Automatisierte Reaktionsüberwachung und -beeinflussung mit Mikrocontrollern auf dem Weg zum „Internet of Lab“

Dissertation

zur Erlangung des Grades "Doktor der Naturwissenschaften"
im Promotionsfach Chemie am Fachbereich Chemie, Pharmazie und Geowissenschaften
der Johannes Gutenberg-Universität Mainz

vorgelegt von **Dipl.-Chem. Julian Heinrich**
geb. in Mainz

Mainz, den 13. August 2018

Dekan: [redacted]

1. Berichterstatter: [redacted]
2. Berichterstatter: [redacted]

Tag der mündlichen Prüfung: 26. September 2018

Erklärung der Selbstständigkeit

Die vorliegende Dissertation wurde im Zeitraum vom 01. Februar 2015 bis zum 13. August 2018 am Institut für Organische Chemie des Fachbereichs 09 für Chemie, Pharmazie und Geowissenschaften der Johannes Gutenberg-Universität Mainz unter Betreuung von [redacted] angefertigt.

Hiermit versichere ich, Julian Heinrich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen verwendet habe.

Mainz, den 13. August 2018,

Julian Heinrich

Meiner Familie gewidmet.

Danksagung

Inhaltsverzeichnis

Einleitung

Seit den ersten ausführlich dokumentierten Chargenprozessen während der Zeit der Renaissance [1] werden diese zur Durchführung chemischer Produktionsverfahren stetig weiterentwickelt und sind doch immer dem gleichen Prinzip treu geblieben. Neben der Optimierung der Reaktionsgefäße, der Entwicklung und fortlaufenden Verbesserung analoger Messinstrumente zur Bestimmung von Reaktionsparametern wie Thermometern, Barometern oder (Überdruck-)Ventilen stieg auch die Chargengröße, die innerhalb bestimmter Sicherheitsgrenzen handhabbar ist, bis hin zum heutigen industriellen Maßstab an. Seit dem letzten Drittel des vergangenen Jahrhunderts beschäftigt man sich darüber hinaus mit der Mikroprozesstechnik oder auch Mikroreaktionstechnik, welche eine Abkehr vom herkömmlichen Chargenprozess, hin zu einer kontinuierlichen Synthese bedeutet. Speziell durch den weltweiten Siegeszug der Mikroelektronik konnte auch die Mikroprozesstechnik in den letzten 20 Jahren eine weite Verbreitung finden [2–4].

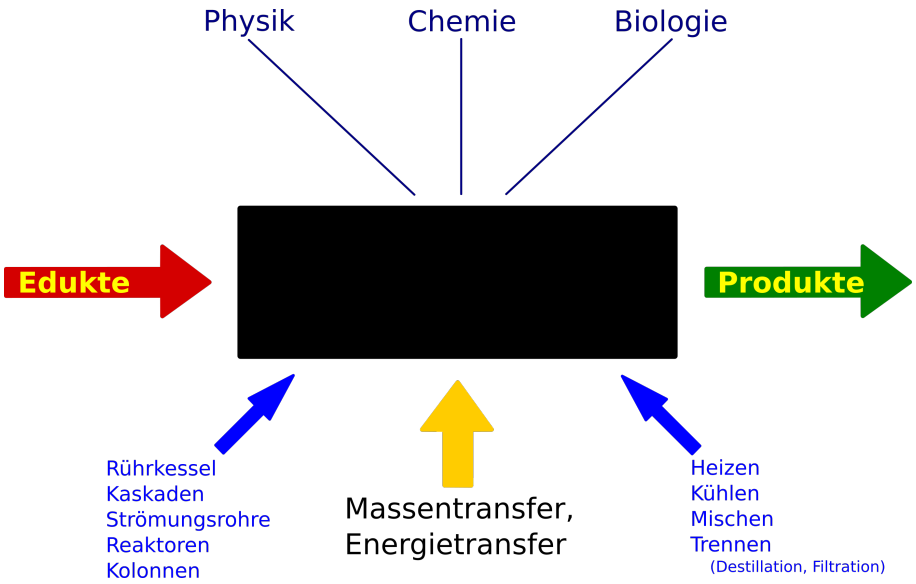


Abbildung 1: Schema des grundlegenden Aufbaus einer chemischen Reaktion in Form einer *Blackbox*, die die Veränderungsfunktion zur Umsetzung der Edukte zu den Produkten enthält.

Allen Verfahren bleibt jedoch ein Grundprinzip gemein, wie es Abbildung 1 anschaulich darstellt: Auf die Edukte, ob in einem Kolben, einem großtechnischen Rührkesselreaktor, in einer Kapillare oder in einem Tropfen wird ein Verfahren angewendet, welches die Edukte zum Produkt oder den Produkten umwandelt. Dieses Verfahren kann von einer einfachen Temperaturänderung bis hin zu einem mehrstufigen Prozess reichen, welchen man selbst in solche Einzelschritte aufteilen kann.

Wirft man einen Blick auf die Informatik und die daraus hervorgegangenen Programmiersprachen, so entdeckt man das identische Grundprinzip auch hier: Ein oder mehrere Werte (Edukte) werden in einer Variablen (Reaktionsgefäß) gespeichert und erfahren im Rahmen einer auf sie angewandten Funktion eine Veränderung, welche als Resultat der Funktion hinterher in eben dieser Variablen oder ein einer anderen Variablen abrufbar ist (Reaktionsprodukt). Diese Analogie in den Grundpfeilern beider Disziplinen befeuert das interdisziplinäre Arbeiten und die Verknüpfung von Informationstechnologie mit chemischen Reaktionen: So, wie für uns Chemiker das Entziffern von Retrosynthesereaktionen aus dem Endprodukt bis hin zu den typischen Edukten der chemischen Industrie beliebt ist, stellt jede Softwareprogrammierung vergleichbare Herausforderungen, da man ein Programmierziel vor Augen hat und dieses mit den vorhandenen (Mess-)Daten erreichen muss.

Diese Arbeit setzt daher ihren Schwerpunkt nicht auf die Entwicklung neuer Reaktionen oder Reaktionswege für bekannte Verfahren, sondern möchte durch die Entwicklung neuer informationstechnischer Verfahren (Software) in Kombination mit der Entwicklung von Verknüpfungsstellen (Sensoren und Aktoren) mit chemischen Reaktionssystemen diese besser erfassbar machen und so zur Verbesserung des Laboralltags (Laborsicherheit, Reaktionsüberwachung und -automatisierung) beitragen. Daher wendet sich die Arbeit bewusst an Chemiker, welche eine Grundkenntnis elektronischer Bauteile und die Offenheit für einfache Programmiersprachen mitbringen sollten.

Kapitel 1

Theoretische Grundlagen

1.1 Mikroreaktoren

Die fortschreitende Miniaturisierung chemischer Reaktionssysteme bringt eine Vielzahl von verfahrens- und reaktionstechnischen Vorteilen und kommt in den letzten Jahren zunehmend nicht nur in der akademischen Forschung und Entwicklung, sondern auch im industriellen Umfeld zum Einsatz. Die kontinuierliche Prozessführung findet in Kanälen, häufig Kapillaren, deren Durchmesser im Mikrometerbereich liegen, statt. Hiermit ist ein entscheidendes Merkmal der Mikrofluidik, dass die Reaktion in kleinen Volumina stattfindet [5], gewährleistet. Auch wenn eine Reaktion innerhalb eines Tröpfchens stattfindet, kann man dieses ebenfalls als Mikrobatchreaktor bezeichnen.

Das Spektrum an Einsatzmöglichkeiten für die Mikroprozesstechnik ist sehr weit und umfasst neben der Synthese von Pharmazeutika und anderen hochreinen Feinchemikalien auch die Partikelherstellung mit definierbaren Größen im Mikro- und Nanobereich, sowie die Nutzung in der Analytik, darunter für High-Throughput-Screenings (HTS), Lab-on-Chip-Anwendungen und automatisierte Analysensysteme [6–13].

Durch die Verringerung der räumlichen Dimensionen im Vergleich zu herkömmlichen Reaktionssystemen steigen per se die Oberflächen-zu-Volumen-Verhältnisse stark an und können Werte bis zu mehrere Hunderttausend $m^2\,m^{-3}$ erreichen, wohingegen in herkömmlichen Reaktorsystemen lediglich Verhältnisse in der Größenordnung von $100 - 1000\ m^2\,m^{-3}$ erreicht werden [14].

Weitere Vorteile gegenüber konventionellen Syntheseverfahren erwachsen aus den enormen spezifischen Oberflächen und der damit verbundenen Intensivierung von Stoff- und Wärmetransportprozessen, was gerade bei stark exothermen oder mischungssensitiven Reaktionen zum Tragen kommt. Neben der Sicherheit können auch die Selektivität, Ausbeute und Produktqualität signifikant verbessert werden. Manche Reaktionen bzw. Reaktionsprodukte

sind auf diesem Wege überhaupt erst zugänglich [5, 6, 8]. Durch die Miniaturisierung sinkt das Gefährdungspotential beim Einsatz von sehr hohen Temperaturen und/oder Drücken, was neue Synthesewege öffnet [15–17]. Die Mikroprozesstechnik erlaubt neben der Kontrolle der Reaktionszeit darüber hinaus eine Verweilzeitkontrolle, welche es ermöglicht, hoch reaktive oder auch instabile Zwischenprodukte zu handhaben und gezielt umzusetzen [18].

Die reine Miniaturisierung eines bestehenden Reaktionssystems macht dieses nicht zwingend zu einem Mikroreaktor, erst die Kombination von einzelnen Elementen in denen verfahrenstechnische Grundoperationen wie Mischen, Reagieren, Kühlen etc., beim Durchfluss einer Reaktionslösung kontrolliert und vor allem schnell ablaufen, macht ein System zu jenem. Zum Betrieb sind Kenntnisse wichtiger Betriebs- und Reaktionsbedingungen des mikrofluidischen Systems notwendig. Neben dem Druck, der Zusammensetzung, der Temperatur und der Konzentration der einzelnen Komponenten wird die Reaktion auch von der Fluiddynamik, der Mischeffizienz und der Verweilzeitverteilung stark beeinflusst. Insbesondere ein Missverhältnis der Verweilzeit von Reaktanden wirkt sich direkt und maßgeblich auf die Selektivität und Reaktorleistung und damit den Umsatz und die Produktausbeute einer mikrofluidischen Reaktion aus. Nur bei genauem Wissen um die Verweilzeitverteilung kann ein Mikroreaktor effizient für einen kontinuierlichen chemischen Prozess eingesetzt werden [19].

Umso wichtiger ist es, von Beginn an Reaktionsparameter zu optimieren und möglichst umfassend über die Auswirkung von Parameteränderungen in die ein oder andere Richtung Bescheid zu wissen. Die in dieser Arbeit vorgestellten Bausteine eines mikrofluidischen Systems unterstützen den Experimentator bei der Ermittlung dieser wichtigen Reaktionsparameter. Mit Hilfe von Sensoren zur Parametererfassung und Aktoren zur Parameteranpassung und einer zentralen Koordinierungsstelle (Computer) erfolgt die Automatisierung des vollständigen Reaktionszyklus. Kombiniert mit einer sich daran anschließenden automatisierten Produktanalytik kann ein zeit- und kostensparendes Parameterscreening durchgeführt werden.

1.2 Segmentgeneratoren

Zunehmender Beliebtheit erfreuen sich tropfenbasierte Reaktionen im kontinuierlichen Fluss, da diese gerade bei Reaktionen, deren Reaktionsparameter eine besondere Kontrolle und Handhabung erfordern, ihre Stärken ausspielen. Wie bereits erwähnt, stellt bei solchen Reaktionen jeder Tropfen einen eigenständigen Mikrobatchreaktor[1] dar, sodass

[1] Abgeleitet vom Batchreaktor ist der auf einen Tropfen verkleinerte Mikrobatchreaktor in seinen Reaktionsparametern scharf definierbar, sodass insbesondere schwer zugängliche Reaktionspfade erreichbar werden.

in jedem Tropfen immer gleiche Bedingungen vorherrschen. Solche tropfenbasierte Reaktionen sind vergleichbar der Flaschengärung von Sekt oder Champagner. In jede Flasche werden die gleichen Edukte eingefüllt; jede Flasche ist ein eigener (Mikro)reaktor zur alkoholischen Gärung; am Ende erhält man in jeder Flasche das identische Produkt einer Charge. Übertragen auf die Mikrofluidik bedeutet dies, dass man mit tropfenbasierten Reaktionssystemen viele Hundert, Tausend oder gar Hunderttausend in sich geschlossene, eigenständige Mikrobatchreaktionen durchführt und eben diese hinterher wieder zu einem Endprodukt zusammenführen kann.

Neben dieser skalierbaren, immer gleichen Mikroreaktornutzung lassen sich tropfenbasierte Systeme darüber hinaus hervorragend zum Parameterscreening chemischer Reaktionen nutzen. Da jeder Tropfen einen abgeschlossenen Mikroreaktor darstellt, kann neben der Reproduzierbarkeit einer Reaktion über viele Tropfen hinweg auch eine schnelle Parameteränderung je Tropfen oder über ein Tropfenensemble hinweg geschehen. Dies erfordert keine Veränderung oder langwierige Spülprozesse nach Parameteränderung für eine bestimmte Reaktion, da die häufigsten veränderlichen Parameter wie Temperatur, Zusammensetzung der Edukte, Verweilzeit und Spannung bzw. Stromstärke bei elektrochemischen Reaktionen komfortabel und schnell für kleine Volumina variiert werden können.

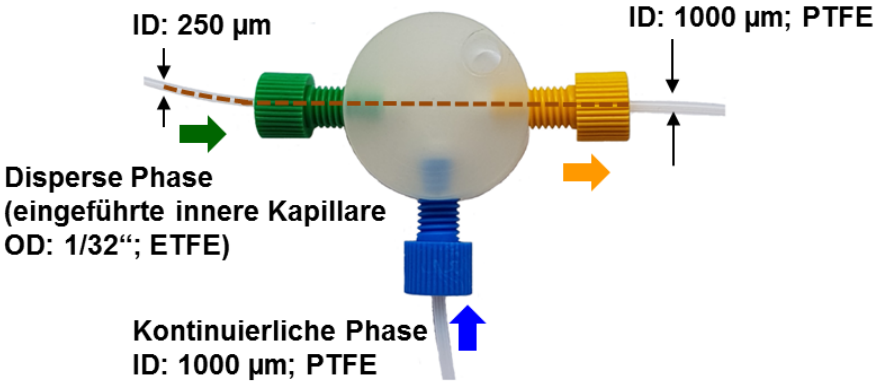


Abbildung 1.1: Aufbau eines koaxialen Tropfengenerators

Zur Segmentgeneration eignen sich verschiedene mikrofluidische Bauteile. Die bekannten T- und Y-Verbinder, welche sich durch das direkte bzw. rechtwinklige Aufeinandertreffen der Tropfenphase mit der kontinuierlichen Phase nur bedingt zur Tropfengeneration eignen, da die Tropfendispersität zwischen einem Spray und einem Slug[2] variieren kann, insbesondere wenn es innerhalb des Systems zu Druckschwankungen kommt. Deutlich besser eignet sich ein schon in [20–23] vorgestellter, aus mehreren koaxialen Kapillaren aufgebauter Tropfengenerator. In vereinfachter Form für zweiphasige Systeme, in welcher

[2]Verformter Tropfen in einem zweiphasigen Fluss, dessen Länge größer als der Kanaldurchmesser ist.

eine Kapillare mit geringem Durchmesser in eine größere Kapillare eingebracht wird, wird der an der Austrittsöffnung der inneren Kapillare entstehende Reaktionstropfen durch die von außen umspülende kontinuierliche Phase abgerissen. In einem solchen System lässt sich durch Anpassung des Flussratenverhältnisses und der Gesamtflussrate die Größe der entstehenden Tropfen sehr genau definieren.

Die generierten Tropfen lassen sich nicht nur als eigenständige Mikroreaktoren in einer kontinuierlichen Phase nutzen, sondern durch das große Oberflächen-zu-Volumen-Verhältnis auch hervorragend für Phasengrenzflächenreaktionen verwenden[20]. Eine Weiterentwicklung stellt der Janus-Tropfengenerator dar, der für elektrochemische Reaktionen im kontinuierlichen Fluss in Kapitel 3.1 entwickelt wurde. Durch die Anordnung zweier sehr kleiner Kapillaren nebeneinander innerhalb einer Kapillare größeren Durchmessers, entstehen sogenannte Janustropfen[3].

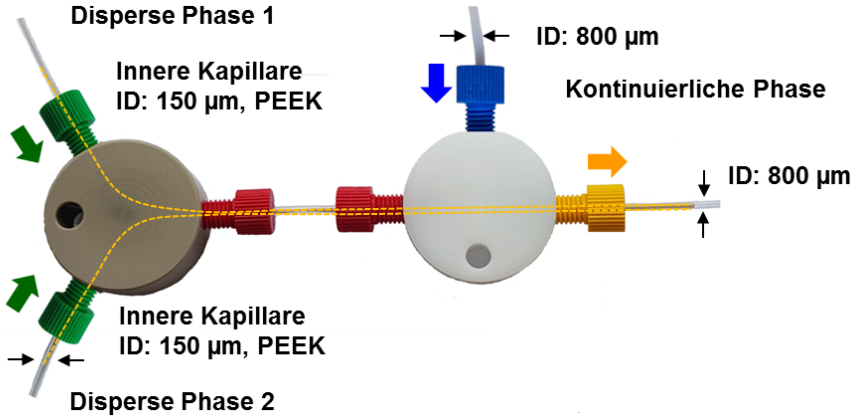
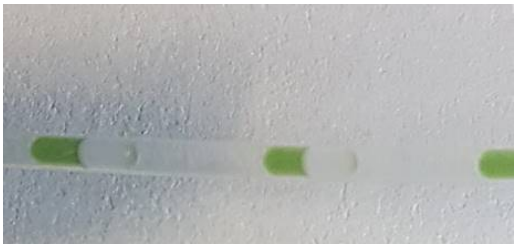
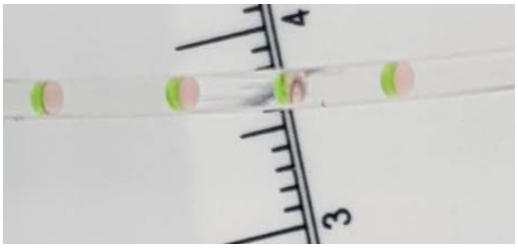


Abbildung 1.2: Aufbau eines koaxialen Janustropfengenerators



(a) Janustropfen aus Wasser (grün) und FC-40® in Toluol



(b) Janustropfen aus Wasser (grün) und Dichlorethan (rot) in FC-40®

Abbildung 1.3: Beispiele für mit obigem Tropfengenerator generierte Janustropfen

[3]Janustropfen [24–28] bestehen, vergleichbar mit Januspartikeln, aus zwei Teilen, die, nicht mischbar, den Tropfen bilden. Zwischen den beiden Teilen besteht eine direkte Kontaktoberfläche, die nicht durch die den Tropfen umgebende kontinuierliche Phase unterbrochen wird.

1.3 Mikrocontroller

Unter dem Begriff Mikrocontroller versteht man ein meistens auf einem Halbleiterchip integrierten Prozessor mitsamt unterschiedlichen, teilweise spezialisierten Zusatzkomponenten. Dazu zählen neben dem Arbeits- und Permanentspeicher auch digitale und analoge Ein- und Ausgänge, Timer und weitere elektronische Schaltungen und Schnittstellen wie beispielsweise *UART*[4], *I²C*[5], *1-Wire*[6] oder *SPI*[7].

Die Benennung von Mikrocontrollern erfolgt häufig nach der Bit-Zahl ihres internen Datenbusses: 4, 8, 16 oder 32 Bit. Die Bit-Zahl entspricht der Länge der Daten die der Mikrocontroller in einem einzigen Befehl verarbeiten kann. So können die in dieser Arbeit überwiegend verwendeten 8 Bit-Mikrocontroller[8] in einem Befehl immer nur Zahlen bis 255 verarbeiten. Zur Bearbeitung größerer Zahlen sind mehrere Befehle hintereinander nötig, was längere Ausführungszeit in Anspruch nimmt. Durch diese Einschränkung sind die reinen Rechengrenzen des Prozessors eines Mikrocontrollers schnell erreicht, was jedoch nicht dem Anwendungszweck eines Mikrocontrollers entspricht. Der Mikrocontroller spielt seine Stärken durch die Integration von Zusatzkomponenten aus, die neben der schnellen Aufnahme und Abgabe von digitalen und analogen Zuständen, ermöglicht durch die direkt an den Prozessor angebundenen digitalen und analogen Ein- und Ausgänge, eine Abarbeitung einfacher Aufgaben in Echtzeit erlaubt. Diese Fähigkeit muss heutigen PCs erst mit darauf spezialisierten Echtzeit-Betriebssystemen aufwändig erneut beigebracht werden[9], um ein deterministisches zeitliches Verhalten zurück zu gewinnen.

Die in vielen Einzelprojekten dieser Arbeit genutzte Arduino-Plattform[29–34] ist eine im Jahre 2005 von Massimo Banzi und David Cuartielles vorgestellte[35], quell-offene Hard-

[4] *UART*, Abkürzung für *Universal Asynchronous Receiver Transmitter*, einer elektronischen Schaltung zur Bereitstellung von digitalen seriellen Schnittstellen

[5] *I²C*, Abkürzung für *Inter-Integrated Circuit*, ein von Philips 1982 vorgestellter serieller Datenbus mit Master-Slave-Architektur, beispielsweise zur Anbindung von Displays (siehe Kapitel 2.2.1.1, Aufbau eines Tropfenzählers)

[6] *1-Wire*, der sogenannte Eindrahtbus, ist eine von Dallas Semiconductor entwickelte serielle Schnittstelle, welche mit einer einzelnen Datenader (neben der Masse) sowohl für die Sende- als auch die Empfangsrichtung arbeitet (siehe Kapitel 2.1, Temperatursensoren).

[7] *SPI*, Abkürzung für *Serial Peripheral Interface* ist ein hauptsächlich geräteintern genutzer serieller Datenbus mit Master-Slave-Architektur, der Anfang der 1980er Jahre von Motorola entwickelt wurde (siehe Kapitel 3.2, Phasentrenner).

[8] 8 Bit ergibt 1 Byte und entspricht einem im Dezimalsystem darstellbaren Wert zwischen 0 und 255

[9] Herkömmliche Betriebssysteme können eine Reaktion auf eine zufällige Eingabe innerhalb einer festgelegten, sehr kurzen Reaktionszeit nicht garantieren, da sie mit Lese- und Schreibpuffern arbeiten, die Datenverarbeitung verlangsamen und die erforderliche Abarbeitung nach Eingangsreihenfolge nicht garantieren können. Echtzeitbetriebssysteme (RTOS - Real Time Operating Systems) sind speziell für solche zeitkritischen Szenarien entwickelt, wie sie im Maschinenbau, der Medizintechnik und anderen Bereichen mit zeitkritischen Anwendungen vorkommen.

und Softwareplattform für einfache Mikrocontroller mit analogen und digitalen Ein- und Ausgängen. Die Mikrocontroller können über eine leicht verständliche Entwicklungsumgebung, die Arduino-IDE[10], in der Programmiersprache *Processing*, einer Vereinfachung der Programmiersprache Java[11], programmiert werden. Die Arduino-IDE erlaubt eine direkte Programmierung des Mikrocontrollers, ohne dass separate Programmieradapter notwendig sind. Arduino-Boards sind so aufgebaut, dass sie auf Grund ihrer einheitlichen Größe und Buchsenleisten leicht um weitere Platinen, sogenannte Shields, durch Aufstecken eben dieser erweitert werden können, welche den Mikrocontroller um zusätzliche Funktionen ergänzen. So ist es beispielsweise möglich, den Mikrocontroller um die nötige Hardware zum Speichern von Messdaten auf SD-Karten oder zur Ansteuerung von Motoren, Messgeräten und Pumpen zu erweitern.

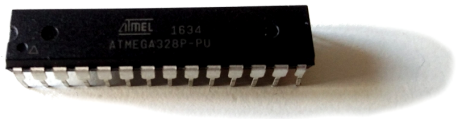


Abbildung 1.4: Mikrocontroller ATMEGA328P

Die aktuelle Generation der 8-Bit-Arduino-Plattform, der Arduino Uno[12], basiert auf einem ATMEGA328P der Firma Atmel. Auf dem Board sind 14 digitale Ein- und Ausgänge, wovon 6 Pulsweitenmodulations-Signale (kurz: PWM-Signale), vergleichbar analogen Ausgängen, ausgeben können, direkt auf Buchsenleisten heraus geführt. Zusätzlich sind 6 analoge Eingänge über einen im Mikrocontroller integrierten Analog-Digital-Wandler[13] zugänglich, die unter Anderem für das Auslesen analoger Sensoren nötig sind. Der Arduino Uno verfügt über *I²C*-, *SPI*- und *1-Wire*-Schnittstellen, um weitere Peripheriegeräte in Form von ICs[14] anzubinden. Die analogen Ein- und Ausgänge können bei Bedarf auch als digitale Ein- und Ausgänge genutzt werden. Der Arduino kann wahlweise über die USB-Schnittstelle eines angeschlossenen Computers oder mittels separatem 9 V-Netzteil mit Strom versorgt werden. Über den USB-Anschluss kommuniziert der Arduino mit einem Computer und wird darüber hinaus mit Programmcode bestückt. Zur einfachen Datenvisualisierung, Datenspeicherung, Programmierung und Übertragung des Programmcodes steht die bereits erwähnte Arduino-IDE zur Verfügung.

[10] *IDE*: integrierte Entwicklungsumgebung, vom engl. *Integrated Development Environment*, Softwareumgebung zum Erstellen von Programmen für den Arduino Mikrocontroller[29]

[11] Java ist eine 1995 von Sun Microsystems entwickelte objektorientierte Programmiersprache, die heute zu den weltweit beliebtesten Programmiersprachen zählt.

