe ausgewertet und der Richtungspin entsprechend geschaltet
- Danach werden die Positionsberechnungen durchgeführt
  - Berechnung der Anzahl Pulse bis zur Zielposition
  - Berechnung der Anzahl Pulse der Konstantfahrt
  - Berechnung der Anzahl Pulse der Rampen
- Dann die Geschwindigkeitsberechnungen durchgeführt
  - Berechnung der zu bewältigenden Geschwindigkeitsdifferenz
  - Berechnung der Geschwindigkeitsdifferenz je Puls während der Rampenfahrt

- Abschließend die Geschwindigkeit des Steppers dynamisch je nach Fahrabschnitt angepasst
  - Beschleunigungsrampe
  - Konstantfahrt
  - Verzögerungsrampe

3.7.3.11 Funktion driveY()

Diese kann die Plattform relativ zur aktuellen Position anheben oder absenken. Dabei werden der funktion die nachfolgenden Werte übergeben:

- Richtung
- Korrekturwert

```
// Function for driveY
void driveY(int dir, int corrPulseY){
  if(dir == 0) {digitalWrite(dirPinY,HIGH);};        //rotations direction (High =
Up; low = down)
  if(dir == 1) {digitalWrite(dirPinY,LOW);};
  pulseYcorr = pulseY + corrPulseY;
  for(int x = 0; x < pulseYcorr; x++) {
    onePulseY();
  }
}
```

Erklärung

- Zunächst wird die Richtung aus der Variablenübergabe ausgewertet und der Richtungspin entsprechend geschaltet
- Danach eine vollständige Y-Fahrt durchgeführt
  - Die Anzahl Pulse ist in einer globalen Variable hinterlegt

3.7.3.12 Funktion pourY ()

Die Funktion *pourY()* fährt bei Aufruf (ohne anzugebende Variablen) einen vollständigen Ausgieß-Zyklus der Plattform und vereinfacht damit den Umgang mit der *driveY()*-Funktion.

```
void pourY(){ //1x Up/pour/down
  driveY(1,0); //up
  delay(timeDelayPour);
  driveY(0,0); //down
  delay(timeDelaySwitchAxis);
}
```

Erklärung

- Es wird die Funktion *driveY()* mit entsprechender Richtungsangabe zum Ausheben aufgerufen
- Danach wird eine global festgelegte Zeit zum Ausgießen gewartet
- Abschließend die Plattform per *driveY()* wieder gesenkt

3.7.3.13 Funktion menuButtonCounterLCD()

Diese Funktion variiert die Laufvariable zur Steuerung des LCD-Inhalts. Dabei wird zwischen „kurzem“ Drücken des „Select“-Tasters und „langem“ Drücken des „Select“-Tasters unterschieden.

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
// LCD-Controller                                                     //
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //

//Menu-Controlling
void menuButtonCounterLCD(){
//each Button Press increases the counter for menu. Last one resets to top.
  if (btn1State == HIGH && btn1prevCheck == LOW && (millis() - btn1firstTime) >
200){           //check if btn1 pressed
    btn1firstTime = millis();
//remember the timestamp of btn-pressed
  }
  btn1holdMil = (millis() - btn1firstTime);
//count the time the btn1 is pressed
  btn1holdSec = btn1holdMil / 1000;
//calculate the millis to seconds