[12] Datenblatt des Arduino Uno siehe Anhang B

[13] Der im Arduino Uno integrierte Analog-Digital-Wandler hat eine Auflösung von 10 Bit, was einer Unterscheidung einer Spannung zwischen 0 und 5 Volt in 1024 Schritte entspricht.

[14] IC, engl. *Integrated Circuit*, integrierter Schaltkreis, typischerweise realisiert als eigenständiger Chip mit einer standardisierten Bauform und Anzahl an Anschlüssen.

Kapitel 2

Sensoren

Sensoren sind Messgeräte und somit die Datenlieferanten für Mikrocontroller und Computer. Mit ihrer Hilfe können Daten dem Steuerungssystem zugeführt werden: auch eine Tastatur lässt sich als eine Ansammlung vieler Druck- bzw. kapazitiver Sensoren beschreiben. In diesem Kapitel werden unterschiedliche Sensoren, wie sie zur Reaktionsüberwachung eingesetzt werden können, und deren Aufbau betrachtet. Neben dem Thermometer, dem analogen wie digitalen Sensor zur chemischen Reaktionskontrolle schlechthin, braucht man bei der automatisierten Kontrolle von Reaktionen weitere Möglichkeiten, unterschiedlichste Reaktionsparameter kontinuierlich zu ermitteln und im Verlauf mittels Aktoren auch steuern zu können.

2.1 Temperatursensoren

Zur digitalen Messung von Temperaturen werden üblicherweise Widerstandsthermometer verwendet. Dies sind elektrische Bauteile, deren Widerstandswert sich mit der Temperatur ändert. Naheliegenderweise eignen sich hierfür reine Metalle, beispielsweise Platin, deren Widerstandsänderung eine lineare Temperaturabhängigkeit aufweisen. Heute lassen sich auch aus Keramiken und Halbleitern solche Widerstandsthermometer, sogenannte Thermistoren, herstellen. Diese sind um einen Faktor 10 empfindlicher, als herkömmliche auf Metallen basierende Thermometer. Man unterscheidet zwischen zwei Gruppen von Thermistoren, den Heißleitern und den Kaltleitern. Bei Heißleitern steigt die Leitfähigkeit mit Erhöhung der Temperatur, sie besitzen somit einen negativen Temperaturkoeffizienten. Kaltleiter hingegen haben einen positiven Temperaturkoeffizienten: die Leitfähigkeit sinkt und der Widerstand steigt mit steigender Temperatur. Kaltleiter werden in der Elektrotechnik bevorzugt als Überlastungsschutz bei Geräten im Dauerbetrieb eingesetzt: sie begrenzen bei zu starker Erwärmung den Stromdurchfluss und schützen das Gerät vor Überhitzung. Mit Heißleitern hingegen lassen sich heute Temperaturen bis 150°C präzise messen. Nach vorheriger Kalibrierung (innerhalb der Messsoftware) lässt sich die Temperatur direkt aus dem gemessenen Widerstand bestimmen.

Eine Weiterentwicklung dieses Prinzips, welches sich leichter in einen Mikrochip integrieren lässt, ist die Temperaturermittlung mittels eines sogenannten *switched-capacitor integrator*, einem als Integrierer eingesetzten Operationsverstärker mit geschalteten Kondensatoren. Die geschalteten Kondensatoren arbeiten hier äquivalent zu Widerständen. Bei der Chip-Herstellung wird neben dem sowieso erforderlichen Prozessschritt für Kondensatoren auf den zusätzlichen Prozessschritt für Widerstände verzichtet.

(a) Verkapselte Form des DS18S20

(b) TO-92 Form des DS18S20

Abbildung 2.1: Temperatursensor Maxim Integrated DS18S20

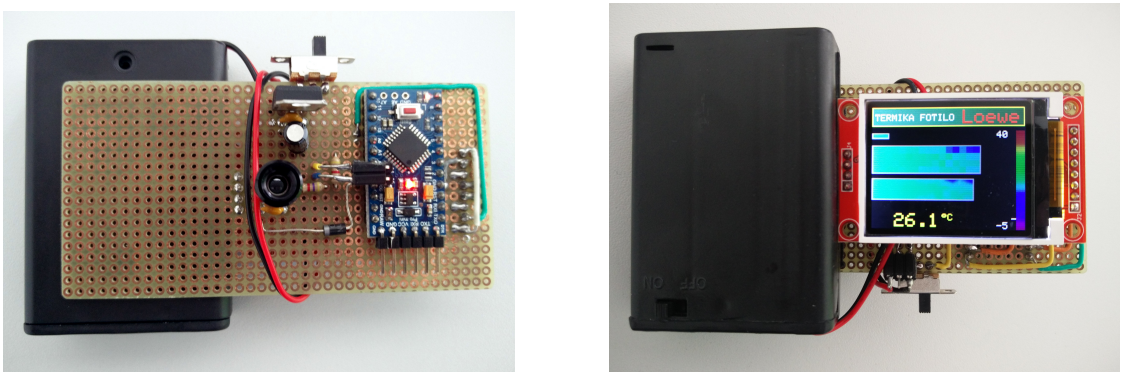
Der in dieser Arbeit zur Temperaturermittlung eingesetzte Temperatursensor **DS18S20**[1] stammt von *Maxim Integrated (früher Dallas Semiconductor)* und nutzt die in [36] beschriebene *direct-to-digital thermometer*-Technologie zur Temperaturermittlung. Der Sensor ist mit der von *Dallas Semiconductor* entwickelten 1-Wire-Schnittstelle (auch One-Wire geschrieben) ausgestattet, welche mittels einer einfachen seriellen Kommunikation über eine einzelne Datenleitung (zuzüglich einer Masseleitung und einer optionalen Leitung für die Betriebsspannung) von seinem Gegenüber abgefragt wird. An dieser Datenschnittstelle können beliebig viele weitere 1-Wire-Komponenten hängen, da jede über eine eindeutige ID auf dem Datenbus identifiziert und angesprochen wird. Durch die weite Verbreitung des Sensors und gute Unterstützung des 1-Wire-Busses auf diversen Mikrocontroller-Plattformen und bei Kleinstcomputern ist eine Einbindung dieses Temperatursensors in unterschiedlichste Systeme sehr einfach. Der Sensor ist sowohl in offener TO-92-Bauform als auch fertig flüssigkeitsdicht verkapselt preiswert am Markt verfügbar. In der verkapselten Bauweise kommt der Temperatursensor direkt zur Temperaturmessung in Wärme- oder Kältebädern, im Luftstrom oder an Kühlkörpern zum Einsatz.

2.1.1 Pyrometer

Neben der Bestimmung der Temperatur von Wärme- und Kältebädern, durch welche Kapillaren führen, in denen die kontinuierliche Reaktion stattfindet, ist für manche Reaktionen auch die direkte Kontrolle der Reaktionstemperatur interessant. Diese lässt sich sehr einfach mit Hilfe eines Pyrometers bestimmen. Verfügt es über eine Vielzahl einzelner Infrarotsensoren und ist dazu in der Lage, die gewonnenen Informationen zu einem Bild

[1] Datenblatt des DS18S20 siehe Anhang B

zusammenzusetzen, spricht man von einer Thermografie- oder Wärmebildkamera. Da die Anschaffung solcher Geräte mit hohen Kosten verbunden ist, wurde für diese Arbeit eine Möglichkeit gesucht, durch Selbstbau einer Thermografiekamera die zeitliche Veränderung der Temperatur und damit den Verlauf von Reaktionen zu kontrollieren. Zur digitalen Verarbeitung im Mikrocontrollers ist nur die Messung der Temperaturwerte jedes einzelnen Bildpunkts nötig, eine Umwandlung zu einer kolorierten Wiedergabe kann bei Bedarf lokal im Mikrocontroller zur grafischen Ausgabe auf einem Display oder innerhalb der Controller-Software am Client-Computer erfolgen.



(a) Objektivseite (b) Rückseite

Abbildung 2.2: Fertig gelötete Termika Fotilo, mit Batteriefach und Display

Die im Folgenden vorgestellte Wärmebildkamera entstammt einem Projekt zum Selbstbau einer Wärmebildkamera aus der Zeitschrift Make 3/2017 mit dem Titel *Termika Fotilo* [37]. Das Projekt basiert auf dem Chip MLX90261[2] von Melexis, einem Infrarot-Mess-Chip mit einer Matrix aus 16 mal 4 Messpunkten für einen Temperaturbereich bis 300°C. Abbildung 2.2 (a) zeigt den Chip in seinem runden Gehäuse mittig auf der Platine. Die fertige Wärmebildkamera verfügt neben der Echtzeitanzeige der gemessenen Temperaturwerte auch über eine serielle Übertragung dieser Werte direkt auf einen via USB angeschlossenen Computer. Dort werden die Daten wahlweise direkt weiter verarbeitet oder in Form einer Logdatei dauerhaft gespeichert.

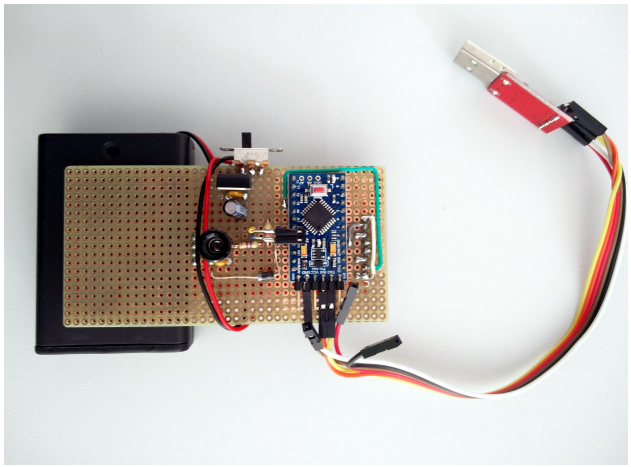


Abbildung 2.3: Termika Fotilo mit angeschlossenem USB-Seriell-Wandler zur Datenübertragung

[2]Datenblatt des MLX90261 siehe Anhang B

2.1.1.1 Technische Realisierung

Der gesamte Aufbau der Wärmebildkamera befindet sich auf einer beidseitig bestückten Lochrasterplatine, worauf neben dem IR-Chip und dem Display zur Visualisierung der Messwerte auch der verwendete Mikrocontroller Arduino Pro Mini[3] und ein Batteriefach zur portablen Stromversorgung Platz finden. Der Schaltplan ist in Abbildung 2.4 gezeigt. Da der MLX90261 innerhalb eines Spannungsbereichs zwischen 2,6 V und 3,2 V arbeitet und initial bei 2,6 V kalibriert wurde, müssen alle verwendeten Bauteile für eine Spannung unter 3,3 V ausgelegt sein. Der hier verwendete Arduino Pro Mini ist in einer speziellen 3,3 V-Version erhältlich. Bei dieser Spannung läuft der ATMEGA328P-Prozessor nur mit einem Takt von 8 MHz, was keine umfangreichen Berechnungen lokal auf dem Mikrocontroller zulässt. Auch das verwendete 1,8" TFT-Display, basierend auf dem ST7735-Chip, ist für die niedrige Betriebsspannung geeignet.

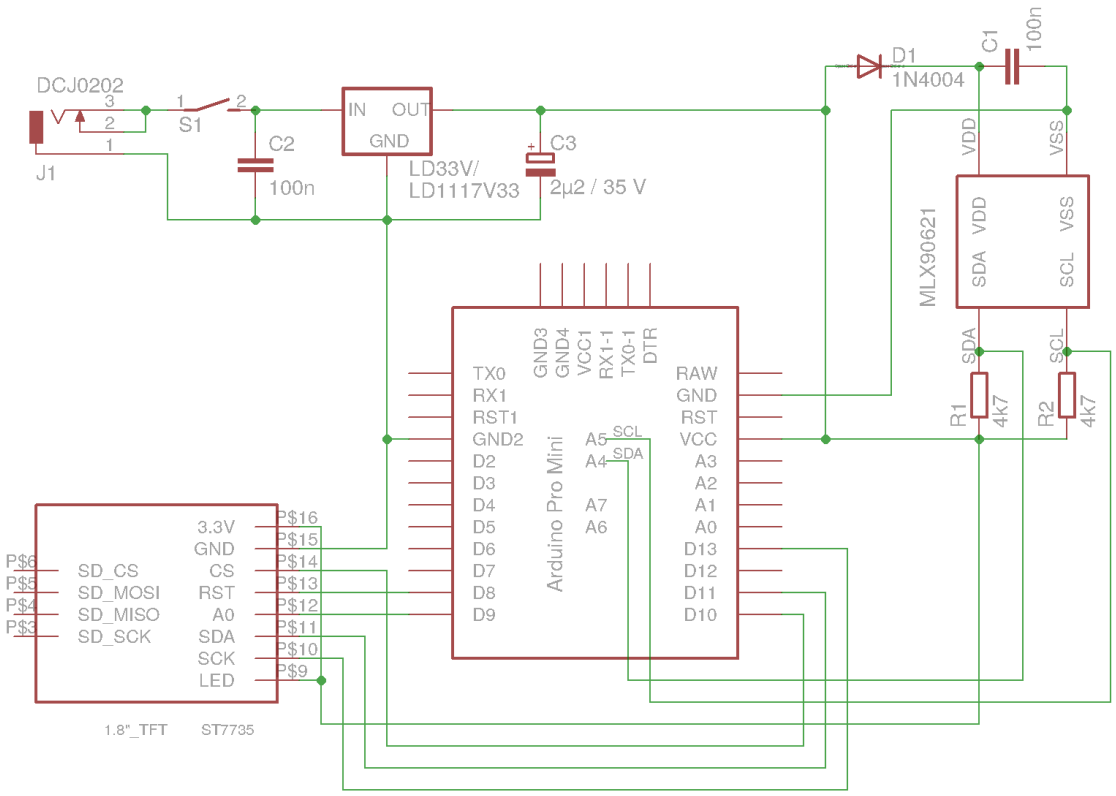


Abbildung 2.4: Schaltplan des Pyrometers auf Basis des MLX90621

[3]Der Arduino Pro Mini nutzt den identischen Microcontroller ATMEGA328P von Atmel, besitzt jedoch eine stark verringerte Bauform und kann neben 5 V auch mit einer Betriebsspannung von 3,3 V bei verringerter Taktfrequenz betrieben werden.

2.1.1.2 Software

Zusätzlich zu den zwei Standardbibliotheken `SPI.h` und `TFT.h` wird noch die im Anhang A.1 vollständig gezeigte und von der Make-Redaktion online zur Verfügung gestellte Bibliothek zur Ansteuerung des MLX90621, `mlx90621.h` und `mlx90621.cpp`, benötigt. Neben der Initialisierung verschiedener Parameter lässt sich innerhalb der Initialisierungsroutine auch der für die grafische Ausgabe wichtige zu erwartende Temperaturbereich angeben (`MINTEMP` und `MAXTEMP`).

Listing 2.1: Quellcodeausschnitt: Initialisierungsroutine - Pyrometer

```
1  /* ursprüngliche Quelle: Make 3/2017, "Termika Fotilo"     */
2  /* keine weitere Lizenz durch die Originalquelle angegeben */
3  #include <SPI.h>
4  #include <TFT.h>
5  #include "mlx90621.h"
6  uint8_t CS_  = 10;
7  uint8_t DC_  = 9;
8  uint8_t RST_ = 8;
9  int8_t MINTEMP = -5;
10 uint8_t MAXTEMP = 30;
11 uint16_t HUEMAX = 320;
12 uint8_t ZOOM = 7;
13 void StartScreen(void);
14 void OutAmbientTemp(void);
15 void HSVtoRGB(float &, float &, float &, float, float, float);
16 void OutTempField(void);
17 float LinInterpol (float, float, uint8_t, uint8_t, uint8_t);
18 float FindMinTemp (float *); float FindMaxTemp (float *);
19 TFT TFTscreen = TFT(CS_, DC_, RST_);
20 MLX90621 MLXtemp;
21 void setup() { }
```

Der Großteil der auf dem Mikrocontroller laufenden Software wurde in separate Funktionen ausgelagert, so sorgt die Funktion `StartScreen` für die Initialisierung und den Grundaufbau der grafischen Ausgabe auf dem Display, die Funktion `OutAmbientTemp` sorgt für die Anzeige der Umgebungstemperatur, mittels der Funktion `OutTempField` wird nach der linearen Interpolation der Temperaturwerte durch `LinInterpol` und der Umrechnung in RGB-Werte (`HSVtoRGB`) das Wärmebild auf dem Display und als Rohdaten über die serielle Schnittstelle ausgegeben.

Listing 2.2: Quellcodeausschnitt: Ausgelagerte Funktionen - Pyrometer

```
/* ursprüngliche Quelle: Make 3/2017, "Termika Fotilo"     */
/* keine weitere Lizenz durch die Originalquelle angegeben */
void StartScreen(void) {
  uint8_t i;
  float R, G, B;
  float gradstep;
  char puffer[10];
  TFTscreen.begin ();
  TFTscreen.background (0, 0, 0);
  TFTscreen.stroke (0, 0xFF, 0xFF);
  TFTscreen.fill (50, 100, 20);
  TFTscreen.rect (0, 0, 159, 20);
  TFTscreen.stroke (0,0,255);
  TFTscreen.setTextSize (2);
  TFTscreen.text ("Loewe",95,3);
  TFTscreen.stroke (0xFF , 0xFF, 0xFF);
  TFTscreen.setTextSize (1);
  TFTscreen.text ("TERMIKA FOTILO",3,7);
  TFTscreen.stroke (0x50, 0x50, 0xFF);
  TFTscreen.setTextSize (1);
  TFTscreen.text ("Init...", 0, 112);
  TFTscreen.text ("Please wait", 0, 120);
  TFTscreen.stroke (127, 127, 127);
  TFTscreen.fill (100, 0, 0);
  TFTscreen.rect (0, 27, 16+2, 4+2);
  TFTscreen.rect (0, 40, 16*ZOOM+2, 4*ZOOM+2);
  TFTscreen.rect (0, 75, 15*ZOOM+2, 3*ZOOM+2);
  for (i=0; i < 103; i++) {
    HSVtoRGB (R, G, B, i * (HUEMAX/103.0), 1, .5);
    TFTscreen.stroke (B * 255, G * 255, R * 255);
    TFTscreen.line (150, i+25, 159, i+25);
  }
  TFTscreen.stroke (0xFF , 0xFF, 0xFF);
  TFTscreen.setTextSize (1);
  itoa (MINTEMP, puffer, 10);
  TFTscreen.text (puffer, 130, 120);
  itoa (MAXTEMP, puffer, 10);
  TFTscreen.text (puffer, 130, 25);
  gradstep = (MAXTEMP + abs(MINTEMP)) / 103.0;
  i = 25 + (uint8_t)(MAXTEMP / gradstep);
  TFTscreen.line (145, i, 150, i);
}

void OutAmbientTemp(void) {
  float t = MLXtemp.get_ptat();
  char puffer[10];
  TFTscreen.stroke (0, 0, 0);
```

```
48    TFTscreen.fill (0, 0, 0);
49    TFTscreen.rect (0, 112, 70, 128-112);
50    TFTscreen.stroke (0, 0xFF, 0xFF);
51    TFTscreen.setTextSize (2);
52    dtostrf (t, 6, 1, puffer);
53    TFTscreen.text (puffer, 0, 112);
54    TFTscreen.setTextSize (1);
55    TFTscreen.text ("\xF7", 74, 112);
56    TFTscreen.text ("C", 80, 112);
57  }
58
59  void HSVtoRGB(float &r, float &g, float &b, float h, float s, float v
        ) {
60    int i;
61    float f, p, q, t;
62    if( s == 0 ) {
63      r = g = b = v;
64      return;
65    }
66    h /= 60;
67    i = floor( h );
68    f = h - i;
69    p = v * ( 1 - s );
70    q = v * ( 1 - s * f );
71    t = v * ( 1 - s * ( 1 - f ) );
72    switch( i ) {
73      case 0: r = v; g = t; b = p; break;
74      case 1: r = q; g = v; b = p; break;
75      case 2: r = p; g = v; b = t; break;
76      case 3: r = p; g = q; b = v; break;
77      case 4: r = t; g = p; b = v; break;
78      default: r = v; g = p; b = q; break;
79    }
80  }
81
82  void OutTempField(void) {
83    int8_t x, y, xmod, xorg1, xorg2, ymod, yorg1, yorg2;
84    char puffer[10];
85    float temps[16][4];
86    float i, i1, i2;
87    float hue;
88    float R, G, B;
89    float interpoltemp;
90    MLXtemp.read_all_irfield (temps);
91    for (y = 0; y < 4; y++) {
92      for (x = 0; x < 8; x++) {
93        i = temps[15 - x][y];
94        temps[15 - x][y] = temps[x][y];
95        temps[x][y] = i;
```

```
96      }
97    }
98    for (y = 0; y < 4; y++) {
99      for (x = 0; x < 16; x++) {
100       hue = HUEMAX - (temps[x][y] + abs(MINTEMP)) * (HUEMAX / (float)
        (MAXTEMP + abs(MINTEMP)));
101       HSVtoRGB (R, G, B, hue, 1, .5);
102       TFTscreen.stroke (B * 255, G * 255, R * 255);
103       TFTscreen.fill (B * 255, G * 255, R * 255);
104       TFTscreen.point (x + 1, y + 28);
105       TFTscreen.rect (x * ZOOM + 1, y * ZOOM + 41, ZOOM, ZOOM);
106       dtostrf (temps[x][y], 6, 1, puffer);
107       Serial.print (puffer);
108     }
109     Serial.println();
110   }
111   Serial.println("***");
112   for(x = 0; x < (ZOOM * 15); x++) {
113     for(y = 0; y < (ZOOM * 3); y++) {
114       xmod = x % ZOOM;
115       xorg1 = x - xmod;
116       xorg2 = xorg1 + ZOOM;
117       ymod = y % ZOOM;
118       yorg1 = y - ymod;
119       yorg2 = yorg1 + ZOOM;
120       i1 = LinInterpol (temps[x / ZOOM][y / ZOOM], temps[x / ZOOM +
        1][y / ZOOM], xorg1 , xorg2, x);
121       i2 = LinInterpol (temps[x / ZOOM][y / ZOOM + 1], temps[x / ZOOM
        + 1][y / ZOOM + 1], xorg1 , xorg2, x);
122       interpoltemp = LinInterpol (i1, i2, yorg1, yorg2, y);
123       hue = HUEMAX - (interpoltemp + abs(MINTEMP)) * (HUEMAX / (float
        )(MAXTEMP + abs(MINTEMP)));
124       HSVtoRGB (R, G, B, hue, 1, .5);
125       TFTscreen.stroke (B * 255, G * 255, R * 255);
126       TFTscreen.fill (B * 255, G * 255, R * 255);
127       TFTscreen.point (x + 1, y + 75 + 1);
128     }
129   }
130 }
131
132 float LinInterpol (float t1, float t2, uint8_t x1, uint8_t x2,
      uint8_t x) {
133   return t1 * (x2 - x) / ZOOM + t2 * (x - x1) / ZOOM;
134 }
135
136 float FindMinTemp (float temperatures[]) {
137   uint8_t i;
138   float temp = 300.0;
139   for (i = 0; i < 64; i++) {
```

```
140     if (temperatures[i] < temp)
141       temp = temperatures[i];
142   }
143   return temp;
144 }
145
146 float FindMaxTemp (float temperatures[]) {
147   uint8_t i;
148   float temp = -20.0;
149   for (i = 0; i < 64; i++) {
150     if (temperatures[i] > temp)
151       temp = temperatures[i];
152   }
153   return temp;
154 }
```

Innerhalb der Wiederholungsroutine wird mit jedem Zyklus die Schnittstelle und der Wärmebildchip initialisiert, die Ausgabe am Display vorbereitet, die Umgebungstemperatur gemessen und dann das Wärmebild aufgenommen, zur Anzeige am Display aufbereitet, über die serielle Schnittstelle übertragen und am Ende auf dem Display ausgegeben. All dies geschieht über den Aufruf der zuvor deklarierten Funktionen.

Listing 2.3: Quellcodeausschnitt: Wiederholungsroutine - Pyrometer

```
1  /* ursprüngliche Quelle: Make 3/2017, "Termika Fotilo"     */
2  /* keine weitere Lizenz durch die Originalquelle angegeben */
3
4  void loop() {
5    Serial.begin(9600);
6    Serial.println("Starting....");
7    StartScreen();
8    while (!MLXtemp.init())         // MLX90620 init failed
9      delay (100);
10   while (1) {
11     OutAmbientTemp();
12     OutTempField();
13     delay(100);
14   }
15 }
```

2.1.1.3 Anwendungsbeispiel

Zur Prüfung der Verwendbarkeit der Termika Fotilo unter Laborbedingungen wurde der bereits in der Arbeit von Raoul Axinte[38] aus dem Jahr 2009 beschriebene Versuch der kontinuierlichen Synthese von 1-Ethyl-3-methylimidazolium ethylsulfat ([EMIM]$EtOSO_3$) aus 1-Methylimidazol (MIM) und Diethylsulfat (DES) durchgeführt. In einem ersten Versuch wurden analog der Vorschrift mittels zweier Spritzenpumpen die beiden Edukte äquimolar jeweils bei einer Flussrate von 40 µL/min in einem T-Stück vereint und der Start der exothermen Reaktion in der Kapillare[4] beobachtet. Die Termika Fotilo wurde nach empirischer Ermittlung des maximalen Temperaturanstiegs in einem Abstand von ca. 7 cm vom T-Stück (≡ einer Verweilzeit von 15 Sekunden), an dem die beiden Edukte zusammentreffen, auf die Kapillare fokussiert. Die Reaktion wurde nach 25 cm Kapillarlänge (≡ einer Verweilzeit von 95 Sekunden) in einem Becherglas mit Wasser gequencht, das entstandene Produkt jedoch weder aufbereitet noch analysiert, da der Umsatz für die Einsatzprüfung der Termika Fotilo nicht von Belang war.

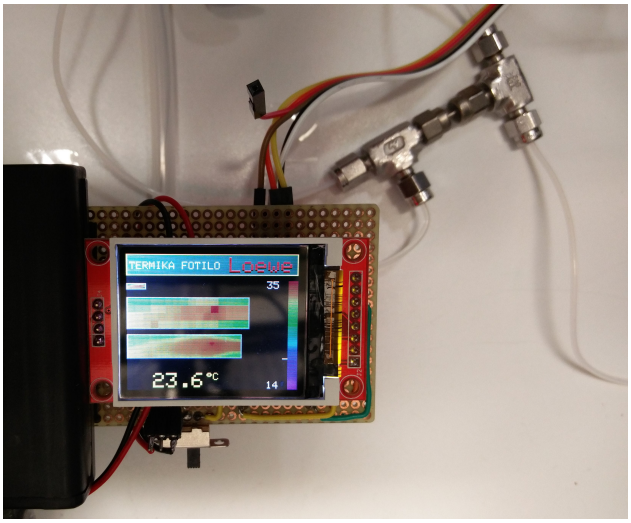


Abbildung 2.5: Momentaufnahme der Temperatur bei der kontinuierlichen Synthese von [EMIM]$EtOSO_3$
(Erläuterungen zum Display siehe Abbildung 2.6)

Im weiteren Verlauf wurde das Reaktionsgemisch zusätzlich über ein T-Stück mit Einspeisung von FC-40® (Flussrate: 100 µL/min) in einzelne Tropfen segmentiert, um die Wärmeentwicklung in jedem einzelnen Tropfen beobachten zu können. Darüber hinaus diente die kontinuierliche Phase aus FC-40® der Kühlung der Kapillare, um eine Fehlinterpretation der Messdaten durch einen Temperaturgradienten der Kapillare über die Laufzeit

[4] Verwendet wurde eine 1/16" Kapillare mit einem Innendurchmesser von 800 µm aus FEP, welches eine besonders gute IR-Durchlässigkeit im Vergleich zu Kapillaren aus PTFE aufweist.

der Reaktion zu vermeiden. Durch den Einbau des zweiten T-Stücks erweiterte sich das Verweilelement in Form der Kapillare um weitere 7 cm, wodurch unter Berücksichtigung der gesteigerten summierten Flussrate der Abstand des maximalen Temperaturanstiegs erneut bei ca. 7 cm Distanz zu letzten T-Stück lag.

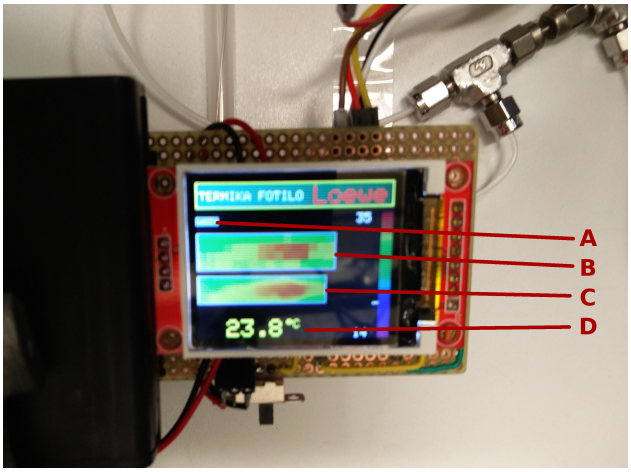


Abbildung 2.6: Momentaufnahme der Temperatur bei der segmentierten kontinuierlichen Synthese von [EMIM]EtOSO$_3$
A Temperaturarray in Originalgröße (4x16 Pixel)
B Temperaturarray vergrößert (16x64 Pixel)
C Temperaturarray interpoliert (13x61 Pixel)
D aktuelle Umgebungstemperatur

Die Daten der Messungen wurden direkt über einen Seriell-zu-USB-Wandler an einen Computer übertragen. Bei den übertragenen Daten handelt es sich je Messung um ein Array aus 4 mal 16 Temperaturen, die, bei korrekter Fokussierung der Termika Fotilo auf die Kapillare, dem Temperaturdurchschnitt von 200x200 µm großen Flächen, somit also einer messbaren Kapillarlänge von 3,2 µm entsprechen. In Abbildung 2.7 ist der Rohdatensatz einer Messung, umgewandelt in Farbwerte von grün (≡ kalt) bis rot (≡ warm) zu sehen, welcher mittels Interpolation[5] der Temperatur- und damit auch Farbwerte auf ein Array aus 13 mal 61 Temperaturen erweitert wurde. Hierdurch wird der zuvor nur schlecht erkennbare Temperaturverlauf innerhalb der Kapillare unter Verlust der Bildschärfe deutlich.

[5]Unter Interpolation versteht man in der Bildbearbeitung die Erweiterung der Anzahl der vorhandenen Pixel durch Berechnung von Farbwerten möglicher weiterer Pixel zwischen zwei benachbarten Pixeln. Diese Interpolation geht immer mit dem Verlust von Bildschärfeinformationen einher, erlaubt jedoch die bessere Visualisierung digital abgebildeter Objekte.

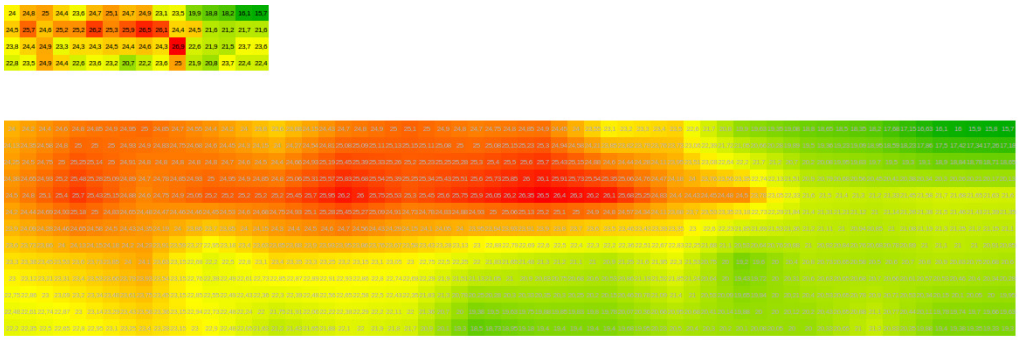
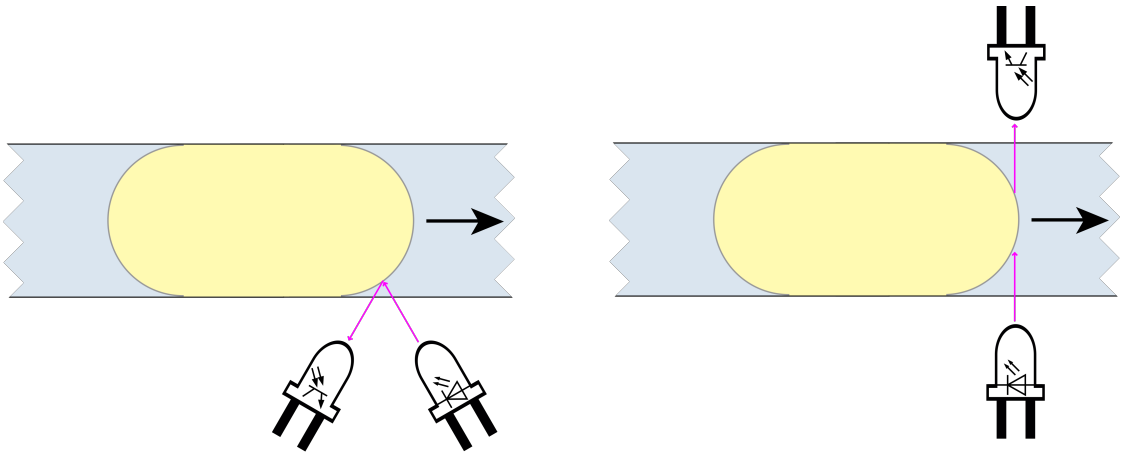


Abbildung 2.7: Visualisierung der gemessenen Temperaturwerte (Originalwerte, 4x16 Werte, oben, Vergleich Abbildung 2.6, **A**) durch Interpolation (Interpoliertes Bild, 13x61 Werte, unten, Vergleich Abbildung 2.6, **C**)

Der Einsatz von Pyrometern zur Temperaturmessung im kontinuierlichen Fluss erlaubt eine berührungslose Temperaturbestimmung in Echtzeit für einen definierten Bereich des Systems. So können nicht nur Temperaturunterschiede innerhalb der Kapillare, beispielsweise zwischen dem Zentrum dieser und wandnahen Bereichen sichtbar gemacht werden, sondern auch der Reaktionsverlauf visualisiert werden. Handelt es sich wie bei der hier gezeigten Synthese von [EMIM]$EtOSO_3$ um eine stark exotherme Reaktion, kann beispielsweise der Reaktionsbeginn durch den sichtbaren Temperaturanstieg im Tropfen genau bestimmt werden. Unter Verwendung von Infrarot-Sensor-Chips mit höherer Auflösung lassen sich wahlweise größere Bereiche des Reaktionssystems abbilden oder noch detailliertere Messungen der Temperatur eines Kapillarabschnitts machen, ohne das Reaktionssystem innerlich verändern zu müssen.

2.2 Optoelektronische Sensoren

Neben der digitalen Erfassung von Temperaturen ist, insbesondere bei tropfenbasierten Reaktionen, die Überwachung des Durchflusses mittels Lichtschranken eine einfache und berührungslose Methode. Eine Tropfenerkennung im Fluss erlaubt nicht nur die Messung der Länge eines jeden Tropfens, sondern daraus resultierend auch die Berechnung seines Volumens. Sind Flussrate und Kapillarlänge bekannt, so lässt sich jeder Tropfen als eigenständiges Reaktionssystem über die gesamte Reaktionsdauer behandeln. Denkbare Anwendungen sind gepluste elektrochemische Reaktionen oder Photoinitiierungsreaktionen mit gepulstem Licht.



(a) Aufbau einer Reflexionslichtschranke (b) Aufbau einer Transmissionslichtschranke

Abbildung 2.8: Vereinfachter Aufbau von Reflexions- und Transsmissionslichtschranken

Im Folgenden werden zwei unterschiedliche Messmethoden der Tropfenerkennung mittels Lichtschranke vorgestellt: die Reflexionsmessung und die Transmissionsmessung einer Lichtquelle an einer Kapillare.
Die Funktionsweise von Transmissionslichtschranken und Reflexionslichtschranken ist zwar grundlegend identisch, jedoch sind die Anwendungsbereiche beider Lichtschrankentypen unterschiedlich: Reflexionslichtschranken haben durch die gewinkelte Anordnung von LED und Phototransistor eine eng begrenzte Fokusdistanz, sehen also beispielsweise nicht sehr weit in eine Kapillare hinein. Für sie ist dadurch aber die Dicke des Körpers nur nebensächlich, sofern die zu detektierenden Tropfen innerhalb der Fokusdistanz die Lichtschranke passieren. Die Transmissionslichtschranke ist hingegen dazu in der Lage, selbst über größere Abstände von LED und Phototransistor Transmissionsunterschiede zu erkennen, wobei mit steigender Distanz die Auflösung der Lichtschranke abnimmt. Ein weiterer wichtiger Unterschied beider Lichtschranken ist der bei der Transmissionslichtschranke bauartbedingte größere Lichteinfallkegel als bei der winkligen Reflexionslichtschranke. Durch Verwendung von Fokuslinsen oder sogar Laserlichtquellen kann dieser Nachteil bei Transmissionslichtschranken aufgehoben werden.

2.2.1 Reflexionslichtschranken

2.2.1.1 Technische Realisierung - Tropfenzähler

Der Aufbau einer Reflexionslichtschranke ist denkbar einfach: wie in Abbildung 2.8 (a) dargestellt, besteht diese aus zwei Bauteilen, einer LED, die in die Kapillare[6] Licht einstrahlt und einem Phototransistor[7], dessen elektrischer Widerstand von der Intensität des in der Kapillare am Tropfen reflektierten Lichts abhängt. Dieser Spannungswert wird dann durch einen Mikrocontroller gemessen und interpretiert.

Fertige Reflexionslichtschranken mit LED und Phototransistor, die in einem Gehäuse mit festem Winkel fixiert sind, sind bis hinunter zu Fokusdistanzen von 0,7 mm erhältlich[8]. Produktiv verwendet wurden die Reflexlichtschranken Everlight ITR9904[9] (Abbildung 2.9), welche eine Fokusdistanz von 3,0 mm haben. Durch diese breite Fokusdistanz lässt sich selbst beim Einsatz von 1/8" Kapillaren der Fokus der Lichtschranke problemlos auf die Mitte der Kapillare justieren.

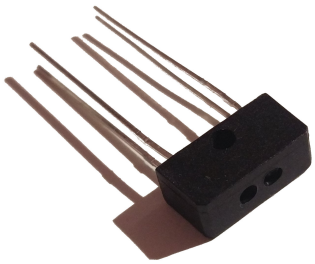


Abbildung 2.9: Everlight ITR9904[9] Kompaktlichtschranke

Außer der Software und dem zum Anschluss der Lichtschranke an den Mikrocontroller notwendigen Schaltplan wurde eine spezielle Halterung (Abbildung 2.11 (a)) entwickelt, die einerseits die Flexibilität der Fokussierung der Lichtschranke auf den Tropfen in der Kapillare erlaubt, aber dennoch einen Abschnitt der Kapillare durch den blockartigen Aufbau absolut geradlinig führt und damit gut vom Umgebungslicht abschirmt, sodass es innerhalb der Lichtschranke nicht zu Fehlmessungen durch einfallendes Umgebungslicht kommt.

[6] Statt herkömmlichen Kapillaren aus PTFE kamen 1/16" FEP-Kapillaren mit einem ID von 1000 µm zum Einsatz, da durch die bessere Transparenz des Materials eine geringere Streuung des Lichts im Kapillarmantel stattfindet.

[7] An Stelle eines Phototransistors kann auch ein einfacher Photowiderstand verwendet werden.

[8] Es fanden testweise Everlight ITR8307[9] Verwendung, welche eine Fokusdistanz von 0,7 mm ab Gehäusekante aufweisen, siehe auch der untere Teil der Messzelle in Abbildung 2.11 (a).

[9] Datenblätter des ITR8307 und ITR9904 siehe Anhang B

Der Anschluss der Lichtschranke an den Mikrocontroller findet direkt statt, kann aber auch, beispielsweise, wenn ein Mikrocontroller mit einem geringeren Spannungsbereich[10] Verwendung findet, über einen Wandlerchip wie den MCP3008[11] oder den MCP3208[11] von Microchip Technology Inc. erfolgen. Abbildung 2.10 zeigt die Verkabelung der Lichtschranke. Es können an die noch freien analogen Eingänge des Arduino (A1-A3) noch weitere Lichtschranken angeschlossen werden. Zusätzlich zu dem Anschluss der Lichtschranke ist in Abbildung 2.10 auch der Aufbau in Verbindung mit einem zweizeiligen beleuchteten Display und beleuchtetem Reset-Knopf gezeigt (grau schraffierter Bereich). Dieser Schaltplan entspricht dem Aufbau des Tropfenzählers in Abbildung 2.11 (b).

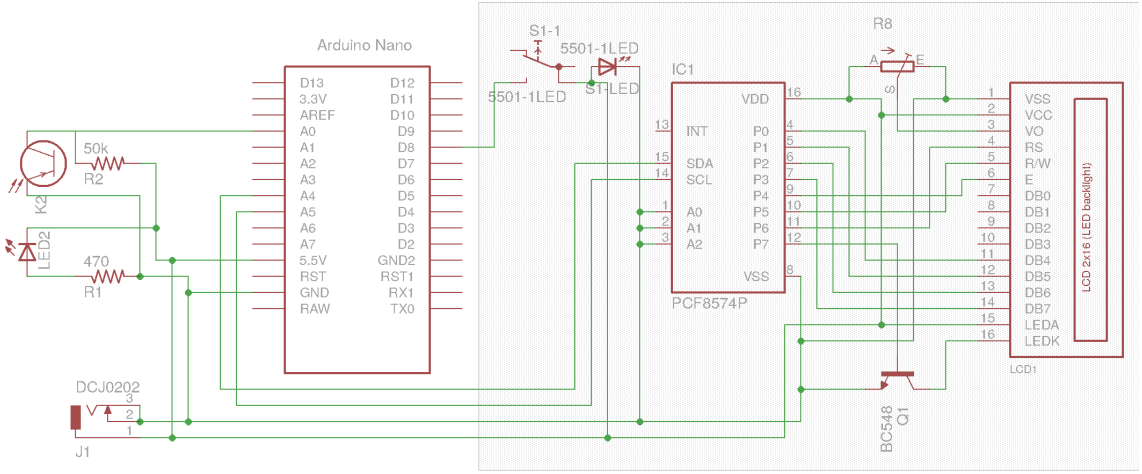
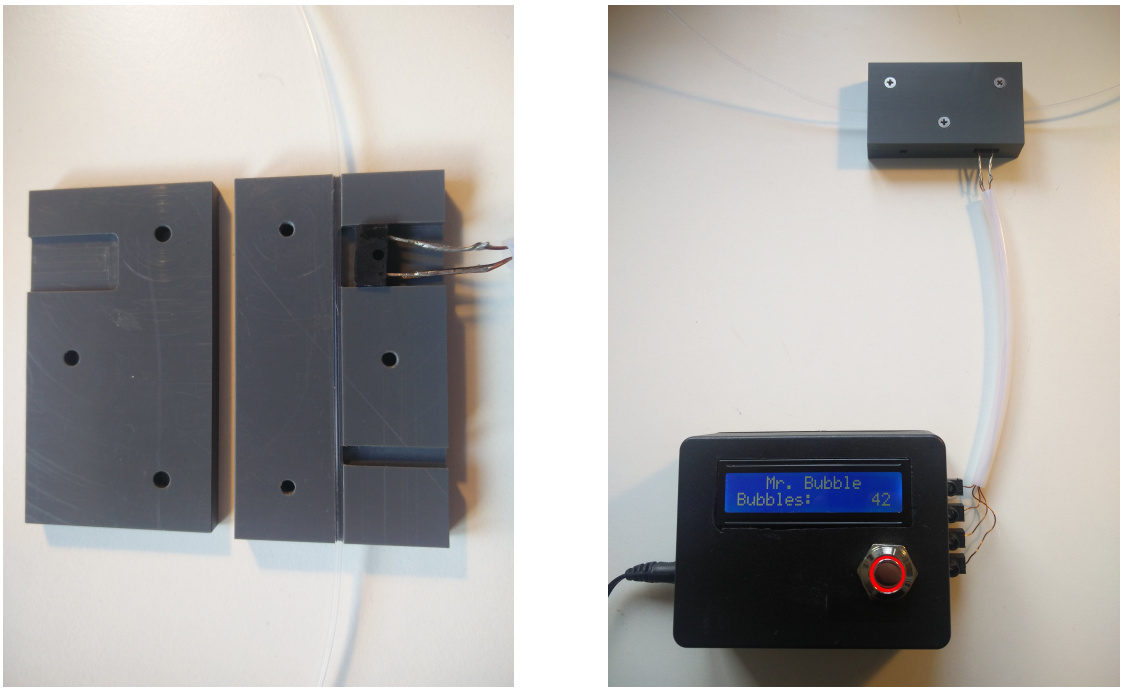


Abbildung 2.10: Schaltplan der Lichtschranke; optional angebundenes Display und beleuchteter Reset-Button (grau schraffierter Bereich)

Der Tropfenzähler erlaubt neben dem reinen Zählen der durch die Kapillare an der Lichtschranke vorbei geflossenen Tropfen auch das Triggern von Ereignissen in Echtzeit in der angeschlossenen zentralen Steuerungseinheit oder via direkt verbundenen weiteren Mikrocontrollern. Ein Ereignis wird innerhalb der Software frei gewählt, sodass schon durch einen einzelnen Tropfen ein Ereignis auslösbar ist: ein Anwendungsbeispiel hierfür ist die gepulste Elektro- oder Photochemie-Reaktion, bei der der Tropfenzähler in Echtzeit den Puls auslöst. Es besteht aber auch die Möglichkeit, ein Ereignis erst nach einer bestimmten Anzahl von Tropfen auszulösen, beispielsweise wenn eine feste Anzahl Tropfen, also ein definiertes Volumen, gesammelt werden soll und dann ein Fraktionierer (siehe Kapitel 3.3, Seite 91) auf den nächsten Sammelbehälter wechseln soll.

[10] Das Gros der Mikrocontroller arbeitet entweder mit einer Standardspannung von 5 V, beispielsweise der Arduino Uno (ATMEGA328P) oder mit einer reduzierten Spannung von 3,3 V. Vertreter dieser Klasse sind der ESP8266 oder im weiteren Sinne der Steuerungscomputer der Raspberry Pi

[11] 8-kanaliger Analog-Digital-Wandlerchip mit *SPI*-Interface und einer Auflösung von 10 Bit, 1024 Schritte (MCP3008) oder 12 Bit, 4096 Schritte (MCP3208), Datenblätter siehe Anhang B



(a) Geöffnete Messzelle mit Lichtschranke (b) Steuerungseinheit mit Display und Button

Abbildung 2.11: Aufbau des einfachen Tropfenzählers

2.2.1.2 Software (Mikrocontroller)

Die Software des Tropfenzählers ist zweigeteilt: Neben der auf dem Mikrocontroller laufenden Software zur Abfrage der Lichtschranken und Ansteuerung des Displays und des Buttons wurde zusätzlich ein Programm zur grafischen Darstellung der Messwerte in Echtzeit auf einem direkt angeschlossenen PC übertragen. Dieses Programm ist auch in der Programmiersprache *Processing*[12] geschrieben und kann sowohl für Windows, macOS, Linux und Android aus der Entwicklungsumgebung[39] heraus übersetzt werden. Auf dem Zielsystem ist zur Ausführung eine installierte Java Laufzeitumgebung erforderlich.

Die auf dem Arduino laufende Software ist im Folgenden in ihren beiden Teilen, der Initialisierungsroutine und der Wiederholungsroutine, dargestellt. Der größte Teil des Quellcodes befasst sich mit der Ansteuerung des angeschlossenen Displays zur Ausgabe der aktuellen Tropfensumme. Hierzu sind auch die im Kopfteil geladenen Standardbibliotheken `Wire.h` und `LiquidCrystal_I2C.h` notwendig, die die notwendigen Routinen zur Ansteuerung des Displays beinhalten.

[12] Die in der Entwicklungsumgebung *Processing* verwendete namensgleiche Programmiersprache ist nicht nur identisch der innerhalb der Arduino-IDE verwendeten, auf einem vereinfachten Java basierenden objektorientierten Programmiersprache, sondern wird auch, speziell für die grafische Ausgabe, Simulation oder Animation von Messdaten aus Mikrocontrollern, zur Entwicklung von Software und Apps für verschiedene Betriebssysteme genutzt.

Die wichtigsten Parameter, welche im Kopfteil gesetzt werden, sind neben der I^2C-Adresse des Displays der analoge Eingang, an dem die Lichtschranke angeschlossen ist (`analogInPin`) und die empirisch zu ermittelnde Unterscheidungsgrenze[13] (`limit`). Die hier festgelegte Unterscheidungsgrenze ist nur für die interne Tropfenzählung und Darstellung auf dem angeschlossenen Display relevant, da über die serielle Schnittstelle des Mikrocontrollers die gemessenen Rohdaten auf einen angeschlossenen Computer übertragen und von der dort laufenden Software ausgewertet werden. So kann auch die initiale Kalibrierung der Unterscheidungsgrenze über einen angeschlossenen Computer erfolgen.

Listing 2.4: Quellcodeausschnitt: Initialisierungsroutine - Lichtschranke

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2); // Set the LCD I2C address
const int analogInPin = A0;       // analog input of the lightbarrier
const int limit = 130;
int sensorValue = 0;
int outputValue = 0;
int buttonValue = 0;
int oldbuttonValue = 0;
float lastByte = 0;
long counter = 0;

void setup() {
  Serial.begin(9600);
  pinMode(8, INPUT);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("   Mr. Bubble   ");
  lcd.setCursor(0,1);
  lcd.print("Bubbles:");
}
```

In der Wiederholungsroutine der Mikrocontroller-Software wird periodisch der Widerstandswert des Phototransistors der Lichtschranke abgefragt, in einen Wert zwischen `0` und `255` umgerechnet und mit der vorher festgelegten Unterscheidungsgrenze verglichen. Liegt der Wert darüber, so befindet sich gerade ein Tropfen auf Höhe der Lichtschranke. Um nicht den selben Tropfen mehrfach zu zählen, findet direkt im Anschluss an die Tropfenerkennung ein Vergleich mit dem Messwert der vorherigen Messung statt. Wurde zuletzt auch ein Tropfen erkannt, löst die aktuelle Messung keine erneute Zählung eines Tropfens aus. Es handelt sich um den gleichen Tropfen, der zuvor schon gezählt wurde. Wurde zuvor hingegen kein Tropfen erkannt, handelt es sich um einen noch nicht gezählten Tropfen. Der

[13] Die Unterscheidungsgrenze hängt von der korrekten Fokussierung der Lichtschranke, der Transparenz der Kapillare, der Lichtbrechung des verwendeten Laufmittels und der Brechung des Tropfens ab.

Zähler wird um eins erhöht. Anschließend wird der Messwert lokal gespeichert, um bei der folgenden Messung als Vergleichswert zu dienen. Nun folgt die Übertragung des Messwerts über die serielle Schnittstelle und die Ausgabe des Zählerstands auf dem Display. Wird im Betrieb der Reset-Button gedrückt, wird innerhalb einer entprellenden Funktion[14] der Zählerstand zurückgesetzt und der Wert auf dem Display genullt.
Neben der hier gezeigten Ausgabe via serieller Schnittstelle und auf dem Display ist auch eine direkte Übertragung über einen digitalen Ausgang des Arduino ohne großen Aufwand umsetzbar, sodass die Erkennung eines Tropfens in Echtzeit an einen anderen Mikrocontroller gemeldet wird und dieser daraufhin beispielsweise ein Ventil schaltet.

Listing 2.5: Quellcodeausschnitt: Wiederholungsroutine - Lichtschranke

```
1  void loop() {
2    sensorValue = analogRead(analogInPin);
3    buttonValue = digitalRead(8);
4    outputValue = map(sensorValue, 0, 1023, 0, 255);
5    if (buttonValue != oldbuttonValue) {      // Define when to reset
6      oldbuttonValue = buttonValue;
7      delay(10);
8      counter = 0;
9      lcd.setCursor(8, 1);
10     lcd.print("        ");
11   }
12   if (outputValue > limit) {                // Define when to count
13     if (lastByte < limit) {
14       counter++;
15     }
16   }
17   lastByte = outputValue;
18   Serial.println(sensorValue);
19   if(counter > 999999999)
20     lcd.setCursor(6, 1);
21   else if(counter > 99999999)
22     lcd.setCursor(7, 1);
23   else if(counter > 9999999)
24     lcd.setCursor(8, 1);
25   else if(counter > 999999)
26     lcd.setCursor(9, 1);
27   else if(counter > 99999)
```

[14]Unter Prellen versteht man bei elektromechanischen Schaltern das ungewollte erneute Auslösen des Schalters während des Drückens: Statt des einmaligen Auslösens eines Ereignisses in der Software beim Betätigen des mechanischen Schalters wird das Ereignis mehrfach gestartet, als ob ein sogenanntes elastisches Zurückprallen vorläge. Programmiertechnisch kann die Abfrage des Zustands eines Druckschalters dadurch entprellt werden, dass der Schalterzustand gespeichert wird und das wiederholende Auslösen so erkannt wird. Dies verhindert das erneute Absenden eines dadurch getriggerten Ereignisses. Einfache Entprellungsroutinen basieren auf einer kurzen Zeitverzögerung vor der erneuten Zustandsabfrage des Druckschalters. Auch die oben erläuterte Tropfenerkennung macht sich das Entprellen zu Nutze und speichert den vorher gemessenen Wert, um nicht erneut denselben Tropfen zu detektieren.