  if(btn1holdMil > 50){
//btn1 has to be pressed at least 50ms
    if(btn1State == LOW && btn1prevCheck == HIGH){
//check for btn1 release
      if(btn1holdSec < 2){
//if btn1 was held for less than 2 sec
         if (lcdDebugMode == false && menuCounter < 10){menuCounter++;}
            else if(lcdDebugMode == false && menuCounter >= 10){menuCounter = 0;};
//increase the menuCounter each press till last page, then back to 1
         if (lcdDebugMode == true && menuCounter > 10 && menuCounter <
95){theDisplay.clear(); menuCounter++;}
            else if(menuCounter >= 94 && lcdDebugMode == true){menuCounter = 90;
theDisplay.clear();};
      }
      if(btn1holdSec > 3) {
//if btn1 was held for more than 3 sec
        if(menuCounter <= 10){
//if the menuCounter is in the normal range
          lcdDebugMode = true;
          menuCounter = 90;
//set menuCounter to debug-screen
          theDisplay.clear();
        }else{
//if the menuCounter is out of the normal range (for example debug-screen active)
          lcdDebugMode = false;
          menuCounter = 0;
//go back to welcome-screen
        }
      }
    }
  }
  btn1prevCheck = btn1State;
//remember the btn1 state for prevent retrigger
  btn1holdSecPrev = btn1holdSec;
}
```

Erklärung

- Durch langes drücken des Tasters wird das LCD in den Debug-Modus versetzt und gibt versteckte Service-Funktionen frei
    - Wird der Taster nur kurz gedrückt, wird der Wert so lange um 1 hochgezählt, bis die letzte Seite erreicht wurde und danach wieder auf 0 gesetzt
    - Wird der Taster lange gedrückt, springt der Wert auf 90 und zählt ab dann bei jedem kurzen Drücken wieder um 1 hoch, bis die letzte Seite des Debug-Screens erreicht wurde, danach wird wieder auf 90 zurückgesetzt
    - Erneutes langes Drücken des Tasters setzt den Wert wieder auf 0 und verlässt den Debug-Modus

3.7.3.14 Funktion updateLCD()

Über die Funktion *updateLCD()* werden die Textbausteine des LCD organisiert und aufgerufen, je nachdem welchen Wert der menuCounter besitzt. Wird der menuCounter zurückgesetzt, so wird das LCD auf den Begrüßungstext umgeschaltet.

```
// LCD update on button press
void updateLCD(){
  switch(menuCounter){
      case 1:
        textDrink1LCD();
        break;
      case 2:
        textDrink2LCD();
        break;
      case 3:
        textDrink3LCD();
        break;
      case 4:
        textDrink4LCD();
        break;
      case 5:
        textDrink5LCD();
        break;
      case 6:
        textDrink6LCD();
        break;
      case 7:
        textDrink7LCD();
        break;
      case 8:
        textDrink8LCD();
        break;
      case 9:
        textDrink9LCD();
        break;
      case 10:
        textDrink10LCD();
        break;
```

```
      case 90:
        textDebug1();
        break;
      case 91:
        textDebug2();
        break;
      case 92:
        textTestYMLCD();  //drive down
        break;
      case 93:
        textTestYPLCD();  //drive up
        break;
      case 94:
        textTestXMLCD();  //drive right
        break;
      case 95:
        textTestXPLCD();  //drive left
        break;
      default:
        textWelcomeLCD();
        break;
    }
}
```

3.7.3.15 Funktionen text…()

Das Anzeigen von Texten auf dem LCD geschieht immer nach dem gleichen Muster. Das LCD besitzt 2x16 Zeichen. Zum Schreiben von Texten wird zunächst der Cursor an die gewünschte Startposition gesetzt und danach über den Befehl LCD-Print der gewünschte Text ausgegeben.

Dazu hier exemplarisch drei verschiedene Text-Funktionen:

```
void textWelcomeLCD(){    //Welcome-Message
  theDisplay.setCursor(0,0);
  theDisplay.print("Cocktailmaschine");
  theDisplay.setCursor(0,1);
  theDisplay.print("by deKode& DeKai");
}
void textWaitLCD(){    //Wait-Message
  theDisplay.setCursor(0,0);
  theDisplay.print("Drink in Arbeit!");
  theDisplay.setCursor(0,1);
  theDisplay.print("Bitte Warten...!");
}
void textFinishLCD(){    //Finish-Message
  theDisplay.setCursor(0,0);
  theDisplay.print("Drink fertig!   ");
  theDisplay.setCursor(0,1);
  theDisplay.print("Bitte entnehmen!");
  delay(textDelayFinish); //show Message for specific time
}
```

3.7.4 Rezept-Auswahl

Zur Rezeptplanung wurde die IOS-App „Cocktails“ verwendet, in der man Zutaten definieren kann und die App alle mit diesen Zutaten herstellbaren Cocktailrezepte ausgibt.

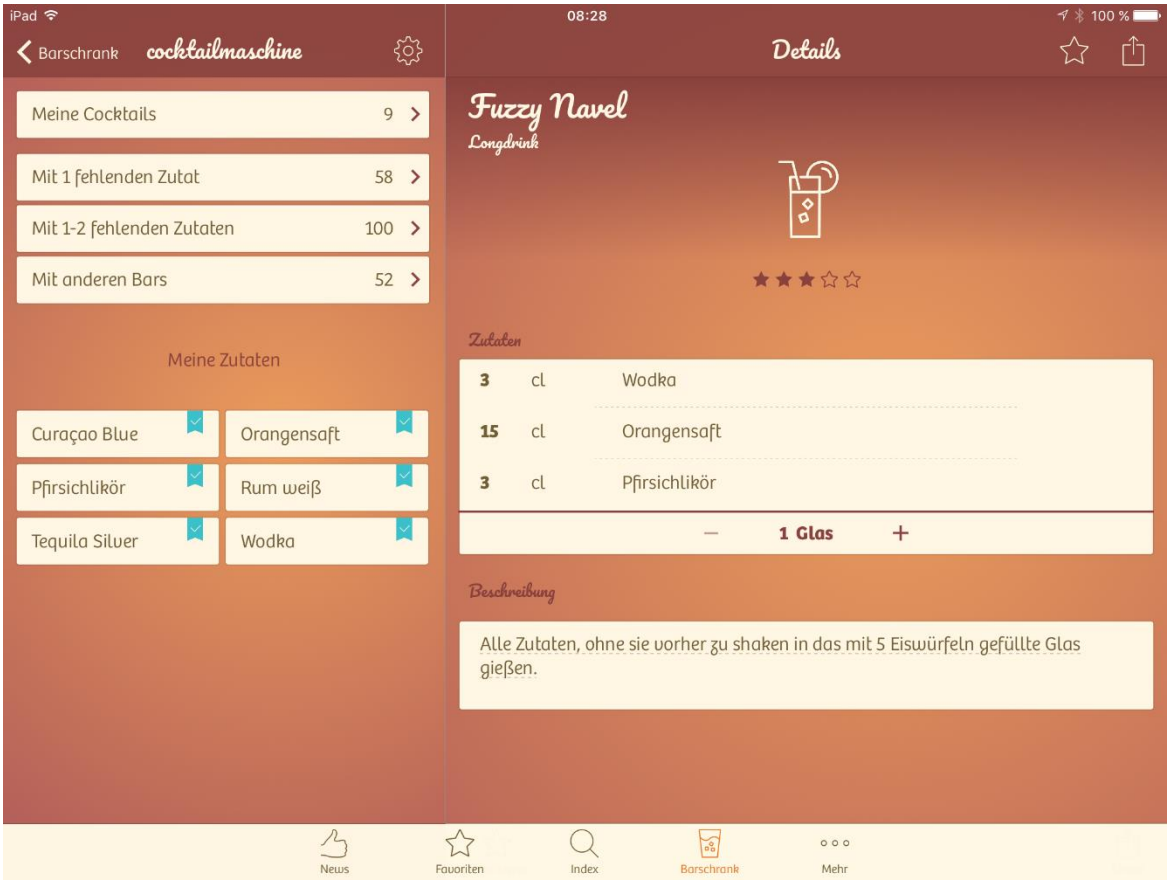

Abb. 28: Cocktails-App

3.7.4.1 Zutatenauswahl

Nach einiger Recherche mit der oben genannten App wurde sich für die Erstbestückung auf die Folgenden Zutaten geeinigt:

Flaschenplatz	**Zutat**
Platz 1	Orangensaft
Platz 2	Limonensaft
Platz 3	Vodka
Platz 4	Weißer Rum (Barcardi)
Platz 5	Tequila Silver
Platz 6	Blue Curacao

Tab. 1: Zutaten

Die Zutaten und die daraus resultierenden Rezepte können selbstverständlich später noch angepasst werden. Für dieses Schulprojekt jedoch, werden die oben definierten Zutaten verwendet.

3.7.4.2 Cocktail-Rezept 1 (Screwdriver)

Zutaten:

- 1x Wodka [Platz E]
- 6x Orangensaft [Platz A]

Abarbeitungsreihenfolge:

E – A – A – A – A – A – A – A

3.7.4.3 Cocktail-Rezept 2 (Fuzzy Novel)

Zutaten:

- 1x Wodka [Platz E]
- 5x Orangendsaft [Platz A]
- 1x Pfirsichlikör [Platz B]

Abarbeitungsreihenfolge:

B – E – A – A – A – A – A

3.7.4.4 Cocktail-Rezept 3 (Green Eyes)

Zutaten:

- 1x Wodka [Platz E]
- 5x Orangensaft [Platz A]
- 1x BlueCuracao [Platz F]

Abarbeitungsreihenfolge:

E – F – A – A – A – A – A

3.7.4.5 Cocktail-Rezept 4 (Green Hurrican)

Zutaten:

- 1x Wodka [Platz E]
- 1x Barcardi [Platz C]
- 4x Orangensaft [Platz A]
- 1x BlueCuracao [Platz F]