```
28      lcd.setCursor(10, 1);
29    else if(counter > 9999)
30      lcd.setCursor(11, 1);
31    else if(counter > 999)
32      lcd.setCursor(12, 1);
33    else if(counter > 99)
34      lcd.setCursor(13, 1);
35    else if(counter > 9)
36      lcd.setCursor(14, 1);
37    else
38      lcd.setCursor(15, 1);
39    lcd.print(counter);
40    delay(10);
41  }
```

2.2.1.3 Software (PC)

Der folgende Quellcode lässt sich zu einer auf Windows- und Linux-Betriebssystemen lauffähigen Software übersetzen, welche vom ersten seriellen Anschluss[15] des Rechners Daten entgegen nimmt und diese in einem dynamischen Koordinatensystem darstellt. Abbildung 2.12 zeigt die grafische Auftragung der Messwerte der Lichtschranke: die x-Achse entspricht der Zeitachse, die dynamisch weiter wandert, auf der y-Achse wird der Messwert aufgetragen. Übersteigt dieser ein vorgegebenes Limit (`limit`, Zeile 5), wird der Raum unterhalb des Messpunktes rot eingefärbt, liegt der Messwert unterhalb des Limits bleibt es bei einer grünen Einfärbung. Zugleich wird analog der Software auf dem Mikrocontroller eine Zählung der detektierten Tropfen durchgeführt.

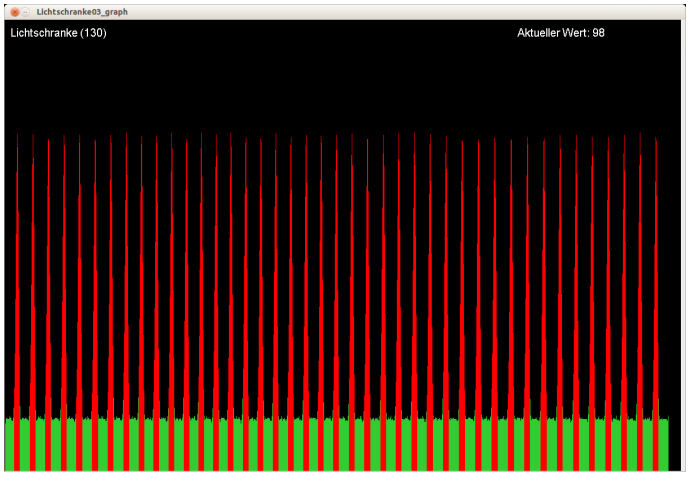


Abbildung 2.12: Screenshot der grafischen Darstellung der Lichtschrankenmesswerte am PC

[15] Ein über USB angeschlossener Arduino Mikrocontroller wird von gängigen Betriebssystemen als per USB angebundene serielle Schnittstelle erkannt.

Da sich der hier gezeigte Quellcode erweitern lässt, sind unzählige Anwendungsmöglichkeiten denkbar: neben der reinen Visualisierung der die Lichtschranke passierenden Tropfen ist beispielsweise eine Volumenberechnung der Tropfen[16], eine Fraktionierung eines vorher festgelegten Ensembles von Tropfen oder die individuelle, gepulste Behandlung eines jeden Tropfens in Echtzeit umsetzbar.

Durch die Nutzung der selben Programmiersprache ist der Quellcode der PC-Software dem der Arduino-Software sehr ähnlich: nach der Initialisierung der Startparameter und der seriellen Schnittstelle erfolgt analog der Arduino-Software die Tropfenerkennung und -zählung. Statt der Ausgabe auf einem angeschlossenen Display wird zur grafischen Ausgabe der Daten am Computer ein Bild gemalt, welches neben einem Hintergrundrechteck aus den vertikalen Linien eines jeden Messwerts, eingefärbt in Abhängigkeit vom vorher eingestellten Unterscheidungslimit, besteht. Darüber wird in der rechten oberen Ecke des Bildes noch der aktuelle Messwert eingebettet, alternativ kann hier auch die bisherige Anzahl an detektierten Tropfen ausgegeben werden.

Listing 2.6: Quellcodeausschnitt: PC-Software - Grafische Messwertauswertung label

```
1  import processing.serial.*;
2  Serial myPort;
3  int xPos = 1;
4  PFont f;
5  int limit = 130;                // Define limit for drop or not
6  float lastByte = 0;
7  int counter = 0;
8
9  void setup () {
10   size(1200, 800);              // Set the window size
11   println(Serial.list());
12   // Open whatever port is the one you're using.
13   // (Windows: [1], Linux: [0])
14   myPort = new Serial(this, Serial.list()[0], 9600);
15   myPort.bufferUntil('\n');
16   background(0);
17 }
18
19 void draw () { }
20
21 void serialEvent (Serial myPort) {
22   String inString = myPort.readStringUntil('\n');
23   if (inString != null) {
24     inString = trim(inString);
25     float inByte = float(inString);
```

[16]Das Tropfenvolumen (Ellipsoid) ist für jeden Tropfen individuell anhand der Flussrate, dem Kapillardurchmesser und der Abtastrate der Lichtschranke mittels einfacher mathematischer Formeln berechenbar.

```
26      inByte = (inByte/1.5);
27      inByte = map(inByte, 0, 1023, 0, height);
28      f = loadFont ("ArialMT-20.vlw");
29      textFont(f,20);
30      noStroke();
31      fill(0);
32      rect(width-300,0,300,80);
33      stroke(0);
34      fill(255);
35      text("Lichtschranke ("+limit+")",10,30);
36      if (counter > 10) {
37        stroke(127,50,255);
38        fill(127);
39      }
40      text("Anzahl der Ereignisse: "+counter,width-290,30);
41      stroke(0);
42      fill(255);
43      if (inByte > limit) {
44        stroke(255,0,0);
45        line(xPos, height, xPos, height - inByte);
46       if (lastByte < limit) {
47         counter++;
48        }
49      }
50      else {
51        stroke(50,205,50);
52        line(xPos, height, xPos, height - inByte);
53      }
54      lastByte = inByte;
55      if (xPos >= width) {
56        xPos = 0;
57        background(0);
58      }
59      else {
60        xPos++;
61      }
62    }
63  }
```

2.2.1.4 Doppelemulsionsdetektion

Neben der bisher vorgestellten Tropfendetektion mittels einer einzelnen Lichtschranke, wurde im Rahmen dieser Arbeit auch die Detektion komplexer Tropfensysteme unter Verwendung von mehreren Lichtschranken durchgeführt. Hierzu entspricht sowohl der Grundaufbau, als auch Teile der Software der zuvor erläuterten Umsetzung. Es wurde eine zu Abbildung 2.11 (a) vergleichbare Halterung mit der Aufnahmemöglichkeit für bis zu 3 Lichtschranken in unterschiedlicher Ausrichtung angefertigt, wie Abbildung 2.13 zeigt.

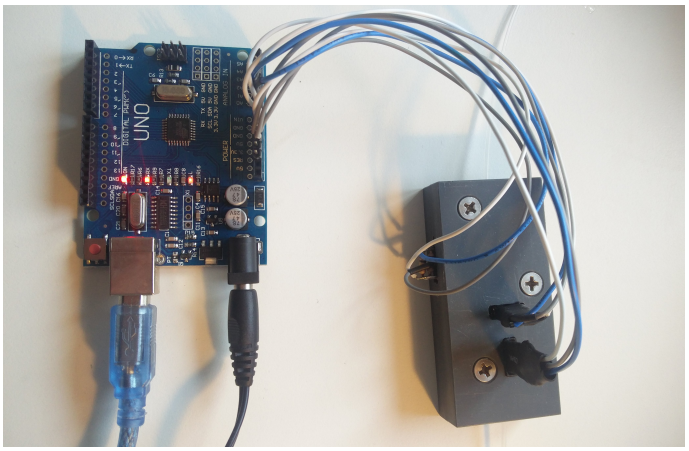


Abbildung 2.13: Messzelle mit 3 unterschiedlich angeordneten Lichtschranken (seitlich, längs und quer zur Kapillare) und dem Arduino Mikrocontroller

Die Anordnung der Lichtschranken ist bei der hier gezeigten Messzelle bewusst so gewählt, dass alle Lichtschranken einen anderen Blick auf die Kapillare haben. Dies diente der Untersuchung, ob ein anderer Messwinkel[17] das Messergebnis nachvollziehbar beeinflusst. Wie erwartet liefert nur die um 90° gedrehte Anordnung der Lichtschranke (quer der Kapillare) zu den anderen beiden Lichtschranken unterschiedliche Messwerte. Die Messwerte dieser gedrehten Lichtschranke liefern zwar auch sichtbare Peaks, jedoch ist deren Intensität weitaus geringer, als die der längs angeordneten Lichtschranken (Vergleich auch in Abbildung 2.15 (b), die gelbe Linie der Messwerte der quer angeordneten Lichtschranke). Dies lässt sich auf die zusätzliche Streuung des Lichts durch die gekrümmte Kapillaroberfläche bei der um 90° gedrehten Lichtschranke zurückführen (Abbildung 2.14), was in einer stark verringerten messbaren Lichtintensität resultiert.

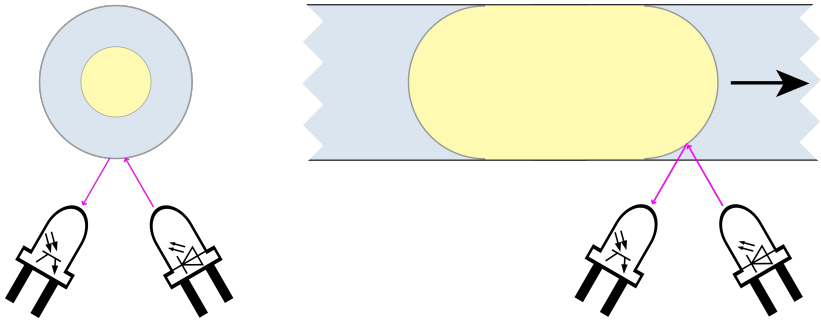
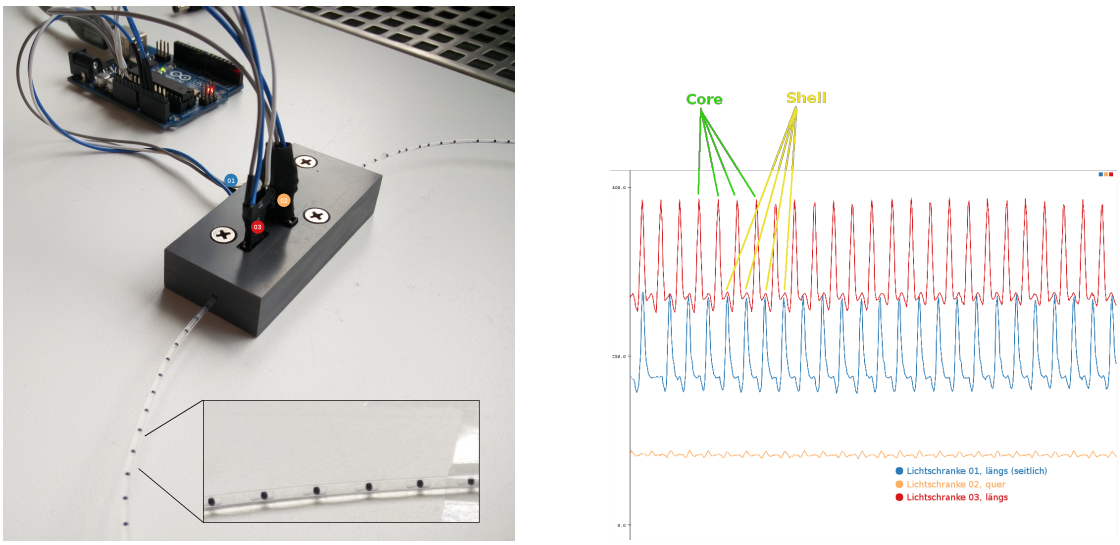


Abbildung 2.14: Unterschied der Messanordnung der Lichtschranken: quer oder längs zur Kapillare

[17] Wie in Abbildung 2.13 sichtbar, ist eine Lichtschranke seitlich längs der Kapillare, zwei Lichtschranken mit Aufsicht auf die Kapillare angeordnet, wovon eine längs der Kapillare und eine quer der Kapillare eingebaut ist.

Trotzdem liefern alle 3 Lichtschranken bei korrekter Fokussierung die gewünschten Ergebnisse: Ein Messaufbau zur Ermittlung der Verhältnisse zwischen dem Kerntropfen (Core) und dem ihn umgebenden Tropfen (Shell) in Doppelemulsionen ist in Abbildung 2.15 (a) gezeigt. Die Messergebnisse (Abbildung 2.15 (b)) zeigen eindeutig die Erkennung der Doppelemulsionen durch die Lichtschranken. Die Abbildung zeigt auch, dass die erkannten Tropfen nicht zwingend unterschiedliche Farbe oder Transparenz aufweisen müssen - selbst zwei farblose Flüssigkeiten, wie der in diesem Versuch verwendete Toluoltropfen in FC-40® als kontinuierlicher Phase ist, wenn auch mit weitaus geringerer Intensität als der mit Tinte blau angefärbte Kerntropfen, deutlich in der Auftragung der Messwerte über die Zeit sichtbar.



(a) Versuchsaufbau mit drei Lichtschranken

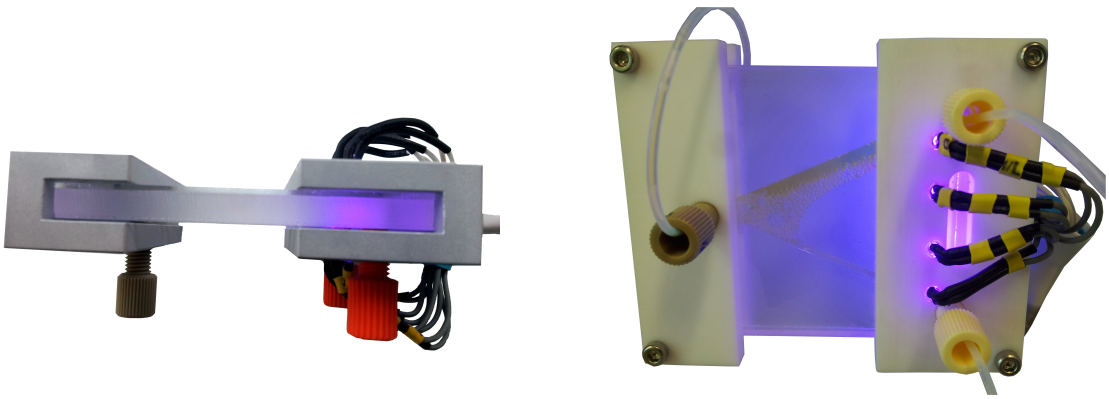
(b) Messwerte der drei Lichtschranken

Abbildung 2.15: Messung von Core-Shell-Doppelemulsionen, Kerntropfen besteht aus mit Tinte angefärbtem Wasser, der Hüllentropfen aus Toluol und die kontinuierliche Phase aus FC-40®, Verschiebung der Messwerte bedingt durch die unterschiedliche absolute Position der Lichtschranken an der Kapillare

Die Nutzung von Reflexionslichtschranken in Kombination mit Mikrocontrollern ermöglicht die intelligente und nicht-invasive Detektion von Tropfen in Kapillaren. Findet mehr als eine Lichtschranke Verwendung, lässt sich durch geschickte Fokussierung dieser nicht nur der einzelne Tropfen erkennen, sondern auch Systeme mit mehreren ineinander verschachtelten Tropfen einzeln aufgelöst betrachten. Somit ist jeder Tropfen (Shell) mit seinem Kerntropfen (Core) individuell und zerstörungsfrei vermessbar.

2.2.2 Transmissionslichtschranken

Auch der Aufbau einer Transmissionslichtschranke ist so intuitiv, wie der der Reflexionslichtschranken: wie Abbildung 2.8 (b), Seite 21, zeigt, ist die Kapillare oder der betrachtete Volumenkörper zwischen der LED und dem Phototransistor[18], dessen elektrischer Widerstand von der Intensität des in der Kapillare am Tropfen reflektierten Lichts abhängt, positioniert. Analog der Reflexionslichtschranke wird der an dem Phototransistor anliegende Spannungswert durch einen Mikrocontroller gemessen und interpretiert.



(a) Aufsicht (b) Seitliche Ansicht

Abbildung 2.16: Beispiel für die Nutzung von Transmissionslichtschranken: die Phasengrenzflächenerkennung in der Phasentrennkammer des autonomen Phasentrenners, Details dazu siehe Kapitel 3.2

Durch den größeren Lichteinfallwinkel stellt die Krümmung der Kapillaroberfläche, vergleichbar der quer verbauten Reflexionslichtschranke, (siehe Abbildung 2.14, vorheriger Abschnitt) ein großes Problem bei der Detektion von Tropfen in Kapillaren dar. Die Brechung und Streuung des Lichts im Kapillarmantel sorgt für eine Erhöhung der gemessenen Grundintensität, sodass die Auflösung der Detektion bei Transmissionslichtschranken in dünnen Kapillaren stark begrenzt bis nahezu unbrauchbar ist.

Ihre Stärken spielt die Verwendung von Transmissionslichtschranken bei flachen Glaskörpern aus, in denen beispielsweise die Veränderung der Phasengrenzfläche detektiert werden soll. So fanden Transmissionslichtschranken bei der Entwicklung des autonomen Phasentrenners, Kapitel 3.2, Seite 65 ff, Verwendung. Auch hier wird die Intensitätsveränderung durch einen Arduino-Mikrocontroller gemessen und dient der Bestimmung der Lage und Veränderung der Phasengrenze innerhalb der Trennkammer des Phasentrenners.

[18] An Stelle eines Phototransistors kann auch hier ein einfacher Photowiderstand verwendet werden.

Kapitel 3

Aktoren

Unter Aktoren versteht man das signalwandlerbezogene Gegenstück zu den Sensoren: ein Aktor wandelt ein elektrisches Signal in eine physikalische Größe, beispielsweise die Bewegung eines Motors, die Veränderung von Druck, Temperatur oder des elektrischen Potentials. Der in der Mikrofluidik wahrscheinlich bedeutendste Aktor ist die Pumpe, die mit definierter Flussrate eine Reaktion mit Edukt(en) speist. Darüber hinaus zählen auch Heizbäder, Kryostaten, Ventile, Potentiostaten und Galvanostaten zu den häufig eingesetzten Aktoren innerhalb der Mikrofluidik. Im Gegensatz zu den Heizbädern und Kryostaten handelt es sich bei Potentiostaten und Galvanostaten um inkludierte Aktoren, deren Elektroden direkt mit dem Reaktionsgemisch in Kontakt stehen müssen. Ventile gelten zwar auch als inkludierte Aktoren, sorgen aber im Reaktionsverlauf nur für die Dosierung und Reaktionswegeleitung und sind nicht Teil der eigentlichen Reaktion.

Dieses Kapitel beleuchtet die Entwicklung eines Potentiostaten, eines Fraktionssammlers und die Weiterentwicklung des bereits in [40] vorgestellten autonomen Phasentrenners. Alle drei Gerätschaften basieren auf dem Arduino Mikrocontroller und können bzw. müssen zur Steuerung und Überwachung über die USB-Schnittstelle mit einem Computer verbunden werden.

3.1 Potentiostat

Die präparative Elektrochemie hat in den vergangenen Jahren eine Renaissance erlebt und an vielen Stellen Einzug in das Standardrepertoire beim Reaktionsdesign in der organischen Chemie erhalten. Neben der klassischen Elektrochemie in einer einfachen oder geteilten elektrochemischen Zelle, teilweise auch in einem Rundkolben versehen mit Elektroden, spielt auch im kontinuierlichen Fluss die Elektrochemie eine immer größere Rolle.

Die Umsetzung von elektrochemischen Reaktionen in mikrofluidischen Systemen bietet durch die auf Grund der Miniaturisierung kleinen Elektrodenabstände die Möglichkeit

sehr hoher Stromdichten und geringer kapazitiver Effekte zwischen den Elektroden. Um die an den Elektroden die gewünschte Spannung einzustellen, wird ein elektronischer Regelverstärker, ein sogenannter Potentiostat, benötigt. Dieser stellt den Strom zwischen der zu regulierenden Arbeitselektrode und der Gegenelektrode so ein, dass das gewünschte Potential erreicht wird. Dies wird über die Referenzelektrode, deren Potential aus der elektrochemischen Spannungsreihe bekannt ist, als Referenzpunkt ermöglicht. Dazu ist es notwendig, dass durch die Referenzelektrode selbst kein Strom fließt.

In diesem Teil des Kapitels wird die Herstellung einer einfachen und doppelten elektrochemischen Durchflusszelle, Versuche zu verschiedenen chemischen Reaktionen mit eben diesen Durchflusszellen (kontinuierlich und tropfenbasiert) und die Entwicklung eines auf dem Arduino Mikrocontroller basierenden Potentiostaten beschrieben.

3.1.1 Entwicklung einer elektrochemischen Durchflusszelle

Neben der in einem späteren Abschnitt dieses Kapitels beschriebenen Entwicklung eines einfachen, auf dem Arduino Mikrocontroller basierenden Potentiostaten, bedurfte es auch dem Bau einer elektrochemischen Durchflusszelle, der Schnittstelle des Mikroreaktionssystems mit dem Potentiostaten.

Bei der Auswahl der Bauart dieser Durchflusszelle standen zwei grundverschiedene Varianten zur Auswahl. Eine Möglichkeit stellt der kontinuierliche Fluss der Reaktionslösung zwischen zwei vergleichbar einem Plattenkondensator angeordneten Elektroden rechts und links an der Kanalwand dar. Dieser Typ Durchflusszelle mit langen, in Flussrichtung angeordneten Elektroden, hat nicht nur den Nachteil eines über die Länge abfallenden und damit nicht konstanten Elektrodenpotentials, sondern ist in dieser Form darüber hinaus nur bedingt für Reaktionen in tropfenbasierten Systemen geeignet, da der gleichzeitige Kontakt zweier Tropfen zwischen den Elektroden zu keiner reproduzierbaren und einzeln mess- und auswertbaren Reaktion innerhalb der Tropfen führt. Bereits dadurch, dass der in Flussrichtung erste Tropfen die Elektroden früher erreicht hat, sorgt für eine -wenn auch geringfügig- andere Konzentration der Edukte innerhalb des ersten Tropfens im Vergleich zum zweiten Tropfen, da schon Produkt gebildet wurde. Alleine durch diese Verschiedenartigkeit der Tropfen und das auf die Elektrodenlänge abfallende Elektrodenpotential lässt sich keine verlässliche Aussage mehr über die Verteilung des Stromflusses zwischen erstem und zweitem Tropfen bei gleichzeitiger Elektrodenberührung machen.
Die alternative Möglichkeit ist die Nutzung von Mikorelektroden[1], welche entweder ver-

[1] In diesem Kapitel wird die aus der Biotechnologie bekannte Bezeichnung *Mikroelektrode* genutzt, obwohl es sich dort um Elektroden mit Durchmessern von bis zu 0,1 µm, also korrekterweise Nanoelektroden, zur Messung elektrischer Potentialdifferenzen in lebenden Zellen handelt. Die hier verwendeten Platin-Drähte haben einen Durchmesser von 300 µm und tragen somit den Namen der Mikroelektrode zurecht.

gleichbar den länglichen Elektroden rechts und links des Kanals angeordnet sind oder, wie im Folgenden beschrieben, quer zur Flussrichtung im Kanal stehen.
Den Vorteilen bei der Verwendung von Mikroelektroden, wie die Nachweisbarkeit sehr kleiner Konzentrationen bei potentiostatischen Messungen, die vernachlässigbar kleinen kapazitiven Effekte und die Realisierbarkeit sehr hoher Stromdichten, steht der Nachteil des sehr kleinen Gesamtstromes gegenüber[41]. Die Ausbeute in elektrochemischen Reaktionen, deren Umsatz gesamtstromabhängig ist, fällt dementsprechend gering aus. Abhilfe schafft hier die Reihenschaltung mehrerer solcher Durchflusszellen, die, bei Einhaltung des mindestens Doppelten des Abstandes, den die Elektroden einer Zelle zueinander haben, auch keine Einflüsse unter den Durchflusszellen ausüben. So kann die Reihenschaltung von mehreren Durchflusszellen neben den tropfenbasierten Reaktionen auch im kontinuierlichen Fluss ohne Einschränkung Anwendung finden. Ein Beispiel für den Bau einer doppelten Durchflusszelle ist Teil der folgenden Beschreibungen.

3.1.1.1 Aufbau

Der Grundaufbau einer elektrochemischen Durchflusszelle setzt sich aus einem Flusskanal, welcher aus einer herkömmlichen Kapillare bestehen kann, und drei Elektroden, im einfachsten Falle drei dünne Platindrähte zusammen. Die drei Drähte müssen einen kleineren Durchmesser als den Innendurchmesser der Kapillare aufweisen, um ein Umfließen der Elektroden zu ermöglichen. Darüber hinausgehend ist der korrekte Abstand der Elektroden zueinander wichtig, um gerade bei tropfenbasierten Reaktionen die Position des Tropfens unter Berührung aller Elektroden sicherzustellen, da andernfalls kein Strom fließt und somit keine Reaktion stattfindet. Die in die Kapillare eingebrachten Platinelektroden müssen an der Eintritts- und Austrittsstelle dicht mit der Kapillarwand abschließen bzw. muss durch Verkleben das Auslaufen von Reaktionslösung verhindert werden.

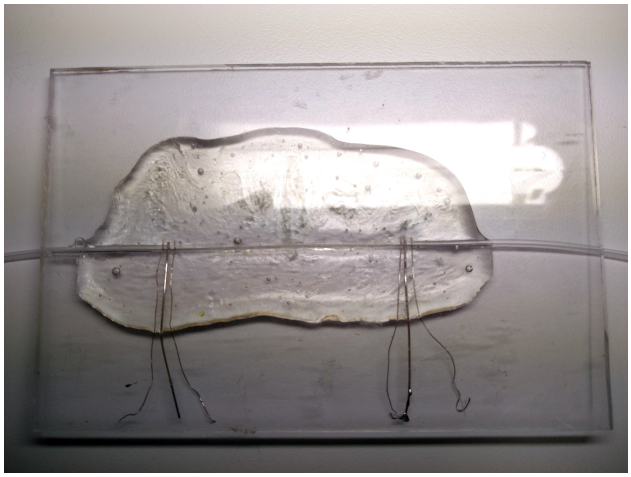


Abbildung 3.1: Prototyp einer doppelten Durchflusszelle: Kapillare 1/16" OD, 800 µm ID; Elektrode 1, 3, 4, 6: 300 µm Platindraht, Elektrode 2 und 5: 500 µm Silberdraht; in Epoxydharz auf PMMA eingegossen

Der in Abbildung 3.1 gezeigte erste Prototyp einer doppelten Durchflusszelle wurde noch unter der Verwendung von Silberdraht als jeweils dritter (mittlerer) Elektrode zur späteren Herstellung einer Silberchloridoberfläche und Nutzung als Ag/AgCl-Referenzelektrode

gebaut. Die Herstellung einer Ag/AgCl-Elektrode aus einem einfachen Silberdraht direkt in der Zelle stellte sich jedoch als nicht umsetzbar dar. In weiteren darauf folgend hergestellten Durchflusszellen wurde daher entweder die Möglichkeit geschaffen, die Referenzelektrode durch einen Schraubverschluss wechselbar zu gestalten oder ganz auf die Verwendung eines Standardpotentials an der Referenzelektrode verzichtet, indem wie an Anode und Kathode ein Platindraht eingebaut wurde. Zusätzlich wurde auch von der Möglichkeit Gebrauch gemacht, die Anode oder Kathode als Referenzelektrode einzusetzen (2-Elektrodensystem), was einen Potential von 0 V gegenüber der Arbeitselektrode entspricht.

(a) Vollansicht (b) Schnitt

Abbildung 3.2: 3D-Konstruktion einer einfachen Durchflusszelle mit schraubbarer Referenzelektrode und schraubbarer Abdichtung von Anode und Kathode, siehe Text

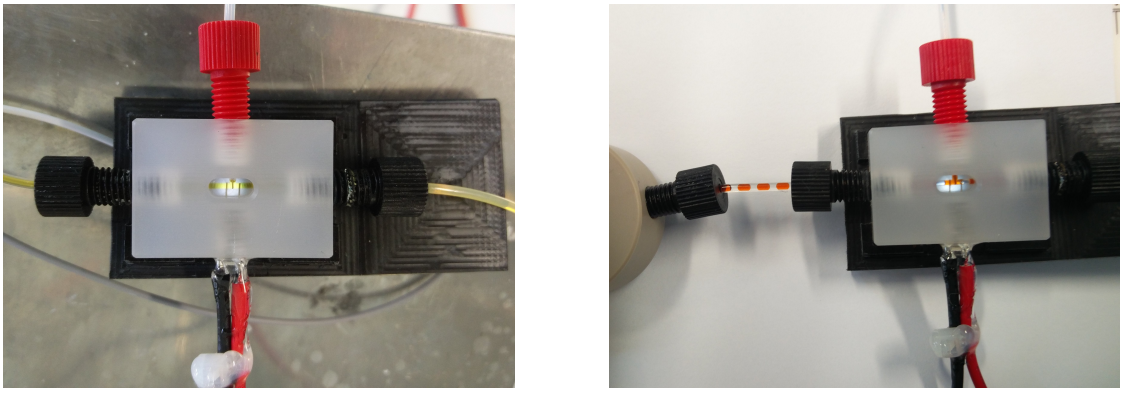
(a) Vollansicht (b) Schnitt

Abbildung 3.3: 3D-Konstruktion einer doppelten Durchflusszelle für je drei Platinelektroden, siehe Text

Die Durchflusszellen wurden aus PCTFE[2] mit einem Kanalinnendurchmesser von 800 µm und einem Elektrodenabstand von 850 µm (Abstand Anode zu Kathode somit 2 mm) gefertigt. Die Bohrungen zur Aufnahme der Elektroden haben einen Bohrdurchmesser von 300 µm. Der Mittenabstand zwischen den beiden Zellen der doppelten Durchflusszelle be-

[2]Polychlortrifluorethylen (PCTFE) ist ein chemisch sehr beständiger Kunststoff, der unter den Fluorkunststoffen die höchste Härte und Festigkeit aufweist und gerade im Miniaturbereich gut fräs- und bohrbar ist. Der Handelsname lautet Kel-F®, Datenblatt siehe Anhang B.

trägt 4 cm, um eine leichtere Handhabbarkeit im Laboralltag zu gewährleisten. Hier sind auch deutlich kleinere Abstände (mind. 4 mm) der Zellen zueinander möglich. Die Durchflusszellen sind an beiden Enden mit Gewindeanschlüssen für handelsübliche Fittinge 1/16" (Gewinde 1/4" -28) ausgestattet, um diese leicht in mikrofluidische Systeme zu integrieren. Beim Einsatz in tropfenbasierten Systemen muss auf einen gratfreien Übergang zwischen Fitting und Durchflusszelle geachtet werden, um eine Beeinflussung der Tropfen bzw. der kontinuierlichen Phase zu vermeiden. Wenn ein in Kapitel 2.2.1.1 beschriebener Tropfenzähler der Durchflusszelle vorgeschaltet ist, so muss auf das exakte Ausmessen der Distanz zwischen Tropfenzähler und Tropfenmittenposition in der Durchflusszelle geachtet werden, um eine gepulste elektrochemische Messung oder Reaktion durchführen zu können.



(a) Kontinuierlicher Fluss (b) Tropfenbasierte Reaktion

Abbildung 3.4: Einfache Durchflusszelle mit Platinelektroden und Hintergrundbeleuchtung bei der potentiostatischen Messung der Reduktion von Ferrocen (unterschiedliche Farben auf Grund unterschiedlicher Ferrocen-Konzentrationen)

Nach der Ausformung des PCTFE-Körpers in einem gekühlten Fräs- und Bohrprozess[3] und dem anschließenden Entgraten werden die knickfreien Platindrähte in die Bohrlöcher eingeführt und kurz vor dem Erreichen der Endposition mit einem kleinen Tropfen Epoxydharz[4] versehen, um eine Abdichtung der Drahteinführung zu gewährleisten. An die drei Enden der Platindrähte wurde nach Aushärtung des Epoxydharzes vorsichtig jeweils ein Kabel zum Anschluss der Elektrode an einen Potentiostaten angelötet. Die in Abbildung 3.4 gezeigte Durchflusszelle verfügt über eine schraubbare Zuführung der mittleren Elektrode, um den Einbau einer separaten Mikroreferenzelektrode[5] zu ermöglichen. Al-

[3]Es ist wichtig, dass das PCTFE-Material nur bei ausreichender Kühlung mit geringen Fräs- und Bohrgeschwindigkeiten hergestellt wird, da eine lokale Überhitzung des Materials zu Haarrissen führt und die Druckstabilität der Durchflusszelle beeinträchtigt.

[4]Es wurde das Epoxydharz *EPOT5.S25* der Firma *Gluetec*, Handelsname *WIKO 2-K-Epoxy Kleber 5 min*, in der Zusammensetzung von 92% Bisphenol-A-Epichlorhydrinharz (M_W>700) und 8% Bisphenol F-Epoxidharz verwendet, Datenblatt siehe Anhang B.

[5]Beispielsweise eine Silber/Silberchlorid-Mikroreferenzelektrode mit $\varnothing = 2$ mm von *Sensolytics*

ternativ kann auch ein durch eine 300 µm-Bohrung in einem Blind-Fitting eingeführter Platindraht als Referenzelektrode dienen.

3.1.1.2 Experimente

Die im vorherigen Abschnitt beschriebenen Durchflusszellen wurden mit Hilfe eines einfachen Versuchs auf Funktion geprüft: die elektrochemische Oxidation von Ferrocen zum Ferriciniumkation ($E^0_{NHE} = 0,4\,V$). Hierzu wurden mit dem in Kapitel 1.2 auf Seite 4 vorgestellten und in Abbildung 3.5 gezeigten Segmentgenerator kontinuierlich Ferrocentropfen (80 µL min^{-1}), bestehend aus in Dichlorethan gelöstem Ferrocen unterschiedlicher Konzentration versetzt mit dem Leitsalz Tetrabutylammoniumhydrogensulfat (1 mol L^{-1}), in einer kontinuierlichen, nicht-leitenden Phase aus FC-40® (30 µL min^{-1}) erzeugt. Diese wurden in der unmittelbar verbundenen Durchflusszelle potentiostatisch gemessen.

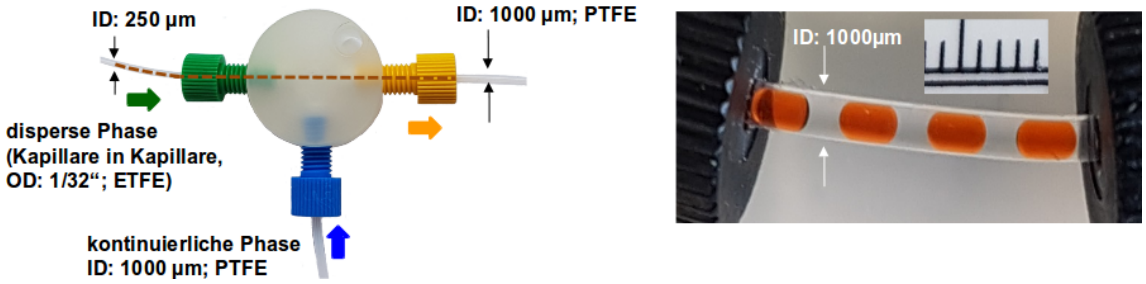


Abbildung 3.5: Kapillare-in-Kapillare Segmentgenerator (siehe Kapitel 1.2, Seite 4) zur Herstellung von Ferrocen-Tropfen in einer kontinuierlichen FC-40®-Phase

In einem ersten Messaufbau wurde ein Potential von +500 mV (gegen Platin) an der Arbeitselektrode (Abk. *WE* für working electrode) angelegt und jeweils Versuchsreihen für die Ferrocenkonzentrationen von 10 µmol L^{-1}, 100 µmol L^{-1}, 1,0 mmol L^{-1} und 10 mmol L^{-1} gemessen. Dies entspricht bei einem ungefähren Tropfenvolumen von 2,3 µL absoluten Ferrocenmengen zwischen 4,3 ng (10 µmol L^{-1}) und 4,3 µg (10 mmol L^{-1}). Wie in der Auftragung in Abbildung 3.8 zu sehen, sind selbst die geringsten hier verwendeten Ferrocenkonzentrationen deutlich messbar.

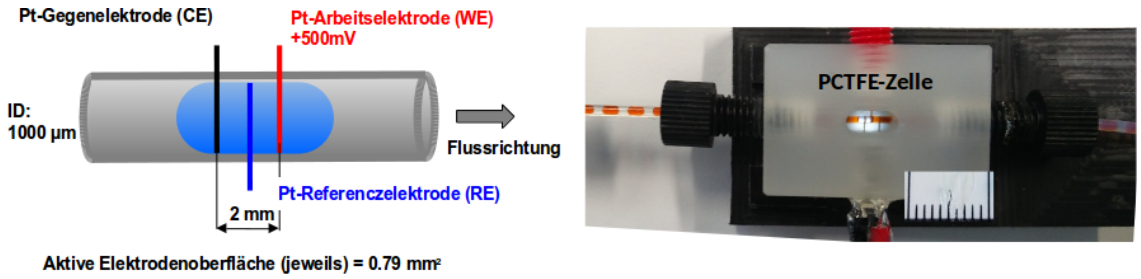


Abbildung 3.6: Aufbau der Durchflusszelle mit Elektrodenanordnung

Am Beispiel der potentiostatischen Messung einer Ferrocenkonzentration von 1,0 mmol L^{-1} (Ferrocentropfen (80 µL min^{-1}), bestehend aus in Dichlorethan gelöstem Ferrocen versetzt mit dem Leitsalz Tetrabutylammoniumhydrogensulfat (1 mol L^{-1}), in einer kontinuierlichen, nicht-leitenden Phase aus FC-40® (30 µL min^{-1})) und der daraus resultierenden Auftragung der Stromstärke gegen die Zeit (Abbildung 3.7) lässt sich der charakteristische Stromstärkeverlauf drei Ereignissen zuordnen: Die kurze Stromstärkeerhöhung bei der ersten elektrischen Verbindung der beiden Elektroden durch den leitfähigen Tropfen, verursacht durch eben diesen Einschaltstrom, verbunden mit der umgehend beginnenden Elektrodenbelegung. Dieser Stromstärkeverlauf ist bei höheren Konzentrationen (Vergleich Abbildung 3.8) nicht erkennbar, da es bei höheren Konzentrationen zu erhöhten Elektrodenbelegungen kommt und diese den Peak des Einschaltstromes stark dämpfen.

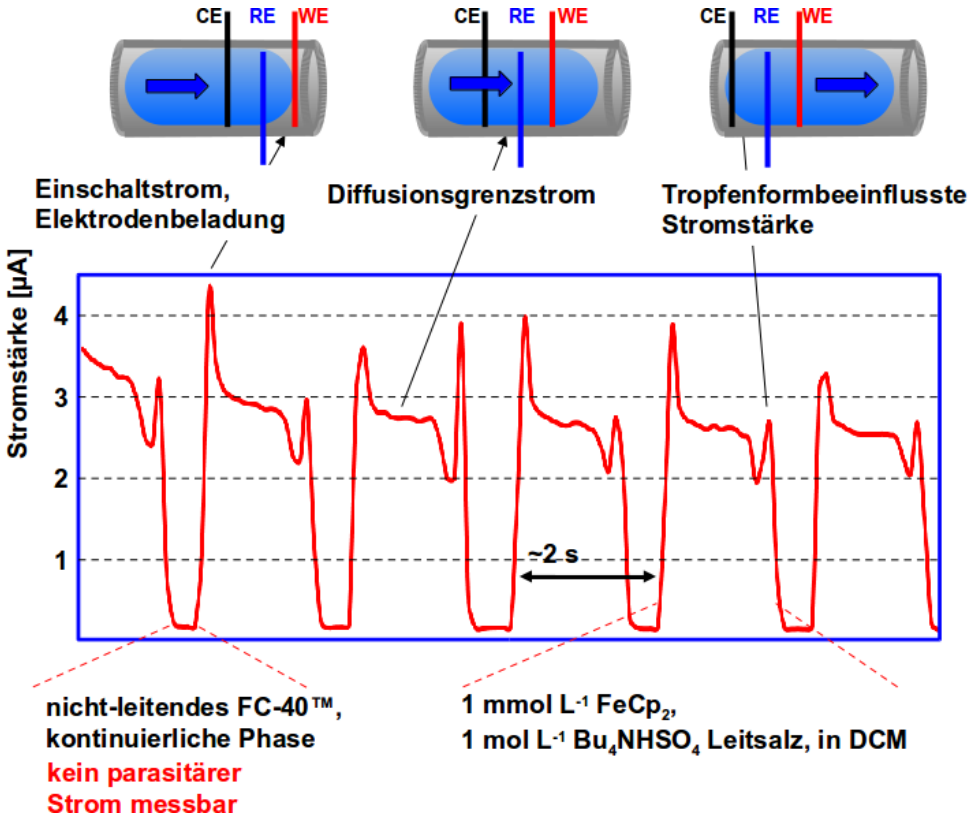


Abbildung 3.7: Zeitlicher Stromstärkeverlauf der tropfenbasierten Umsetzung von Ferrocen (1 mmol L^{-1}) zum Ferriciniumkation mit einem Potential von +500 mV

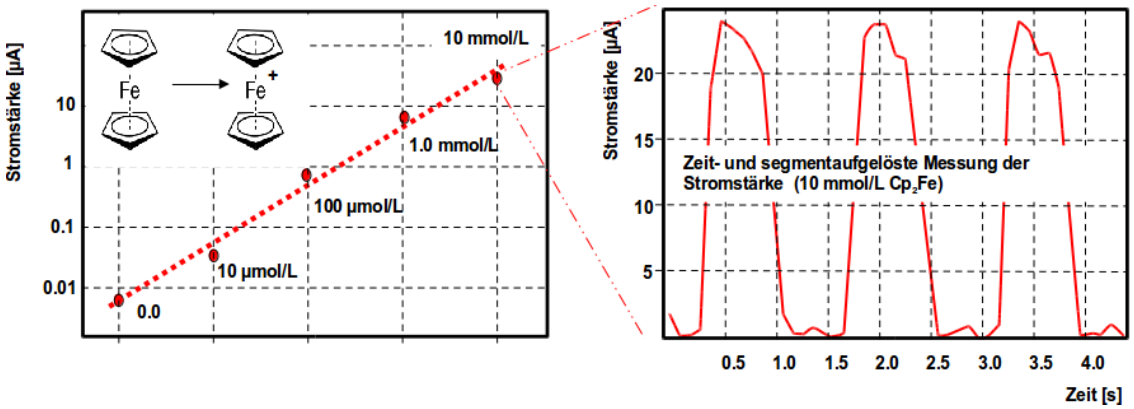


Abbildung 3.8: Auftragung der gemessenen Stromstärke bei der Umsetzung von Ferrocen in DCE für unterschiedliche Konzentrationen bei einem Potential von +500 mV

Auf den kurzen Peak des Einschaltstromes folgt dann ein Bereich konstanter Stromstärke, bedingt durch die Umsetzung des Ferrocens mit anschließendem Abfall nach vollständiger Umsetzung der elektrodennahen Ferrocenmoleküle[6], welcher am Ende des Tropfens, bedingt durch die Tropfenform, erneut einen kurzen Anstieg erfährt. Erwartungsgemäß bricht die Stromstärke zwischen zwei Tropfen ein, da in der kontinuierlichen Phase das nicht-leitende FC-40® Verwendung findet. Durch den Zusatz eines fluorierten Leitsalzes zur kontinuierlichen Phase kann das Auftreten des Einschaltimpluses zu Beginn der Messung eines jeden Tropfens verhindert werden.

In einem nächsten Versuchsschritt wurde der Einfluss der Elektrodenanordnung auf das Messresultat geprüft. Hierzu wurde erneut die kontinuierliche potentiostatische Messung von Tröpfchen mit einer Ferrocenkonzentration von 1,0 mmol L^{-1} neben der in allen anderen Versuchen verwendeten Reihung der Elektroden *Gegenelektrode - Referenzelektrode - Arbeitselektrode* (Abbildung 3.7) diesmal in umgekehrter Reihung *Arbeitselektrode - Referenzelektrode - Gegenelektrode* (Abbildung 3.9) durchgeführt. Der Peak durch den Einschaltstrom bleibt nahezu vollständig aus, da durch Verunreinigungen der kontinuierlichen Phase in der hier vorgestellten Messung der Stromfluss zwischen zwei Tropfen nicht unterbrochen wurde. Dennoch zeigt sich ein im Vergleich zu Abbildung 3.7 anderer Verlauf, da durch die Kontaktherstellung zwischen Arbeitselektrode und Referenzelektrode bereits ein Stromfluss einsetzt und durch diese im Vorfeld gestartete Elektrodenbelegung der Arbeitselektrode der Stromfluss während der gesamten Messung gehemmt wird.

[6] Aufgrund der sehr kurzen Verweildauer der Tropfen in der elektrochemischen Durchflusszelle ist die Diffusion der Ferrocenmoleküle in dieser Zeit vernachlässigbar. Eine weitere extern induzierte Durchmischung findet nicht statt, sodass nur elektrodennahe Ferrocenmoleküle zum Ferriciniumkation umgesetzt werden.

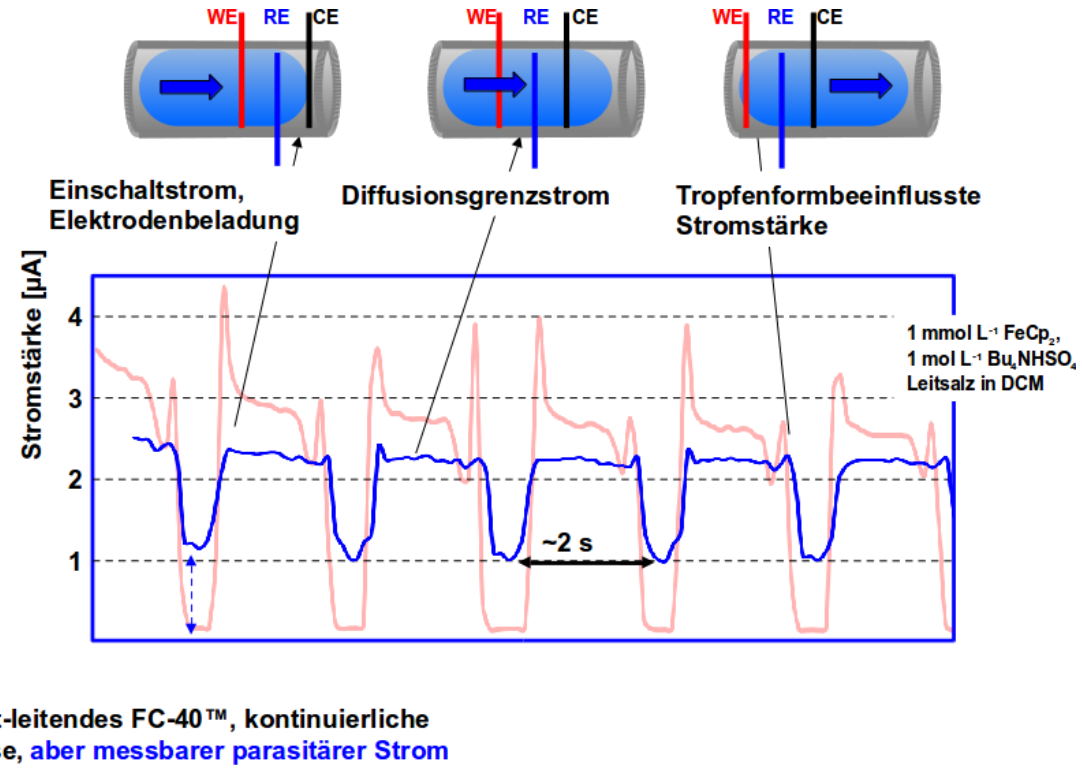


Abbildung 3.9: Zeitlicher Stromstärkeverlauf der tropfenbasierten Umsetzung von Ferrocen zum Ferriciniumkation mit einem Potential von +500 mV bei Vertauschung von Gegenelektrode und Arbeitselektrode

Durch die Erweiterung des Versuchsaufbaus um eine weitere Durchflusszelle konnte die bereits aus dem zeitlichen Verlauf der Stromstärke bei der Umsetzung des Ferrocens gemachte Annahme, dass nur ein sehr geringer Umsatz eben von den elektrodennahen Molekülen stattfindet, bestätigt werden: In der zweiten Durchflusszelle wurde versucht, die zuvor hergestellten Ferriciniumkationen wieder zum Ferrocen umzusetzen.

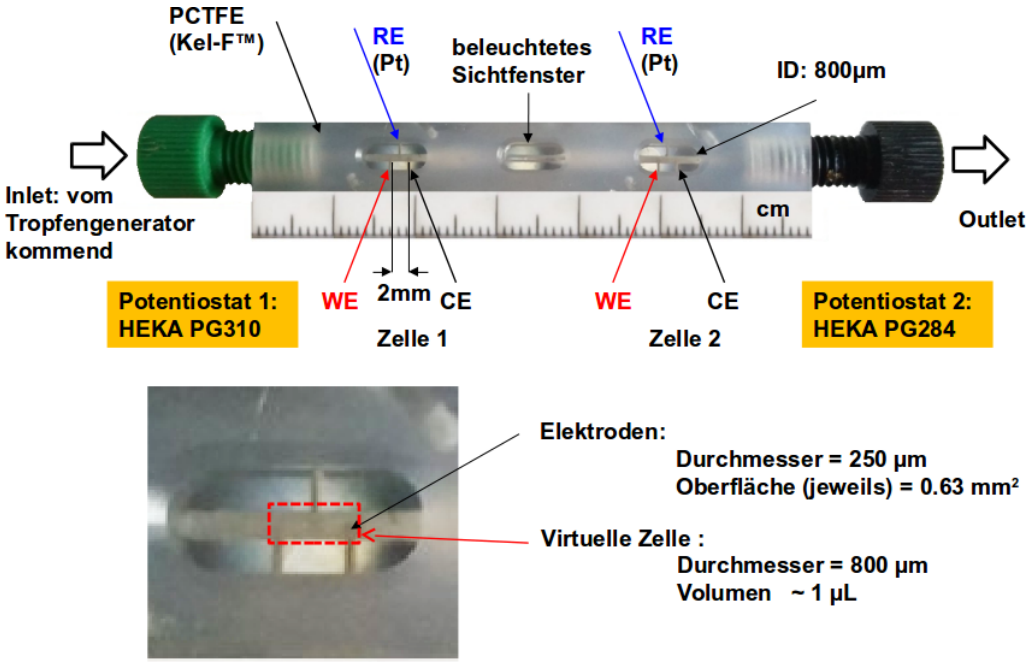


Abbildung 3.10: Aufbau zur potentiostatischen Messung der Hin- und Rückreaktion in einer doppelten Durchflusszelle

Abbildung 3.11 zeigt den zeitlichen Verlauf der Stromstärke im Rahmen der Rückreaktion. Im Gegensatz zu den bisherigen Diagrammen handelt es sich um Stromstärken im Nanoampere-Bereich. Die einzelnen Tropfen sind zwar erkennbar, jedoch ist die Rückreaktion verschwindend gering. Dies lässt sich nicht nur dadurch erklären, dass zuvor eine geringe Umsetzung von Ferrocen zum Ferrociniumkation stattgefunden hat, sondern auch verbunden mit der Distanz zwischen den beiden Durchflusszellen von über 30 mm (dem 15-fachen Elektrodenabstand bzw. der 15-fachen Tropfengröße, siehe Abbildung 3.10), welche Zeit zur Diffusion innerhalb des Tropfens erlaubt. Darüber hinaus sorgen die im Kanal befindlichen Elektroden beider Durchflusszellen für Turbulenzen innerhalb des Tropfens nach Verlassen des Bereichs der Elektroden in der ersten Zelle und beim ersten Kontakt mit den Elektroden der zweiten Zelle. So ist das zuvor erzeugte Ferrociniumkation homogen im gesamten Tropfen verteilt und steht nur zu einem sehr geringen Teil zur Rückreaktion zur Verfügung. Ein Test von zuvor im Batch-Versuch hergestellem Ferrociniumkation in entsprechend höherer Konzentration liefert das erwartete Ergebnis im Mikroampere-Bereich analog der Hinreaktion.

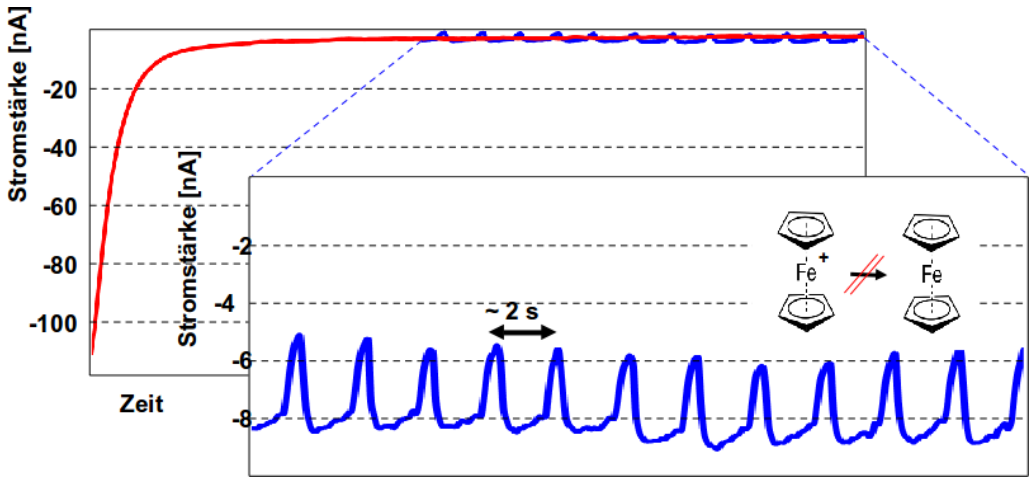


Abbildung 3.11: Messung der entsprechenden Rückreaktion der Umsetzung von Ferrocen zum Ferriciniumkation (siehe Text)

Die einfachen Messreihen zur elektrochemischen Umsetzung von Ferrocen zum Ferriciniumkation zeigen die uneingeschränkte Funktionalität der hier entwickelten kontinuierlichen Durchflusszelle für elektrochemische Mikroreaktionen. Die kurzen Verweilzeiten der einzelnen Tropfen innerhalb der Zelle bei üblichen Flussraten von mehr als 100 µL min^{-1} und die geringe Durchmischung innerhalb des Tropfens in der Zelle bedingen einen nur sehr geringen Umsatz. Für elektroindizierte chemische Reaktionen stellt dies kein Problem dar, auch nicht, wenn es um die reine cyclovoltammetrische Messung der Einzeltropfen geht. Ist das Ziel jedoch der maximale Umsatz zum gewünschten Produkt durch entsprechende Reaktion an den Elektroden, so ist durch die Serialisierung weiterer Zellen eine Steigerung des Umsatzes in jedem Tropfen möglich, was sich durch die preiswerte Herstellung und die standardisierten Anschlüsse der Zellen leicht umsetzen lässt. Da für jede Durchflusszelle ein eigener Anschluss an einem Potentiostat bzw. ein eigener Potentiostat benötigt wird, diese Geräte gerade im Laborbedarf keine preiswerte Massenware sind, musste eine kostengünstige Lösung gefunden werden, welche darüber hinaus noch die zentrale Steuerung vieler solcher Potentiostaten durch einen einzelnen Computer ermöglicht. Auf die Entwicklung dieses sogenannten Ardustat geht das folgende Kapitel ein.

3.1.2 Ardustat

Potentiostaten und Galvanostaten gehören wegen ihrer Messempfindlichkeit und ihrer Einsatzbandbreite zu den teureren Geräten in einem Elektrosyntheselabor. Aufgrund der nur in sehr engen Grenzen nutzbaren und auf veraltete Computertechnik aufsetzenden Software der im Labor zur Verfügung stehenden Potentiostaten Heka PG284 (Software für MS-DOS) und Heka PG310[7] (Software für MS Windows XP) wurde nach alternativen, kostengünstigen und (quell)offenen Potentiostatsystemen gesucht. Ein besonderes Augenmerk galt dem in den Jahren 2007 bis 2010 von Daniel A. Steingart aus dem Institut für Maschinenbau und Luftfahrttechnik der Universität Princeton entwickelten Ardustat[42, 43], einem auf dem Arduino Uno aufsetzenden, selbstentwickelten Shield, welches innerhalb der Spannungsgrenzen des Arduino Uno potentiostatische und galvanostatische Messungen ermöglicht. Durch die Verwendung der Arduino-Plattform, welche auch in vielen anderen in dieser Arbeit vorgestellten Projekten zum Aufbau des Internet of Lab zum Einsatz kommt, erschließt sich ein weiteres Anwendungsfeld unter Beibehaltung der Hardwarebasis, was die Programmierung vereinheitlicht und durch Weiterverwendung von Hardwarekomponenten Kosten reduziert.

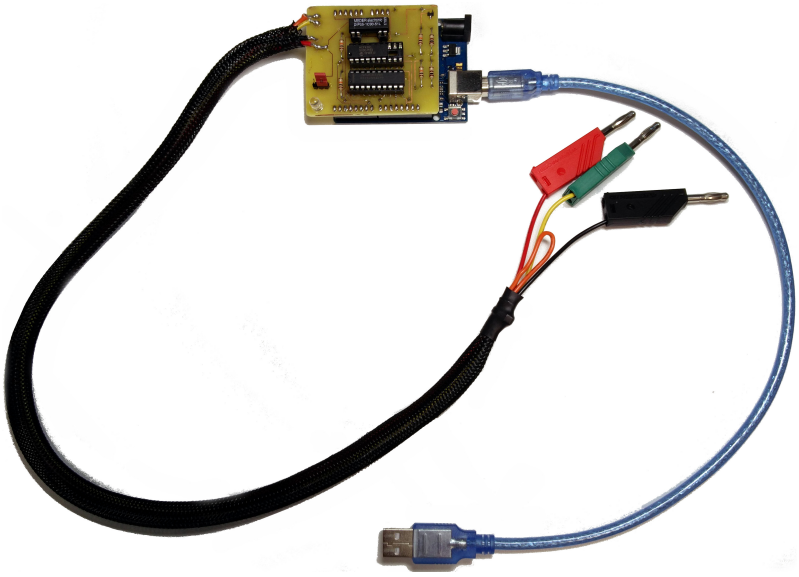


Abbildung 3.12: Gesamtansicht des Ardustat

Im Folgenden wird der Hardwareaufbau des arduino-basierten Potentiostats *Ardustat*, wie von Daniel A. Steingart ursprünglich entwickelt, vorgestellt. Zusätzlich wurde eine neue, eigene Software programmiert, die Fehler und Probleme der ursprünglichen Software, welche eine nicht öffentliche Client-Software für macOS erfordert, behebt und sich einfacher in das Internet of Lab integriert.

[7] Der Heka PG310 wurde freundlicherweise zur Verfügung gestellt vom Fraunhofer-Institut für Mikrotechnik und Mikrosysteme IMM, Mainz.

3.1.2.1 Funktionsweise

Der Ardustat entspricht einem vollwertigen Potentiostaten, welcher mit identischer Hardware auch als Galvanostat betrieben werden kann, der auf den Spannungsbereich des Arduino Uno (0 V bis +4,75 V[8]) beschränkt ist. Abhängig von der Bauteilqualität der aktiven Bauteile, wie dem digitalen Potentiometer und dem Digital-Analog-Konverter, variiert zusätzlich die Messgenauigkeit innerhalb des Spannungsbereichs. Sind diese Einschränkungen akzeptabel, erhält man dafür einen vollwertigen Potentiostaten, welcher durch seine offene Softwarestruktur leicht mittels eigener PC-Software steuerbar ist und problemlos in Gesamtsysteme wie das Internet of Lab integrierbar ist.

3.1.2.2 Technische Realisierung

Da der Mikrocontroller nicht mit negativen Spannungen arbeiten kann, aber dennoch neben der Möglichkeit im Spannungsbereich von 0 V bis +4,75 V[8] zu arbeiten auch negative Spannungen möglich sein sollen, wird für eine solche Arbeitsweise statt der Masse des Arduino als Massenreferenz der zweite Ausgang des Digital-Analog-Konverters mit einem Spannungsausgang in Höhe von +2,5 V genutzt. So sind Spannungen an den Elektroden zwischen +2,25 V (entspricht +4,75 V[8] am Arduino) und −2,5 Volt (entspricht 0 V am Arduino) erreichbar.

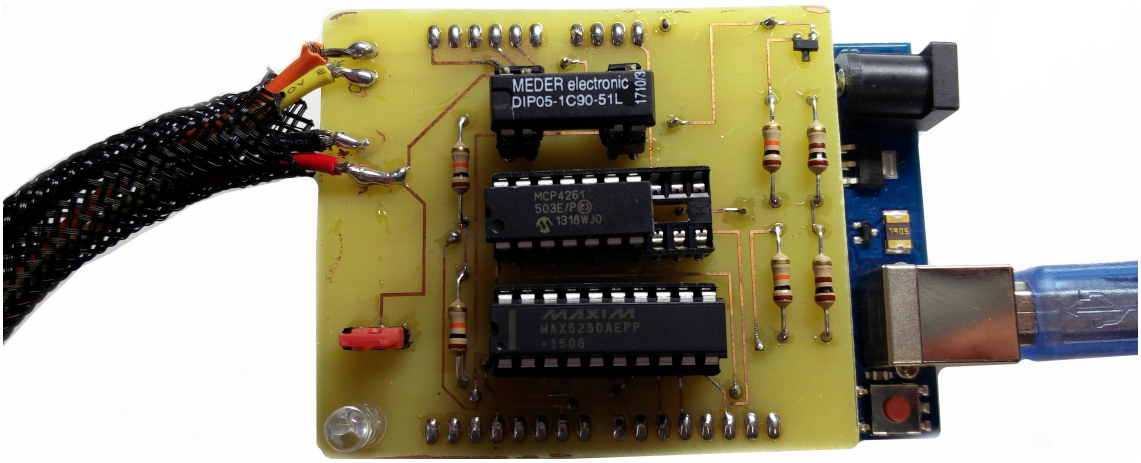


Abbildung 3.13: Bestückte Platine des Potentiostaten, vorne links der Jumper (rot) zur Moduswahl, mittig die drei ICs DIP05-1C90-51L, Meder Electronic: Relay, MCP4261, Microchip: Digitalpotentiometer und MAX5250AEPP, Maxim: Digital-Analog-Konverter

Neben der Auswahl des Betriebsmodus im Quellcode muss dieser zusätzlich durch korrektes Setzen einer Steckbrücke (Jumper) auf dem Ardustat-Shield (siehe Abbildung 3.14, *JP* oben rechts, Verbindung 1→2 = −2,5 V bis +2,25 V; Verbindung 2→3 = 0 V bis +4,75 V[8]).

[8]Der Spannungsbereich wird bewusst nicht auf das Maximum von 5 V ausgereizt, da in Abhängigkeit der Güte der eingesetzten Spannungsversorgung die Maximalspannung von 5 V am Arduino nicht konstant ist und somit etwaige Spannungsschwankungen die Messungen verfälschen würden.

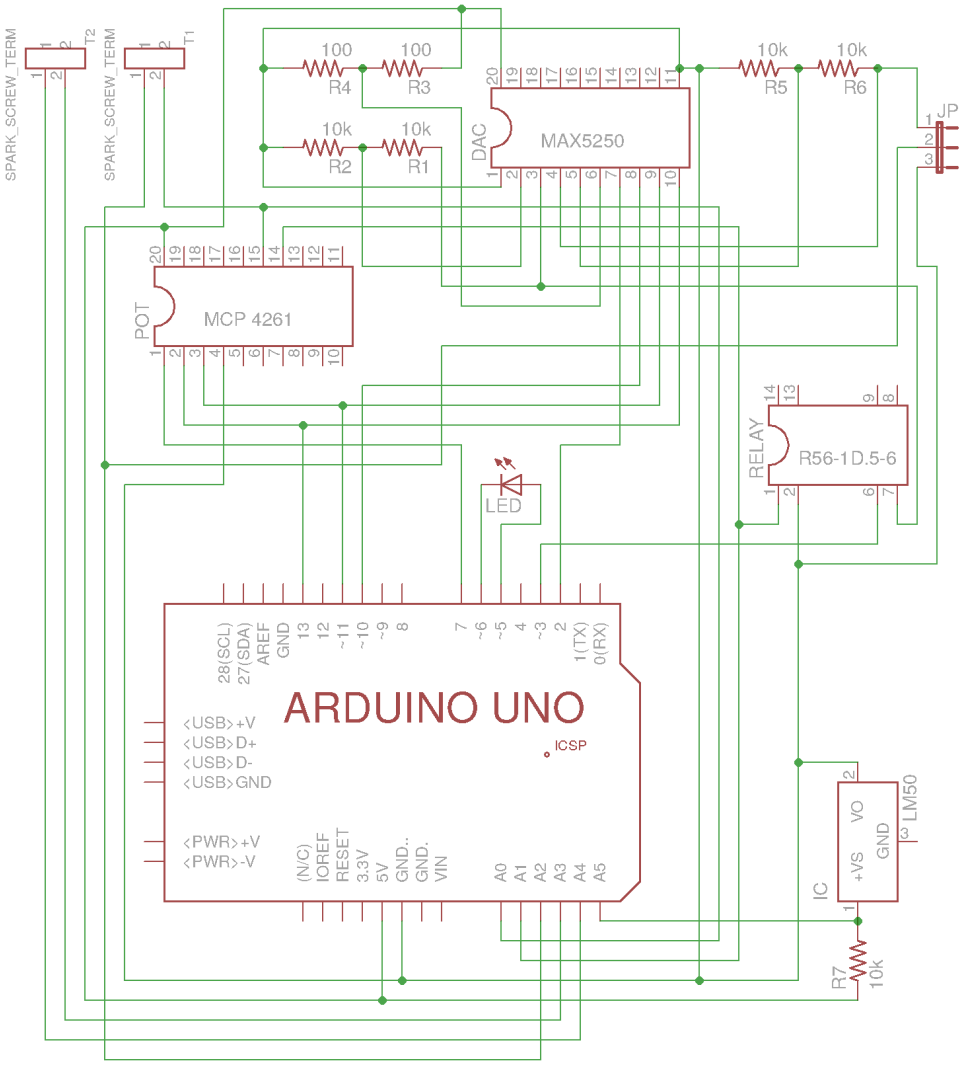


Abbildung 3.14: Schaltplan des Ardustat, IC-Datenblätter siehe Anhang B

Die doppelseitige Platine des Shields wurde entsprechend dem originalen Platinenlayout im Ätzverfahren hergestellt und mit den vorgeschlagenen Bauteilen bestückt. Der in Abbildung 3.14 gezeigte Schaltplan wurde aus den Bauteildatenblättern und dem Platinenlayout abgeleitet und zeigt die wirkliche Schaltung des mit dem Arduino Uno verbundenen Shields. Im Laufe erster Tests des Ardustat wurden Probleme mit dem verwendeten Relay (DIP05-1C90-51L, Meder Electronic) festgestellt: gerade bei hohen Messfrequenzen stört die Schaltgeschwindigkeit des Relays, sodass bei diesen Messungen das Relay aus dem Stecksockel entfernt und durch eine Drahtbrücke zwischen den Pins 1 und 7 (siehe Abbildung 3.14, Mitte rechts) des Stecksockels ersetzt wurde. Die Platine verfügt über vier Elektrodenanschlüsse, von denen zwei Anschlüsse (Kabelfarben orange und gelb, siehe Abbildung 3.13) als Messreferenz nutzbar sind und direkt mit den analogen Eingängen *A3* und *A4* des Arduino Uno verbunden sind. Das rote und das schwarze Kabel stellen erwartungsgemäß die Anode und die Kathode des Potentiostaten bereit.

3.1.2.3 Software

Die im originalen Projekt-Repository enthaltene Software zeigte nicht die gewünschte und beschriebene Funktion, sodass auf der Basis der selben Software-Bibliotheken eine vollständig neue Software geschrieben wurde.
Im Kopfteil der Software werden, wie üblich, diverse wichtige Parameter zum Programmablauf festgelegt. In der hier gezeigten Version legt man auch den Spannungsverlauf über die Messzeit im Quellcode fest. Das Beispiel in Abbildung 3.15 zeigt die Zuordnung der einzelnen Parameter zu den entsprechenden Positionen im Ablauf. Der Verlauf der beiden Rampen wird in der Software zur Laufzeit aus der Spannung zu Beginn der Rampe und am Ende eben dieser berechnet. Übersteigt die vorgegebene Anzahl der Punkte des Spannungsverlaufs die gewünschte Anzahl, so können alle weiteren Spannungen auf einen konstanten Wert gesetzt und die Zeiten auf null Sekunden reduziert werden.

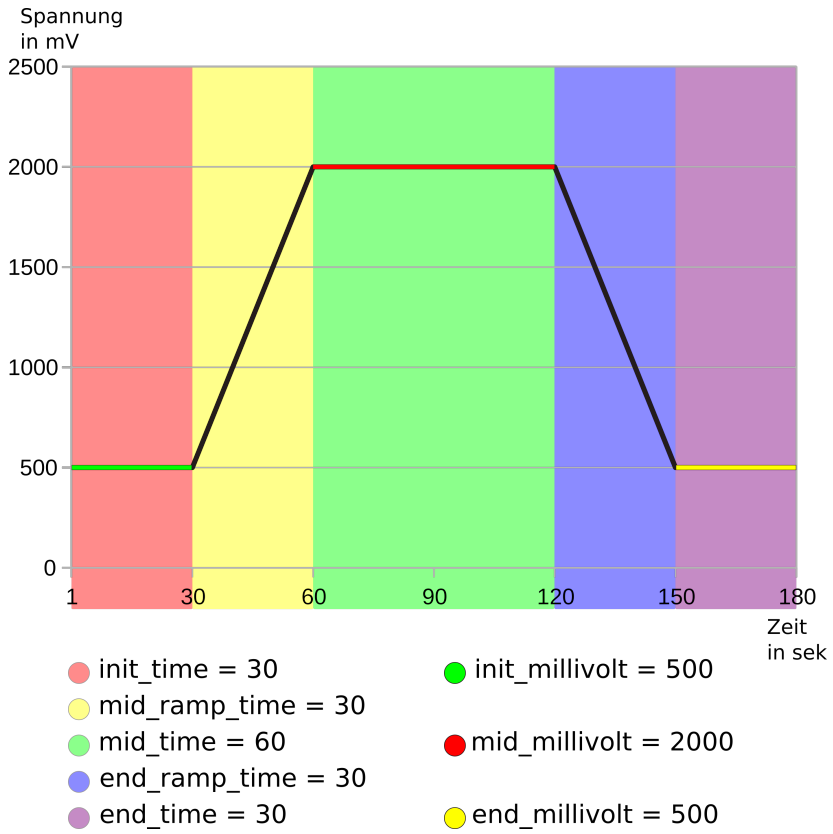


Abbildung 3.15: Beispielhafter Spannungsverlauf des Ardustat

Neben dem Spannungsverlauf stellt die Einstellung des Modus im Kopfteil des Quellcodes den wichtigsten Parameter dar. Hier ist, wie schon im Abschnitt 3.1.2.2 auf Seite 48 erklärt, die Auswahl des Spannungsbereichs entweder zwischen 0 und +4,75 V (mode = 1) oder zwischen −2,5 V und +2,25 V (mode = 2) möglich.

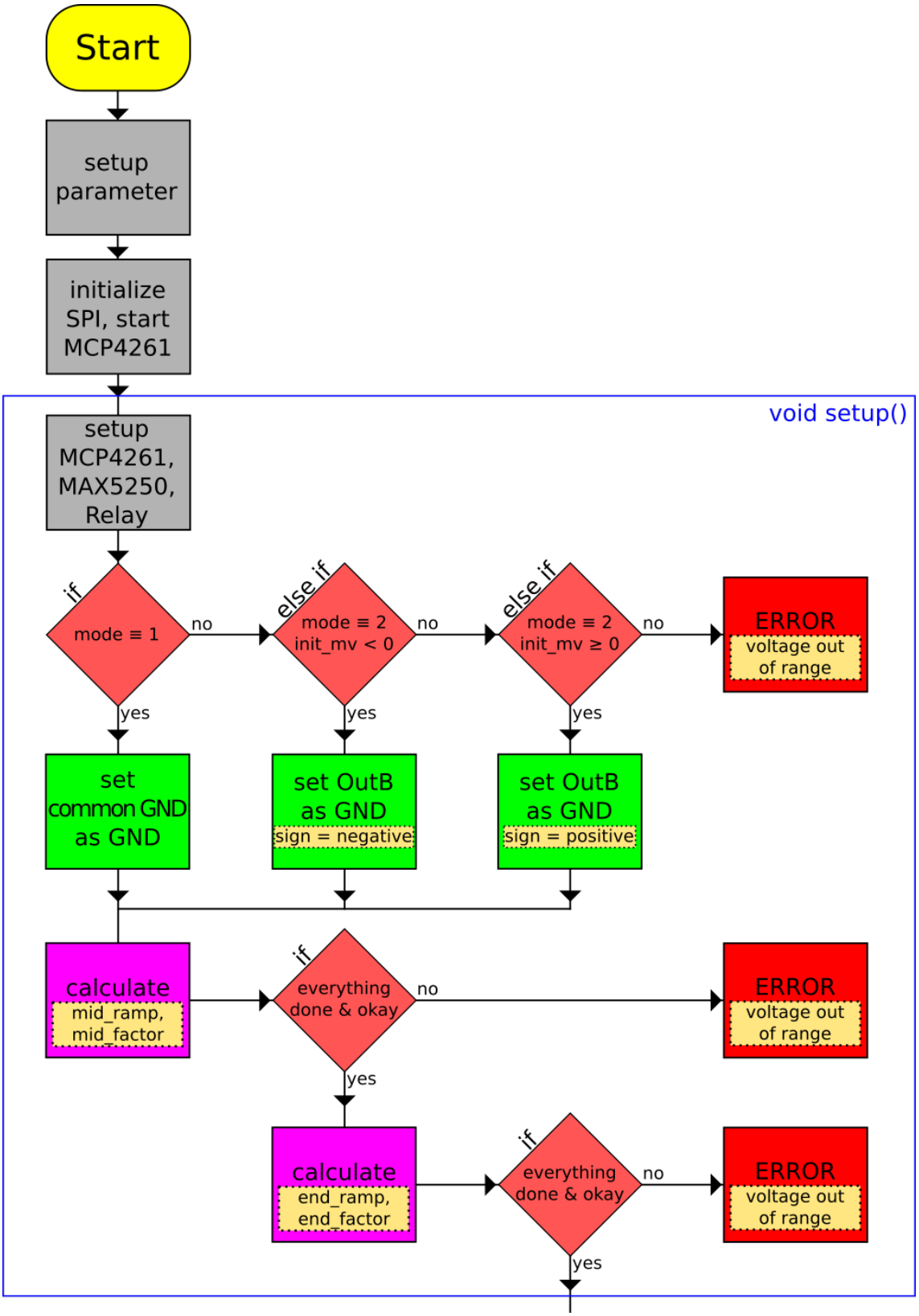


Abbildung 3.16: Programmablaufplan des Ardustat Potentiostaten, Initialisierungsroutine

Der übrige Teil der Initialisierungsroutine befasst sich mit der Variablendeklaration, der Initialisierung des Digital-Analog-Konverters und des Digitalpotentiometers, gefolgt von der Prüfung der eingegebenen Spannungs- und Zeitwerte mit anschließender Berechnung der Steigungen beider Rampen `mid_factor` und `end_factor`.

Listing 3.1: Quellcodeausschnitt: Initialisierungsroutine - Ardustat