Abarbeitungsreihenfolge:

C – E – F – A – A – A – A

3.7.4.6 Cocktail-Rezept 5 (Green Sombrero[Shot])

Zutaten:

- 1x Pfirsichlikör [Platz B]
- 2x Tequila[Platz D]
- 1x BlueCuracao [Platz F]

Abarbeitungsreihenfolge:

B – D – D – F

3.7.4.7 Cocktail-Rezept 6 (Grüne Wiese)

Zutaten:

- 2x BlueCuracao [Platz F]
- 5x Orangensaft [Platz A]

Abarbeitungsreihenfolge:

F – F – A – A – A – A – A

3.7.4.8 Cocktail-Rezept 7 (Grüne Wiese Plus)

Zutaten:

- 1x BlueCuracao [Platz F]
- 5x Orangensaft [Platz A]
- 1x Barcardi [Platz C]

Abarbeitungsreihenfolge:

F – C – A – A – A – A – A

3.7.4.9 Cocktail-Rezept 8 (Peach-O)

Zutaten:

- 2x Pfirsichlikör [Platz B]
- 5x Orangensaft [Platz A]

Abarbeitungsreihenfolge:

B – B – A – A – A – A – A

3.7.4.10 Cocktail-Rezept 9 (Tequila Sunset)

Zutaten:

- 1x Tequila [Platz D]
- 5x Orangensaft [Platz A]
- 1x BlueCuracao [Platz F]

Abarbeitungsreihenfolge:

D – F – A – A – A – A – A

3.7.4.11 Cocktail-Rezept 10 (Barcardi Sunset)

Zutaten:

- 1x Barcardi [Platz C]
- 5x Orangensaft [Platz A]
- 1x BlueCuracao [Platz F]

Abarbeitungsreihenfolge:

C – F – A – A – A – A – A

3.8 Zeitplanung und Ablauf

Das der zeitliche Ablauf in diesem Projekt zu großen Problemen führen kann, mussten auch wir feststellen, da viele Teile aus Fernost geordert wurden und mit entsprechenden Lieferzeiten beaufschlagt waren.

Demnach war eine frühe zeitliche Planung des Projekts unabdingbar, woraus sich folgende Abarbeitungsreihenfolge bis zur Bestellung ergab:

1. Konzept festlegen
2. Mechanik-Konstruktion
3. Stücklisten-Erstellung
4. Materialbestellung

Nachdem innerhalb der ersten drei Wochen nach Projekterteilung die Bestellungen abgeschlossen waren, konnten während der Lieferzeit schon weitere Punkte abgearbeitet werden:

5. Verfeinerung der Mechanik-Konstruktion
6. Elektro-Konstruktion
7. Überlegungen zum Programmablauf
8. Erstellen erster Test-Programme

Nachdem dann die ersten Elektronik-Komponenten eintrafen konnte mit ersten Tests begonnen werden:

9. Testaufbau der Schaltung mittels Breadboard