```
1  //working modes:
2  //      1 = only positive potential between 0 and 4750 mV
3  //      2 = potential between -2500 and + 2250 mV
4  int mode = 2;
5
6  //data aquisition rate factor: 1 = every 10ms; 10 = every 100ms ...
7  int dataratefactor = 10;     // maximum 1000; minimum 1;
8
9  //initial potential with delay
10 int init_millivolt = -1000; // maximum 4750 mV; minimum -2500 mV;
11 int init_time = 1000;       // time until ramp starts
12
13 // mid-potential (raise or fall) with its delay
14 int mid_millivolt = 2000;   // maximum 4750 mV; minimum -2500 mV;
15 int mid_ramp_time = 1000;   // time ramp should take (in mV/ms)
16 int mid_time = 1000;        // time after ramp reached until next
17
18 // end-potential (raise or fall) with its delay
19 int end_millivolt = 100;     // maximum 4750 mV; minimum -2500 mV;
20 int end_ramp_time = 1000;   // time ramp should take (in mV/ms)
21 int end_time = 1000;        // time after last ramp reached
22
23 #include <EEPROM.h>
24 #include <SPI.h>
25 #include <McpDigitalPot.h>
26
27 #define DATAOUT 11           // MOSI
28 #define DATAIN 12            // MISO - not used, but part of SPI
29 #define SPICLOCK  13         // SCK
30 #define SS_DAC 10            // SLAVESELECT for MAX5250
31 #define SS_POT 7             // SLAVESELECT for MCP4261
32 #define CLEARPIN 2           // CL for MAX5250
33 #define RELAYPIN 3           // PIN for activating Relay
34
35 int adc;
36 int adc_real;
37 int outa;
38 int outb;
39 int adcgnd;
40 int adcref;
41 int adcref_real;
```