10. Testen und Optimieren von Programmabschnitten
11. Strukturierung der Dokumentation
12. Lösen erster Probleme

Als dann die Mechanik-Teile eintrafen, kümmerte sich der Einer um den Mechanik Aufbau und die weitere Materialdisposition und der Andere um die Software und Elektronik. So dass dann relativ zeitnah zueinander die letzten Schritte abgearbeitet werden konnten:

13. Fertigung der Elektronik-Komponenten (Platinen etc.)
14. Zusammenbau der Mechanik
15. Holzzuschnitt und Lackieren
16. Testen und Optimieren der Elektronik und Software

3.9 Probleme im Projektablauf

Während der Projektabarbeitung kam es natürlich auch zu diversen Problemen die nachfolgend, gemeinsam mit ihrer Lösung beschrieben werden.

3.9.1 Motortreiber zu schwach

Bei den ersten Tests auf den Breadboard-Aufbauten mussten wir feststellen, dass die zuerst eingeplanten Motortreiber A4988 zu schwach waren, um die gewünschte Leistung zu fahren. Daraufhin entschieden wir uns, die Treiber gegen die DRV8825 auszutauschen, was auch den gewünschten Erfolg brachte.

3.9.2 Schrittmotoren ruckeln bei der Rampenfahrt

Das nächste Problem ereignete sich beim Testen des Fahrprogramms für die Rampenfahrt. Während der Beschleunigungs- und Verzögerungsrampe stockte der Motor immer an der gleichen Stelle. Behoben werden konnte das Problem, indem wir von der Vollschritt-Betriebsart in die 1/2-Schritt-Betriebsart wechselten. Dadurch wurde der Schrittmotor zwar langsamer, was wir durch Anpassung der Pulspausenzeit kompensieren konnten, aber dafür deutlich sanfter und mit spürbar mehr Drehmoment.

3.9.3 Serial.Print bremst die Pulspausenzeiten

Zur Fehlersuche und Prüfung, ob die Rampen wie gewünscht funktionieren, ließen wir uns per seriellen Monitor die berechneten Puls-Pausenzeiten ausgeben, damit wir diese dann in Excel Importieren und als Kurve darstellen konnten.

Dabei stellten wir fest, dass die Zeiten alle korrekt berechnet wurden, die Motoren aber nicht die gewünschte Zeit fuhren.