```
42 int refvolt;
43 unsigned long timer = 0;
44 unsigned long timer_m = 0;
45 int firstdac = 0;
46 int seconddac = 0;
47 int sign = 0;                // positive (1) or negative (-1) sign
48 int init_sign = 0;           // positive (1) or negative (-1) sign
49 int mid_sign = 0;            // positive (1) or negative (-1) sign
50 int end_sign = 0;            // positive (1) or negative (-1) sign
51 float mid_factor = 0.0;
52 float end_factor = 0.0;
53 int correction = 0;
54 float rAB_ohms = 49100;      // 49.1k Ohm
55 float rW_ohms  = 110;        // 110 Ohm
56 McpDigitalPot digitalPot = McpDigitalPot(SS_POT, rAB_ohms, rW_ohms);
57
58 int millivolt_A_set = 0;    // maximum 4750 mV; minimum -2500 mV; OutA
59 int millivolt_B_set = 0;    // no need to set; OutB
60
61 int newvolt = 0;
62
63 float current = 0;
64 float refcurrent = 0;
65
66 int res_rel = 100;          // relative resistance of digital pot:
67                             // 100% = 49.1k Ohm | 0% = 24.6k Ohm)
68 float res_ohm = 0;          // absolute resistence: (calculated
69                             // from relative resistance later)
70
71 void setup() {
72   Serial.begin(57600);
73   Serial.println("Starting....");
74   pinMode(DATAOUT, OUTPUT);
75   pinMode(RELAYPIN, OUTPUT);
76   pinMode(DATAIN, INPUT);
77   pinMode(SPICLOCK, OUTPUT);
78   pinMode(SS_DAC, OUTPUT);
79   pinMode(5, OUTPUT);
80   pinMode(6, OUTPUT);
81   pinMode(SS_POT, OUTPUT);
82   pinMode(CLEARPIN, OUTPUT);
83   digitalWrite(SS_DAC, HIGH);
84   digitalWrite(SS_POT, HIGH);
85   digitalWrite(CLEARPIN, LOW);
86   delay(1000);
87   digitalWrite(CLEARPIN, HIGH);
88   digitalWrite(DATAOUT, LOW);
89   digitalWrite(SPICLOCK, LOW);
90   digitalWrite(RELAYPIN, HIGH);
```

```
91    delay(500);
92    SPI.begin();
93    digitalPot.setPosition(0, 0);
94    digitalPot.setPosition(1, 0);
95    digitalPot.scale = 100.0;
96
97  // blink when ready
98    digitalWrite(5, HIGH);
99    digitalWrite(6, LOW);
100   delay(500);
101   digitalWrite(5, LOW);
102   digitalWrite(6, LOW);
103
104 // check data rate factor
105   if (datarateFactor <= 0 || datarateFactor > 1000) {
106     datarateFactor = 1;
107   }
108 // convert millivolt to output
109   if (init_millivolt >= 0 && init_millivolt <= 4750 && mode == 1) {
110     newvolt = init_millivolt;
111     millivolt_A_set = init_millivolt;
112     init_sign = 1;
113   }
114   else if (init_millivolt >= -2500 && init_millivolt < 0 && mode ==
        2) {
115 // set OutB to 2500 mV as new GND
116     millivolt_B_set = 2500;
117     newvolt = init_millivolt;
118     millivolt_A_set = init_millivolt + 2500;
119     init_sign = -1;
120   }
121   else if (millivolt_A_set >= 0 && millivolt_A_set <= 2250 && mode ==
        2) {
122 // set OutB to 2500 mV as new GND
123     millivolt_B_set = 2500;
124     newvolt = init_millivolt;
125     millivolt_A_set = init_millivolt + 2500;
126     init_sign = 1;
127   }
128   else {
129     Serial.println("Initial voltage out of range!");
130     digitalWrite(5, HIGH);
131     digitalWrite(6, LOW);
132     delay(500);
133     digitalWrite(5, LOW);
134     digitalWrite(6, LOW);
135     delay(500);
136     exit(0);
137   }
```

```
138 // calculate mid_factor and end_factor for ramps
139   if (mid_millivolt >= 0 && mid_millivolt <= 4750 && mode == 1) {
140     if (mid_ramp_time > 0) {
141       mid_factor = ((float)mid_millivolt - (float)init_millivolt) / (
      float)mid_ramp_time;
142     }
143     else {
144       mid_factor = 0;
145     }
146     mid_sign = 1;
147   }
148   else if (mid_millivolt >= -2500 && mid_millivolt < 0 && mode == 2)
      {
149     init_millivolt = init_millivolt + 2500;
150     mid_millivolt = mid_millivolt + 2500;
151     if (mid_ramp_time > 0) {
152       mid_factor = ((float)mid_millivolt - (float)init_millivolt) / (
      float)mid_ramp_time;
153     }
154     else {
155       mid_factor = 0;
156     }
157     mid_sign = -1;
158   }
159   else if (mid_millivolt >= 0 && mid_millivolt <= 2250 && mode == 2)
      {
160     init_millivolt = init_millivolt + 2500;
161     mid_millivolt = mid_millivolt + 2500;
162     if (mid_ramp_time > 0) {
163       mid_factor = ((float)mid_millivolt - (float)init_millivolt) / (
      float)mid_ramp_time;
164     }
165     else {
166       mid_factor = 0;
167     }
168     mid_sign = 1;
169   }
170   else {
171     Serial.println("Middle voltage out of range!");
172     digitalWrite(5, HIGH);
173     digitalWrite(6, LOW);
174     delay(500);
175     digitalWrite(5, LOW);
176     digitalWrite(6, LOW);
177     delay(500);
178     exit(0);
179   }
180   Serial.println(mid_factor);
181
```

```
182   if (end_millivolt >= 0 && end_millivolt <= 4750 && mode == 1) {
183     if (end_ramp_time > 0) {
184       end_factor = ((float)end_millivolt - (float)mid_millivolt) / (
      float)end_ramp_time;
185     }
186     else {
187       end_factor = 0;
188     }
189     end_sign = 1;
190   }
191   else if (end_millivolt >= -2500 && end_millivolt < 0 && mode == 2)
      {
192     end_millivolt = end_millivolt + 2500;
193     if (end_ramp_time > 0) {
194       end_factor = ((float)end_millivolt - (float)mid_millivolt) / (
      float)end_ramp_time;
195     }
196     else {
197       end_factor = 0;
198     }
199     end_sign = -1;
200   }
201   else if (end_millivolt >= 0 && end_millivolt <= 2250 && mode == 2)
      {
202     end_millivolt = end_millivolt + 2500;
203     if (end_ramp_time > 0) {
204       end_factor = ((float)end_millivolt - (float)mid_millivolt) / (
      float)end_ramp_time;
205     }
206     else {
207       end_factor = 0;
208     }
209     end_sign = 1;
210   }
211   else {
212     Serial.println("End voltage out of range!");
213     digitalWrite(5, HIGH);
214     digitalWrite(6, LOW);
215     delay(500);
216     digitalWrite(5, LOW);
217     digitalWrite(6, LOW);
218     delay(500);
219     exit(0);
220   }
221   Serial.println(end_factor);
222   delay(500);
223 }
```

Innerhalb der Software des Potentiostaten lassen sich nur wenige Funktionen sinnvoll auslagern. Hierbei handelt es sich um die beiden Funktionen zum Ansprechen der zentralen Regelungskomponenten Digital-Analog-Konverter und Digital-Potentiometer.

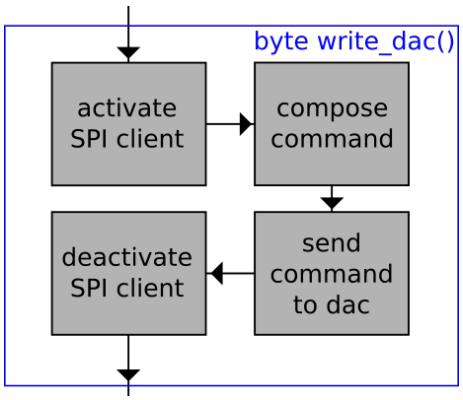


Abbildung 3.17: Programmablaufplan des Ardustat Potentiostaten, ausgelagerte Funktionen

Zum Einstellen einer bestimmten Ausgangsspannung an einem Ausgang des Digital-Analog-Konverters muss dieser erst initialisiert werden. Dies geschieht in der Funktion `write_dac()`, bevor dort die zuvor erzeugte Bytefolge mit Informationen zum gewünschten Ausgangsport `address` mit dem Spannungswert `value` auf den Chip übertragen wird. In Anschluss muss die Verbindung zum Chip ordnungsgemäß terminiert werden, damit ein erneuter Aufruf der Funktion nicht wegen einer noch offen gehaltenen Verbindung mit einem Fehler abbricht. Die Funktion `write_pot()` stellt nur eine zentrale Umrechnungsfunktion des gewünschten, menschenlesbaren Widerstandswerts auf den korrelierenden, chipspezifischen Wert dar.

Listing 3.2: Quellcodeausschnitt: Einstellfunktionen DAC und POT - Ardustat

```
1  byte write_dac(int address, int value) {
2    SPI.beginTransaction(SPISettings(10000000, MSBFIRST, SPI_MODE0));
3    firstdac = (address << 6) + (3 << 4) + (value >> 6);
4    seconddac = (value << 2 ) & 255;
5    digitalWrite(SS_DAC, LOW);
6    delay(1);
7    SPI.transfer(firstdac);
8    SPI.transfer(seconddac);
9    digitalWrite(SS_DAC, HIGH);
10   SPI.endTransaction();
11 }
12 byte write_pot(int address, int value) {
13   digitalPot.setResistance(address, 100 - value);
14 }
```

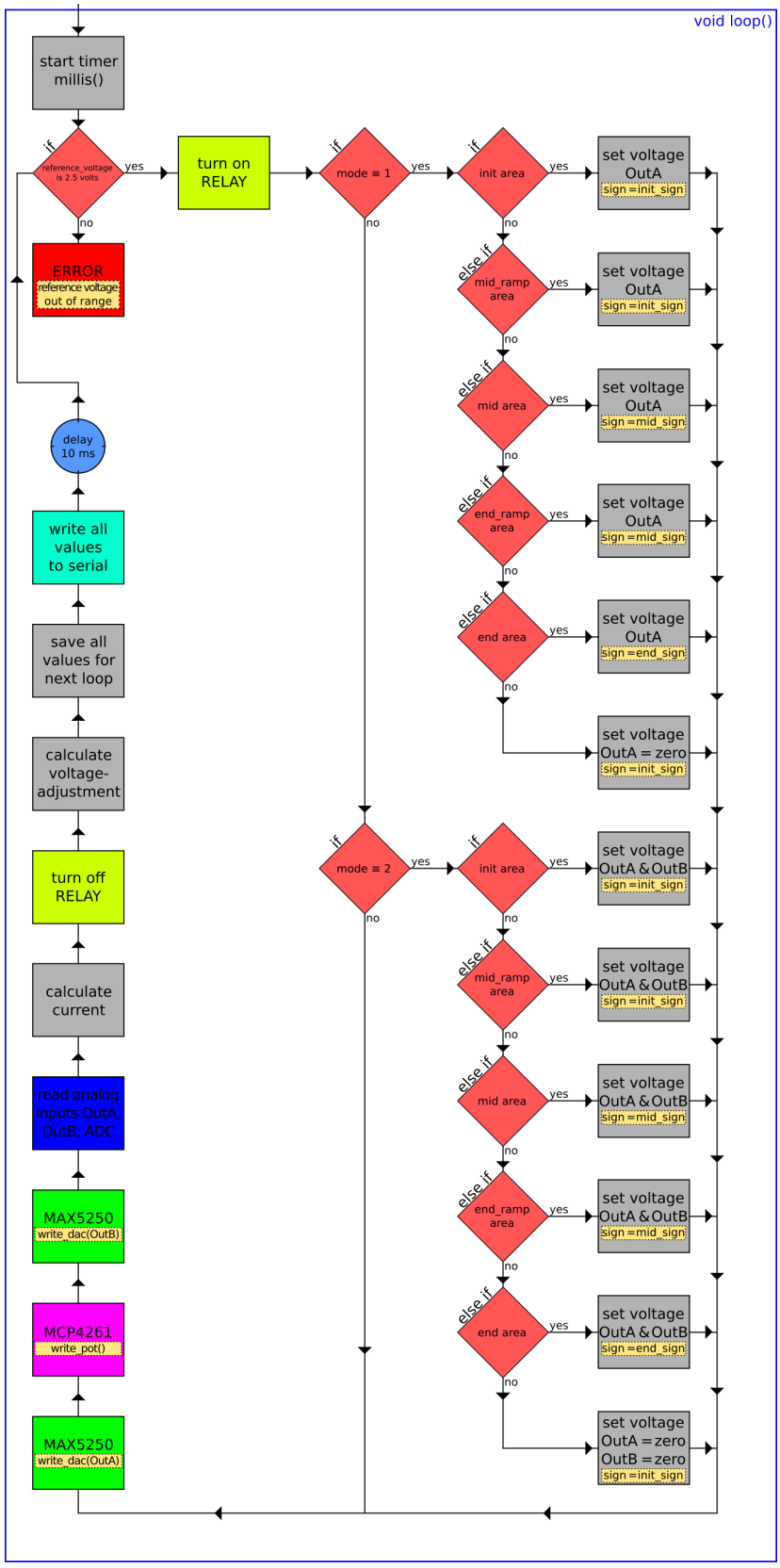


Abbildung 3.18: Programmablaufplan des Ardustat Potentiostaten, Wiederholungsroutine

Der Programmablaufplan der Wiederholungsroutine ist deutlich umfangreicher und unterscheidet zwischen den beiden Betriebsmodi mit Spannungsbereichen entweder zwischen 0 und +4,75 V (mode = 1) oder zwischen −2,5 V und +2,25 V (mode = 2), einstellbar in der Variablendeklaration im Kopfteil der Software. Nach der Spannungseinstellung in Abhängigkeit vom Betriebsmodus wird die Stromstärke durch die Einstellung eines Widerstands am digitalen Potentiometer und die Messung der daraus resultierenden Spannung durch einen analogen Eingang des Arduino über die Gleichung

$$I = \frac{U}{R} \tag{3.1}$$

berechnet. Dies geschieht sowohl für die Arbeitselektrode als auch für die Referenzelektrode. Darauf folgend wird die Spannungsadaptierung für den nächsten Zyklus berechnet und alle Werte über die serielle Schnittstelle des Arduino an den Steuerungscomputer ausgegeben.

Listing 3.3: Quellcodeausschnitt: Wiederholungsroutine - Ardustat

```
1  void loop() {
2    timer_m = millis();
3    refvolt = analogRead(5); // reference voltage from LM50
4    if (500 < refvolt < 524) {
5      digitalWrite(RELAYPIN, HIGH);
6      if (mode == 1) {
7        correction = 11;
8        if (timer <= init_time) {
9          sign = init_sign;
10       }
11       else if (timer > init_time && timer <= (init_time+mid_ramp_time
     )) {
12         millivolt_A_set = (mid_factor * (timer - init_time)) +
     init_millivolt;
13         newvolt = millivolt_A_set;
14         sign = init_sign;
15       }
16       else if (timer > (init_time+mid_ramp_time) && timer <= (
     init_time+mid_ramp_time+mid_time)) {
17         sign = mid_sign;
18       }
19       else if (timer > (init_time+mid_ramp_time+mid_time) && timer <=
      (init_time+mid_ramp_time+mid_time+end_ramp_time)) {
20         millivolt_A_set = (end_factor * (timer - (init_time +
     mid_ramp_time + mid_time))) + mid_millivolt;
21         newvolt = millivolt_A_set;
22         sign = end_sign;
23       }
24       else if (timer > (init_time+mid_ramp_time+mid_time+
     end_ramp_time) && timer <= (init_time+mid_ramp_time+mid_time+
```

```
    end_ramp_time+end_time)) {
25        sign = end_sign;
26      }
27      else {
28        millivolt_A_set = 0;
29        newvolt = millivolt_A_set;
30        sign = init_sign;
31      }
32    }
33    else if (mode == 2) {
34      if (timer <= init_time) {
35        sign = init_sign;
36        correction = -6;
37      }
38      else if (timer > init_time && timer <= (init_time+mid_ramp_time
    )) {
39        millivolt_A_set = (mid_factor * (timer - init_time)) +
    init_millivolt;
40        newvolt = millivolt_A_set - 2500;
41        sign = init_sign;
42        if (mid_factor > 0) { correction = -15; } else { correction =
     -6; }
43      }
44      else if (timer > (init_time+mid_ramp_time) && timer <= (
    init_time+mid_ramp_time+mid_time)) {
45        sign = mid_sign;
46      }
47      else if (timer > (init_time+mid_ramp_time+mid_time) && timer <=
     (init_time+mid_ramp_time+mid_time+end_ramp_time)) {
48        millivolt_A_set = (end_factor * (timer - (init_time +
    mid_ramp_time + mid_time))) + mid_millivolt;
49        newvolt = millivolt_A_set - 2500;
50        sign = end_sign;
51        if (end_factor > 0) { correction = -15; } else { correction =
     -1; }
52      }
53      else if (timer > (init_time+mid_ramp_time+mid_time+
    end_ramp_time) && timer <= (init_time+mid_ramp_time+mid_time+
    end_ramp_time+end_time)) {
54        sign = end_sign;
55      }
56      else {
57        millivolt_A_set = 0;
58        millivolt_B_set = 0;
59        newvolt = millivolt_A_set;
60        sign = init_sign;
61      }
62    }
63    write_dac(0,map(millivolt_A_set, 0, 4750, 0, 1023));
```

```
64      write_pot(0,res_rel);
65      write_dac(1,map(millivolt_B_set, 0, 4750, 0, 1023));
66
67      adc = map(analogRead(0), 0, 1023, 0, (5045 + map(res_rel, 0, 100,
        0, 25)));
68      outa = map(analogRead(1), 0, 1023, 0, 5070);
69      outb = map(analogRead(2), 0, 1023, 0, 5070);
70      adcref = map(analogRead(3), 0, 1023, 0, 5070);
71      adcgnd = map(analogRead(4), 0, 1023, 0, 5070);
72
73      res_ohm = 24600 + map(res_rel, 0, 100, 0, 24500);
74      current = (outa - adc) / (res_ohm / 1000);
75      refcurrent = (adc - adcref);
76      refcurrent = refcurrent/100;
77
78      digitalWrite(RELAYPIN, LOW);
79      adc_real = -1 * (outb - adc + correction);
80      adcref_real = -1 * (outb - adcref);
81
82      if (abs(adc_real) < abs(newvolt) && (abs(newvolt) - abs(adc_real)
        ) >= 20) {
83        millivolt_A_set = millivolt_A_set + (10 * sign);
84      }
85      else if (abs(adc_real) < abs(newvolt) && (abs(newvolt) - abs(
        adc_real)) < 20) {
86        millivolt_A_set = millivolt_A_set + (1 * sign);
87      }
88      else if (abs(adc_real) > abs(newvolt) && (abs(adc_real) - abs(
        newvolt)) >= 20) {
89        millivolt_A_set = millivolt_A_set - (10 * sign);
90      }
91      else if (abs(adc_real) > abs(newvolt) && (abs(adc_real) - abs(
        newvolt)) < 20) {
92        millivolt_A_set = millivolt_A_set - (1 * sign);
93      }
94
95      if (timer % datarateFactor == 0) { // data acquisition rate
96        Serial.println();
97  // csv-output
98        Serial.print(timer*10);     Serial.print("\t");
99        Serial.print(newvolt);      Serial.print("\t");
100       Serial.print(outa);         Serial.print("\t");
101       Serial.print(adc_real);     Serial.print("\t");
102       Serial.print(adcref_real); Serial.print("\t");
103       Serial.print(current,4);    Serial.print("\t");
104       Serial.print(refcurrent,4);
105     }
106     timer++;
107   }
```