Nach einiger Fehlersuche fanden wir heraus, dass die serielle Verbindung mit 9600 bps die gesamte Abarbeitungszeit des Arduino derart beeinflusste, dass die Motoren nicht mehr sauber liefen. Nachdem wir die vollständigen seriellen Funktionen aus dem Code auskommentierten, liefen die Motoren wie gewünscht.

3.9.4 Lieferprobleme Zahnräder

Trotz der frühen Bestellung, brauchten gerade die vier wichtigsten Zahnräder derart lange für die Lieferung, dass wir uns entschieden, die Bestellung zu stornieren und gegen Aufpreis bei einem Deutschen Händler dieselben Zahnräder zu ordern. Kurze Zeit darauf wurden die Zahnräder geliefert und wir konnten mit dem Zusammenbau beginnen.

3.9.5 Falsche Länge der Haupt-Zahnriemen für die Hubfunktion

Nachdem die Zahnräder nun angekommen waren, mussten wir feststellen, dass die ursprünglich berechnete Länge der Zahnräder nicht passte. Demnach mussten die Zahnriemen neu bestellt werden, was abermals eine Verzögerung des Zusammenbaus mit sich zog.

3.9.6 Fehlende Pull-Down-Widerstände ohne angestecktes Handbedienpult

Das Handbedienpult wurde so entworfen, dass die Pull-Down-Widerstände in dem mobilen Pult verbaut wurden. Das brachte bei der Inbetriebnahme das Problem mit sich, das bei nicht angestecktem Pult, die digitalen Eingänge einen nicht definierten zustand einnahmen, wodurch die Motoren anfingen wild hin und her zu zucken.

Abhilfe schaffte ein D-Sub Stecker, der anstelle des Pults an der Maschine angesteckt wird, wenn dieses nicht benötigt wird, welcher integrierte Pull-Down-Widerstände für die entsprechenden digitalen Pins bereitstelle.

4 Materialdisposition

Im Folgenden die Auflistung aller gekauften Komponenten und deren Einkaufspreise, abzgl. der Versandkosten. Befestigungsmaterialien wie Schrauben, Muttern und Kabelbinder werden nicht weiter detailliert aufgelistet.

Pos.	Beschreibung	Lieferant	Einzelpreis [€]	Anzahl [Stk/m]	Einzelpreis Summe [€]
Mechanik-Komponenten					
1-01	SBR16UU Linearführung	eBay	15,78 €	1x	15,78 €
1-02	Spindeln	eBay	6,07 €	4x	24,28 €
1-03	Befestigungswinkel	eBay	3,25 €	2x	6,50 €
1-04	Zahnriemen „lang“ 1360 mm	eBay	6,95 €	2x	13,90 €
1-05	Zahnriemen „kurz“ 320 mm	eBay	4,35 €	2x	8,70 €
1-06	Zahnriemen „Plattform“ 2 m	eBay	8,00 €	1x	8,00 €
1-07	GT2 Zahnriemenscheibe 60Z	eBay	3,99 €	2x	7,98 €
1-08	GT2 Zahnriemenscheibe 36Z	eBay	2,74 €	7x	19,18 €
1-09	8stk VPE 16TGT2 Zahnrad Pulley Räder	eBay	18,99 €	1x	18,99 €
1-10	Pulley 20T 5mm	eBay	2,25 €	1x	2,25 €
1-11	Zahnriemen Befestigung	eBay	3,84 €	1x	3,84 €
1-12	GT2 20T Doppelzahnrad	eBay	2,99 €	1x	2,99 €
1-13	Holzplatten 1200x800mm	Baumarkt	5,00 €	2x	10,00 €
1-14	Flaschenhalter mit Portionierern	Amazon	27,90 €	1x	27,90 €
				Zwischensumme Mechanik:	170,29 €
Elektronik-Komponenten					
2-01	NEMA 17 Schrittmotor	eBay	8,90 €	2x	17,80 €
2-02	A4988 Motortreiber	eBay	2,40 €	2x	4,80 €
2-03	Arduino Mega	eBay	10,64 €	1x	10,64 €
2-04	LCD Rahmen	eBay	6,50 €	1x	6,50 €
2-05	LC-Display	eBay	3,00 €	1x	3,00 €
2-06	DRV8825 Motortreiber	eBay	1,83 €	2x	3,66 €
2-07	D-Sub Buchse/Stecker	eBay	2,00 €	1x	2,00 €
2-08	D-Sub Kabel	eBay	3,46 €	1x	3,46 €
2-09	Platinen (Lochraster Prototyp)	eBay	0,47 €	5x	2,35 €
2-10	Lötmaterial (Widerstände, Kondensatoren, …)	eBay/Amazon	3,00 €	1x	3,00 €
2-11	Spannungswandler (12 V -> 5 V)	eBay	1,38 €	1x	1,38 €
2-12	Netzteil (12 V/5 A)	eBay	12,90 €	1x	12,90 €
				Zwischensumme Elektronik:	71,49 €
				Übertrag Mechanik:	170,29 €
				Gesamtsumme:	**241,78 €**

Tab. 2: Materialdisposition

5 Fazit

Die Cocktailmaschine funktioniert, aber bietet noch reichlich Platz für Erweiterungen.

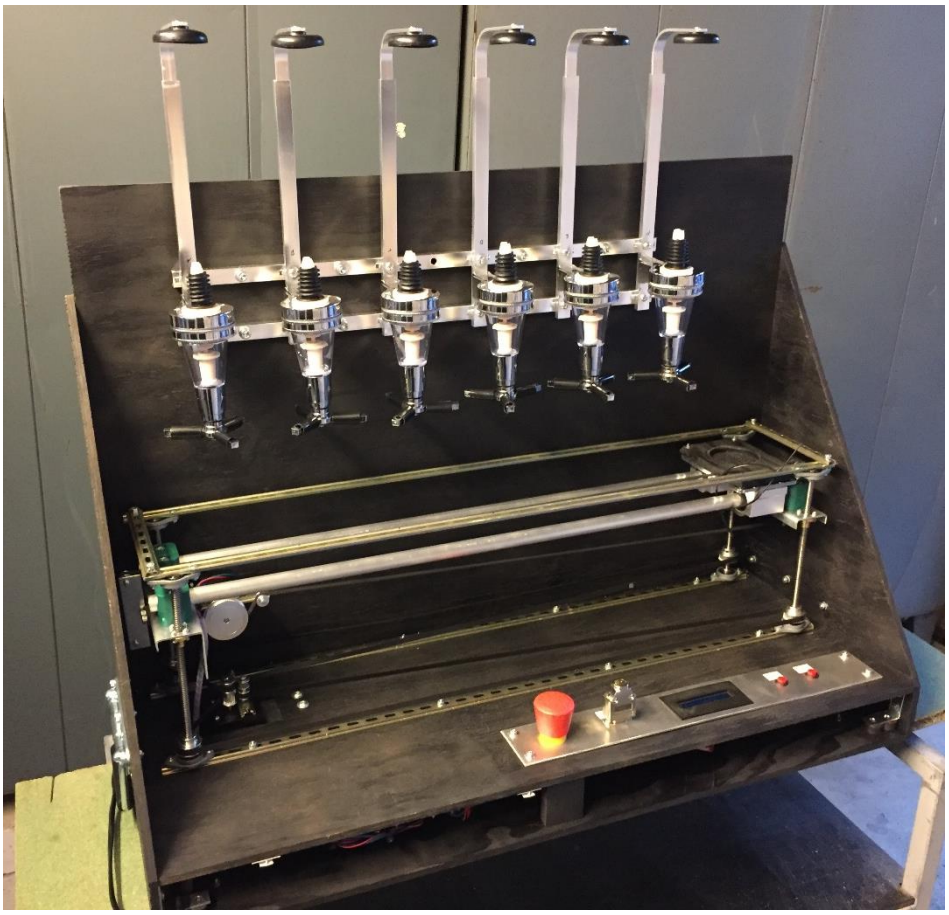

Abb. 29: Fertige Cocktailmaschine

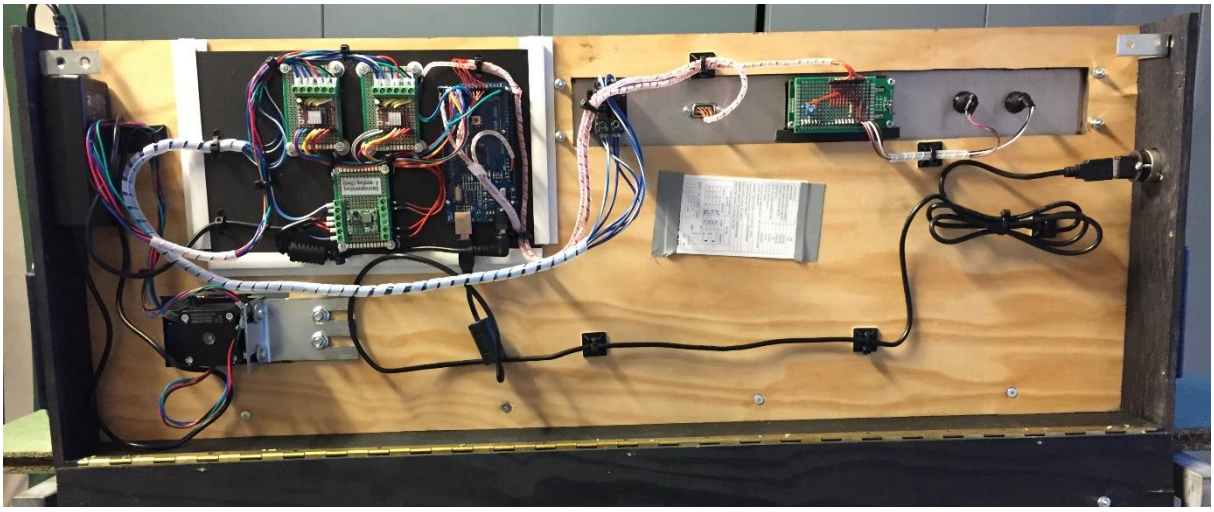

Abb. 30: Fertige Elektronik

Abschließend kann man durchweg sagen, dass das Projekt durchaus interessant war und auch viel Wissen bezüglich der Ansteuerung von Schrittmotoren vermitteln konnte.

Die Cocktailmaschine soll zukünftig für den privaten Einsatz auf Partys und Feierlichkeiten noch erweitert und optimiert werden. Dabei wurden bislang folgende Punkte als verbesserungswürdig und mögliches optimierungspotential herausgestellt:

- Überflüssig werden des Pull-Down-Steckers an der Cocktailmaschine
    - Ändern der Schaltung und Software auf die im Arduino integrierten Pull-Up-Widerstände
- LCD-Rahmen austauschen
    - Der LCD-Rahmen ist ein gekauftes 3D-Gedrucktes Teil, was aber in keiner Weise unseren Qualitätsanforderungen Rechnung trägt und sollte dringend gegen eine Spritzgeschützte Variante ausgetauscht werden.
- Spritzschutz über den Schrittmotoren
    - Über den Schrittmotoren wird noch ein Spritzschutz ergänzt, da die Portionierer u.u. nach dem Ausschenken noch „nachtropfen“ und damit die Elektronik beschädigen könnten.
- Endlagenschalter für automatische Referenzierung
    - Automatische Referenzfahrt beider Achsen durch anfahren von definierten Endlagen
- Ergänzung der Portionierer um Schlauchbasierte Pumpenmotoren für Säfte und Sirups
    - Durch die Erweiterung der Maschine ist es nicht mehr notwendig für die Saft-Ausgabe mehrfach den Portionierer anzufahren
- S-Kurvenfahrt für Fahrmotor X-Achse
    - Um die Fahrt der X-Achse mit vollem Glas noch weiter zu beruhigen und damit ein Überschwappen der Flüssigkeit noch weiter zu verhindern, soll die aktuelle Trapez-Kurve gegen eine S-Kurve mit Ruckreduzierung ausgetauscht werden. Das theoretische Modell dazu wurde bereits im Rahmen dieser Projektarbeit erarbeitet, jedoch die Implementierung aus Zeitgründen verworfen.
- Dynamische Rezeptverwaltung
    - Option A
        - SD-Karte am Arduino auf dem die Rezepte gespeichert werden
    - Option B
        - Windows-Software als „Fernsteuerung“ für die Cocktailmaschine mit Cocktail-Datenbank

Eidesstattliche Erklärung

Hiermit erklären wir an Eides Statt, dass wir diese Projektarbeit und Dokumentation selbständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel angefertigt haben. Alle den Quellen wörtlich oder sinngemäß entnommenen Stellen haben wir als solche gekennzeichnet.

Die Arbeit hat weder in dieser noch in ähnlicher Form als Prüfungsleistung vorgelegen.

Hennef, den 13.06.2018

_____

Dennis Kolvenbach

Hennef, den 13.06.2018

_____

Niklas Krol