```
108     else {
109       Serial.println("Reference voltage out of range!");
110       digitalWrite(5, HIGH);
111       digitalWrite(6, LOW);
112       delay(500);
113       digitalWrite(5, LOW);
114       digitalWrite(6, LOW);
115       delay(500);
116     }
117     delay(10-(millis() - timer_m));   // delay for loop-time of 10ms
118   }
```

Die hier gezeigte Software beherrscht nur die Möglichkeit potentiostatischer Messungen. Sowohl die Hardware, also auch die originäre von Daniel Steingart programmierte Software erlaubt auch galvanostatische Messungen, welche jedoch für diese Arbeit nicht benötigt wurden. Softwareanpassungen bzw. -weiterentwicklungen der hier gezeigten Software, um einen zusätzlichen galvanostatischen Betriebsmodus zu erlauben sind ausdrücklich erwünscht.

3.1.2.4 Messungen

Nach Fertigstellung der Hardware des Ardustat-Shields und Übertragung der Software auf den Arduino Uno wurden erste Messungen durchgeführt. Hierzu wurde erneut auf die elektrochemische Standardreaktion der Umsetzung von Ferrocen zum Ferriciniumkation zurückgegriffen.

Abbildung 3.19 zeigt das resultierende Cyclovoltammogramm einer solchen Messung ohne Glättungsfunktion. Diese Messung entstand jedoch nicht unter Verwendung der Durchflusszelle, sondern in einer herkömmlichen elektrochemischen Einzelzelle in einem 100 mL-Becherglas, welches mit 50 mL 0,01-molarer Ferrocen-Lösung in Dichlorethan (Leitsalz Tetrabutylammoniumhydrogensulfat) befüllt und mit drei Platinblech-Elektroden bestückt war.

Man erkennt, dass bei dem Selbstbaupotentiostaten unter Verwendung günstiger Bauteile gegenüber den zuvor gezeigten Messungen ein starkes Rauschen der Messwerte herrscht. Dies ist neben der Bauteilqualität der ICs auch auf die zu geringen Taktgeschwindigkeiten sowohl des verwendeten Arduino Mikrocontrollers als auch der Verbindung zu den ICs und die schlechte Entkopplung der genutzten Schaltnetzteile vom 50 Hz Wechselstromnetz zurückzuführen. Dennoch ist das Cyclovoltammogramm gut erkennbar. Bei der Verringerung der Ferrocenkonzentration bzw. der Messung im kontinuierlichen, segmentierten Fluss sinkt die absolute Ferrocenmenge soweit ab, dass keine aussagekräftige Messung mit dem Ardustat mehr möglich ist.

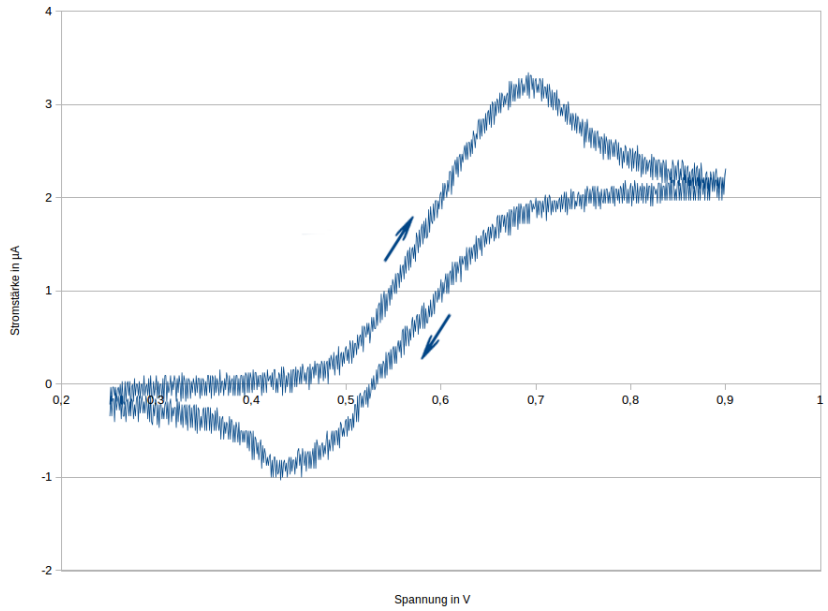


Abbildung 3.19: Ungeglättetes Cyclovoltammogramm von 0.01 M Ferrocen in DCE, gemessen mit Platinelektroden (WE, RE, CE), verbunden mit dem Ardustat-Shield und der oben gezeigten Arduino-Software

Zusammenfassend kann der hier entwickelte arduinobasierte Potentiostat zwar grundsätzlich bei potentiostatischen Messungen als preiswerte Alternative zum Einsatz kommen, ist jedoch für den Anwendungszweck in einer elektrochemischen Mikrodurchflusszelle unter Verwendung der hier eingesetzten Bauteile nicht geeignet. Die entwickelten Durchflusszellen erlauben elektrochemische Reaktionen im kontinuierlichen und segmentierten Fluss, der Umsatz der einzelnen Zelle im Fluss ist auf Grund der zeitlich sehr kurzen Spannungseinwirkung aber gering. Dies kann durch mehrere hintereinander geschaltete Durchflusszellen optimiert werden. Elektroindizierte chemische Reaktionen können schon mit Verwendung einer einzelnen Durchflusszelle durchgeführt werden.

3.2 Phasentrenner

In mehrstufigen organischen Synthesen wird oftmals im Rahmen der Aufarbeitung einzelner Reaktionsschritte eine Phasenseparation benötigt. Im herkömmlichen Chargenprozess wird hierzu im Labor ein Scheidetrichter und großtechnisch ein Absetzbecken (Settler) verwendet, sofern die zu trennenden Phasen die nötige Dichtedifferenz, eine geeignete Grenzflächenspannung und eine geringe Viskosität zur selbstständigen Entmischung aufweisen. Da bei mikrochemischen Reaktionen ein geschlossenes und druckstabiles Gesamtsystem bevorzugt wird, besteht der Bedarf nach einem Phasentrenner innerhalb des kontinuierlich betriebenen Systems.

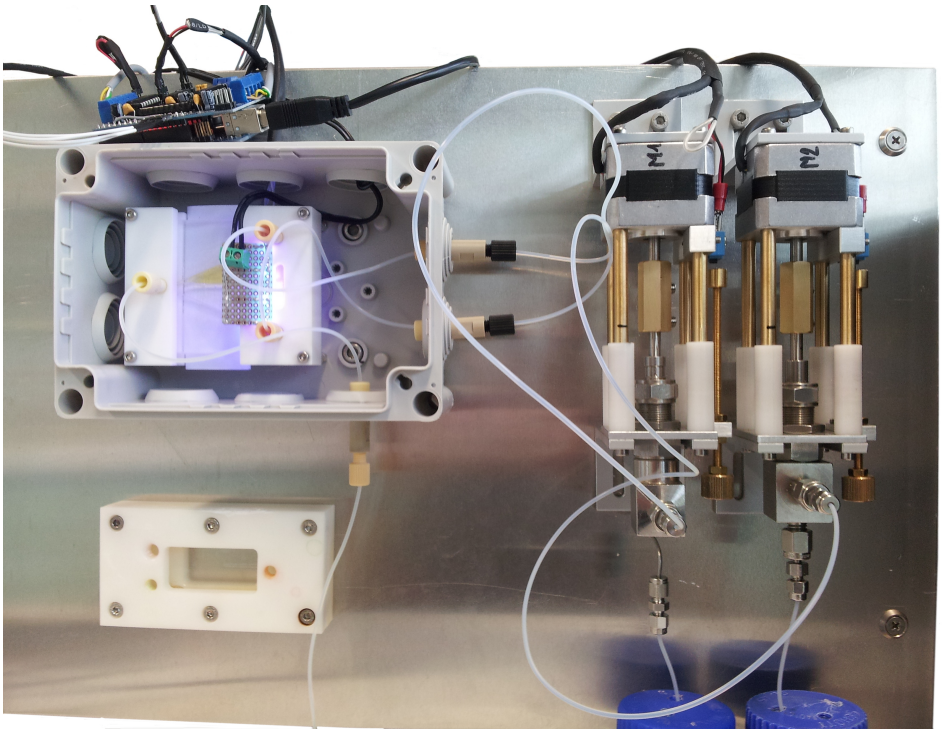


Abbildung 3.20: Entwurfsaufbau des autonomen Phasentrenners

Derzeit beschränkt sich der Markt der kommerziell erhältlichen Phasentrenner auf passive Systeme. Die Trennverfahren dieser Systeme sind sehr unterschiedlich: Es gibt Phasentrenner keilförmiger Geometrie[44], die alleine durch Gravitation für eine Trennung der Phasen sorgen. Um eine gute Trennleistung zu erzielen, welche nicht durch Kapillareffekte beeinflusst wird, ist ein im Verhältnis zu den üblichen Mikrokanälen großer Trennraum notwendig. Ein weiteres Trennverfahren besteht in der Trennung durch unterschiedliche Benetzungsfähigkeit von Oberflächen durch hydrophile und lipophile Phasen[45]. Solche Phasentrenner sind meist halbseitig aus Polytetrafluorethylen (PTFE) und auf der entsprechend anderen Seite aus Glas aufgebaut. Das Koaleszenzverhalten wird durch die Geschwindigkeitsdifferenz innerhalb des Trennsystems auf Grund der unterschiedlichen Wandbeschaffenheit begünstigt.

Ein drittes Trennverfahren beruht auf einem ähnlichen Effekt: Man verwendet einen Extraktor mit Trennmembran[46, 47], in welchem die Membran selektiv unter Nutzung von Kapillarkräften eine Phasentrennung begünstigt.

Die hier genannten Systeme werden vielfach zur Phasentrennung in kontinuierlichen Systemen eingesetzt, bringen jedoch entscheidende Nachteile mit sich. Die meisten Systeme sind durch den Einsatz spezieller Materialien auf Wandoberflächen oder als Membran auf die Trennung bestimmter Flüssigkeiten festgelegt, alle Systeme können nur so lange eine Phasentrennung durchführen, wie ein gleichmäßiger Fluss beider Phasen erfolgt. Sobald eine Veränderung des Verhältnisses der zu trennenden Phasen eintritt, versagt die Phasenseparation der hier genannten Systeme, da allen eine Ausflusssteuerung fehlt.

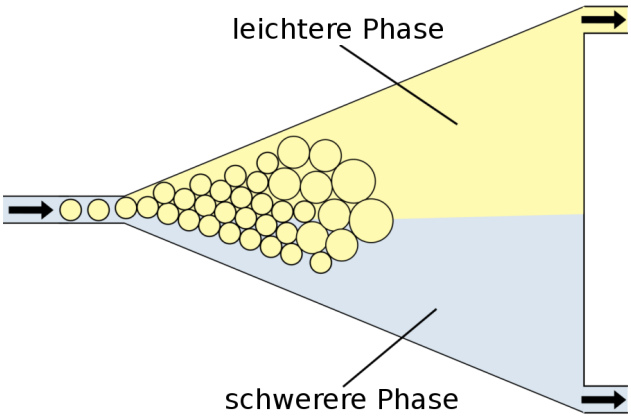


Abbildung 3.21: Schema einer dreieckigen Trennkammer

Um diesen Nachteilen zu entgehen wurde ein Phasentrennsystem geschaffen, welches die Separation zweier Phasen, die nicht zwingend in einem festen Volumenflussverhältnis vorliegen, ermöglicht. Hierzu wurde eine Phasentrennkammer mittels CFD-Berechnungen entwickelt, die eine passive Trennung zweier Phasen durch Gravitation ermöglicht. Um die Trennleistung auch bei Änderungen des Verhältnisses der Volumina beider Phasen zueinander aufrecht zu erhalten, wurde die Trennkammer mit einer aktiven Ventilsteuerung ausgestattet, die abhängig von der Lage der Phasengrenzfläche in Echtzeit eine autonome Regulierung der Auslässe ermöglicht. Die autonome Steuerung des Systems findet durch einen Mikrocontroller statt.

Das Konzept des autonomen Phasentrenners[9] wurde im Rahmen dieser Arbeit weiterentwickelt. Die ursprünglich aus Polydimethylsilan (PDMS) hergestellte dreieckige Trennkam-

[9]Erste Entwürfe des autonomen Phasentrenners wurden bereits 2014 in meiner Diplomarbeit[40] vorgestellt.

mer wurde durch die LTF GmbH[10], als Glaskörper hergestellt, sodass dieser eine weitaus bessere chemische Beständigkeit und Formstabilität gegenüber einer PDMS-Trennkammer aufweist.

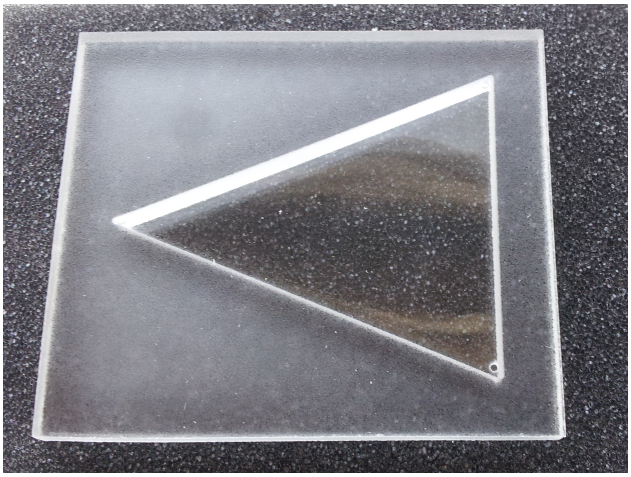


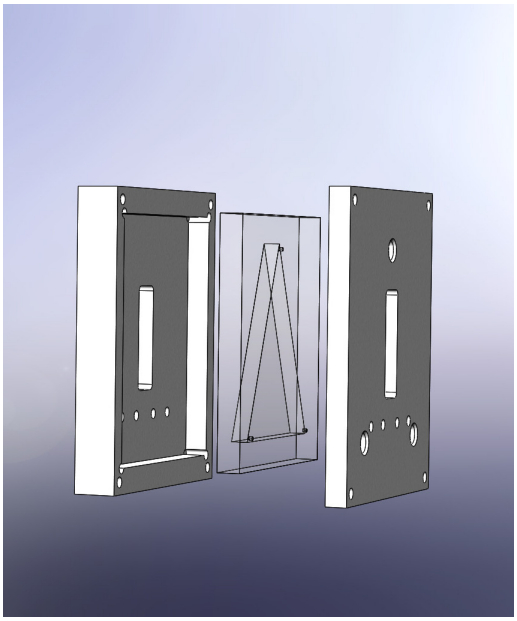
Abbildung 3.22: Dreieckige Trennkammer aus Glas, hergestellt von Little Things Factory GmbH, Elsoff

Die anfangs aus Heizungsthemostaten gebauten Ventilsteuerungen wurden durch marktübliche Nadelventile[11] ersetzt und mit Hochleistungsschrittmotoren[12] für den professionellen Einsatz versehen.
Zum Schutz der Trennkammer aus Glas und zur dichten Befestigung der Ein- und Ausläse am Glaskörper wurde eine spezielle Halterung entwickelt und aus Polyetheretherketon (PEEK) angefertigt. Die Halterung verfügt neben den Gewindeanschlüssen für handelsübliche Fittinge 1/16" (Gewinde 1/4"-28) und den Bohrungen für die Transmissionslichtschranken auch über ein Sichtfenster auf beiden Seiten, welche eine direkte Beobachtung der Phasentrennung durch den Experimentator ermöglicht.
Ebenso wurde der bewegliche Schlitten mit Ventilaufnahme und Motorhalterung für die Schrittmotoren, versehen mit einer elektrischen Endabschaltung, angefertigt. Die Endabschaltung muss bei der Ersteinrichtung der Ventilsteuerung manuell auf die Position justiert werden, dass bei Erreichen des geschlossenen Zustands des Ventils eine Rückmeldung an den Mikrocontroller erfolgt. Dies entspricht in der Software dem Nullpunkt der Ventilstellung.

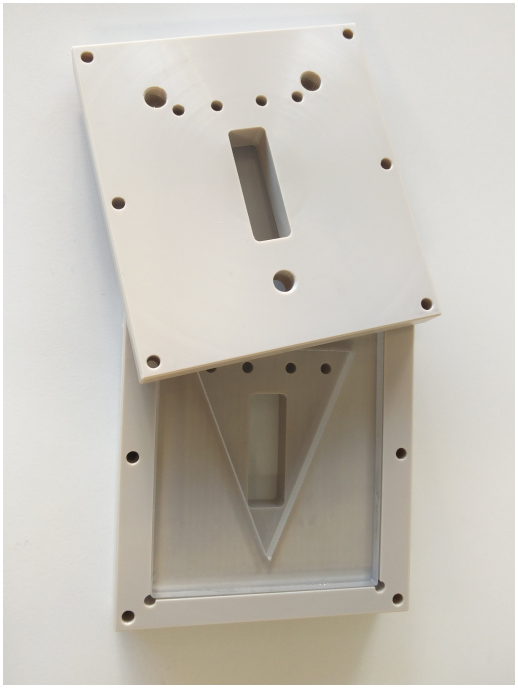
[10]Little Things Factory GmbH, Elsoff, Deutschland, `http://www.ltf-gmbh.com`

[11]Im ersten Prototypen wurden jeweils Whitey SS-22RS2(-A) Nadelventile verbaut, da diese auf Grund ihrer linearen Abhängigkeit zwischen den Umdrehungen des Ventils und dem Durchflusskoeffizienten ideal für eine automatisierte Softwaresteuerung sind. Diese Ventile werden Stand 2017 nicht mehr hergestellt, als Nachfolger wurden seitens Swagelok die Ventile Swagelok SS-SS2(-A) benannt.

[12]Schrittmotoren von Adafruit Industries[48], Product ID 324, 200 Schritte/Umdrehung, 12V, 350mA, NEMA-17, Datenblatt siehe Anhang B

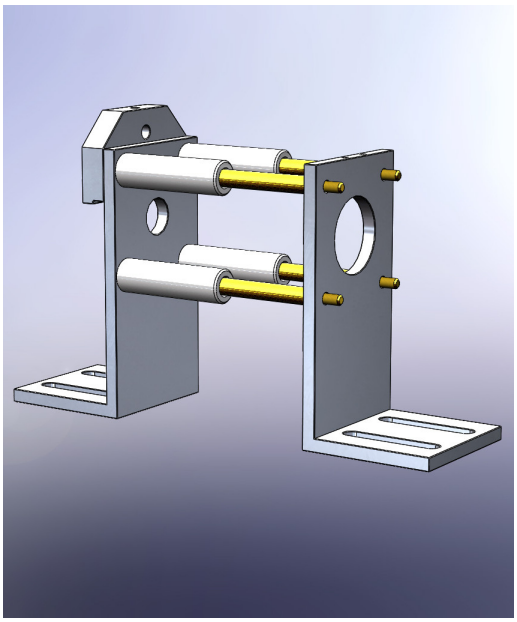


(a) 3D-Konstruktion

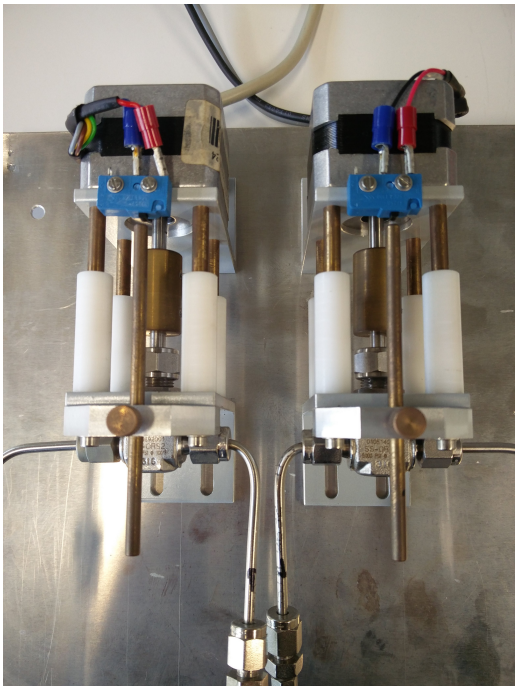


(b) Fertige Trennkammerhalterung

Abbildung 3.23: Trennkammerhalterung mit innen liegendem Glaskörper



(a) 3D-Konstruktion



(b) Fertige Ventilhalterung

Abbildung 3.24: Ventilhalterung mit Schrittmotoren und Endabschaltung

3.2.1 Aufbau

Eine 1/8" oder 1/16" Kapillare, die das zu trennende Phasengemisch führt, wird direkt mit der Trennkammer verbunden. Diese verfügt über einen oberen und einen unteren Ausgang, der jeweils zu einem elektronisch angesteuerten Nadelventil zur Regelung der Austrittsmenge beider Ausgänge führt. Am Ausgang der Nadelventile steht sowohl die schwerere als auch die leichtere Phase unabhängig voneinander für weitere Syntheseschritte zur Verfügung.

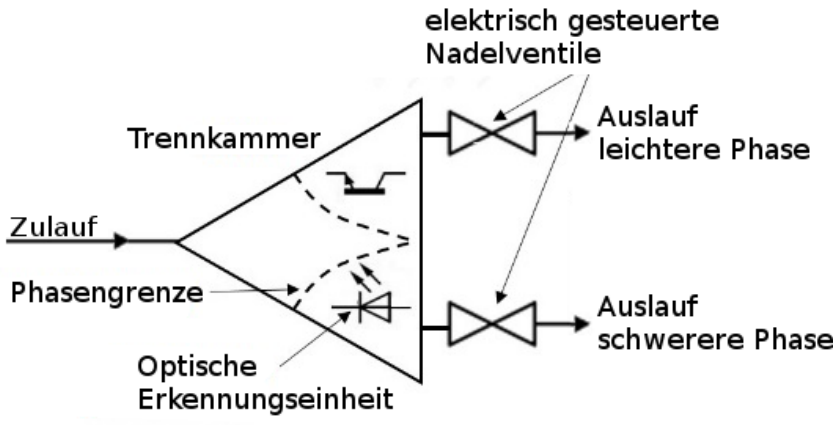


Abbildung 3.25: Fließbild der Trennkammer des Phasentrenners

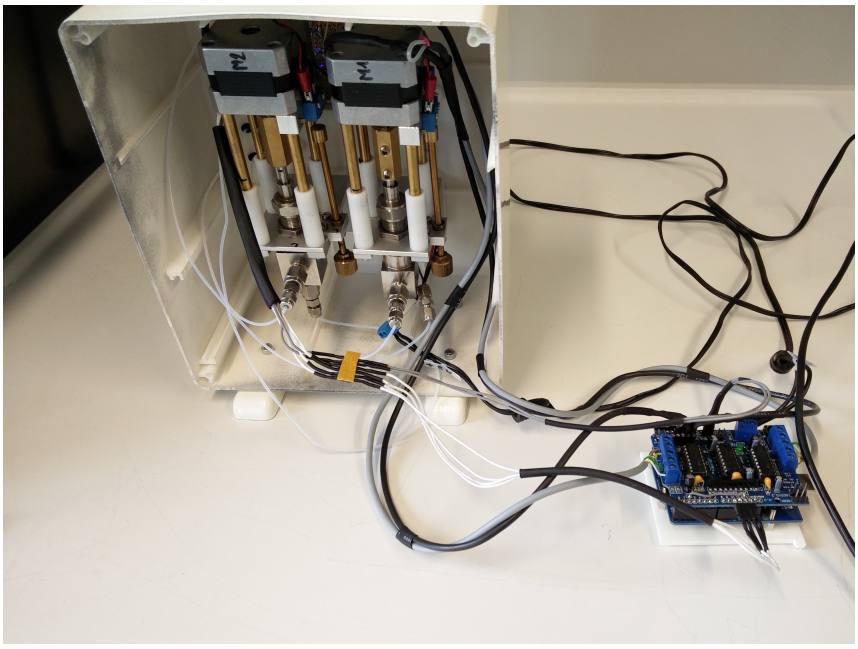


Abbildung 3.26: Rückansicht des Phasentrenners, eingebaut in ein durch selektives Lasersintern (MMT GmbH) hergestelltes Gehäuse.

3.2.2 Funktionsweise

Innerhalb der Trennkammer findet eine physikalische Trennung beider Phasen statt. Dabei stellt sich abhängig von dem Volumenverhältnis der beiden getrennten Substanzen zueinander eine horizontale Phasengrenzfläche auf entsprechender Höhe am Ende der Trennkammer ein. Durch die hier verbauten Transmissionslichtschranken wird nach vorheriger Kalibrierung die Position der Phasengrenze innerhalb der Trennkammer bestimmt. Die auf dem Mikrocontroller laufende Software stellt diese Berechnung an und reguliert daraus resultierend den Durchfluss des oberen und unteren Auslassventils. Hiermit ist gewährleistet, dass auch dynamische Volumenverhältnisse der zu trennenden Phasen effizient separiert werden können, bis hin zum einphasigen Durchfluss bei gleichzeitigem Verschluss des entsprechend anderen Auslasses.

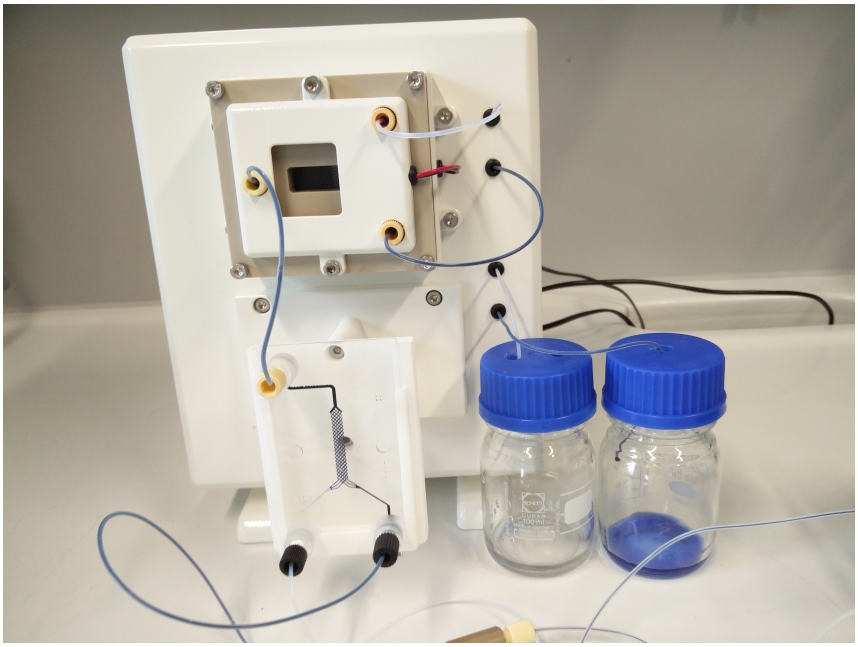


Abbildung 3.27: Frontansicht des Phasentrenners während der Trennung von Wasser (mit Tinte blau angefärbt) und Toluol

3.2.3 Technische Realisierung

Die Bestimmung der Position der Phasengrenze innerhalb der dreieckigen Trennkammer erfolgt durch einen Arduino Uno Mikrocontroller. An vier Positionen in der Vertikalen am Ende der Trennkammer sind Transmissionslichtschranken angebracht, bestehend jeweils aus einer UV-LED[13] auf der einen Seite der Trennkammer und einem Phototransistor[14] auf

[13] Bivar UV5TZ-390-30, Emissionsmaximum bei $\lambda = 390\,nm$, Austrittswinkel von 30°, Datenblatt siehe Anhang B

[14] Osram SFH 309, Datenblatt siehe Anhang B

der gegenüberliegenden Seite. Der Phototransistor verhält sich wie ein von der Intensität der einfallenden UV-Strahlung abhängiger elektrischer Widerstand, dessen Spannungsänderung durch den Arduino gemessen und verarbeitet wird.

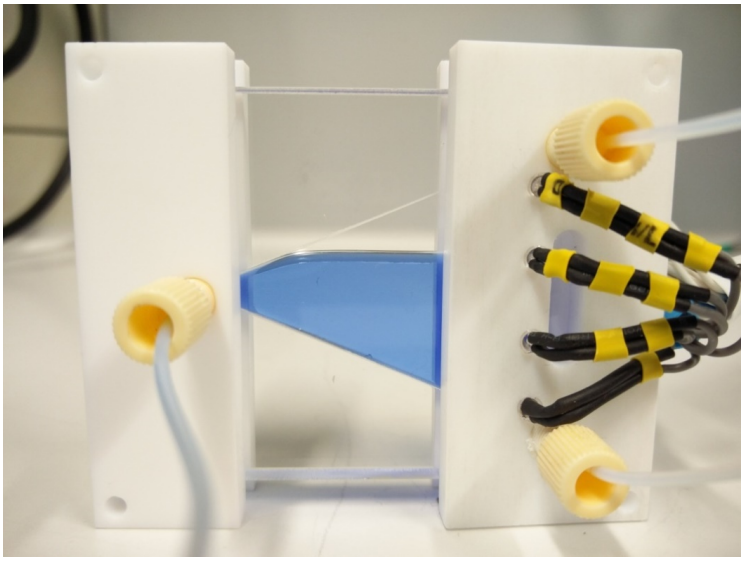


Abbildung 3.28: Trennung von Wasser (mit Tinte blau angefärbt) und Toluol

Ursprünglich wurde zur Messung dieser Spannung jeweils ein analoger Eingang des Arduinos[15] genutzt. Hier zeigten sich jedoch die Grenzen des Mikrocontrollers, da häufiges sogenanntes Übersprechen[16] die Messungen verfälschte und aufwendige Fehlerkorrekturen nötig machte. Auch im Hinblick auf eine Austauschbarkeit des zentralen Mikrocontrollers durch leistungsfähigere Hardware wurde ein speziell für die Umwandlung analoger Signale hergestellter und am Markt verfügbarer Analog-Digital-Wandlerchip[17] mit einer Auflösung von 12 Bit[18] verwendet, der MCP3208[19] von Microchip Technology Inc.. Dieser

[15]Der verwendete Atmel ATMEGA328P des Arduino Uno besitzt 6 analoge Eingänge, die mittels integriertem 10-Bit Analog-Digital-Wandler ausgelesen werden können.

[16]Der Begriff Übersprechen stammt ursprünglich aus der Telekommunikation, als man am analogen Telefon leise andere Gespräche mithören konnte und bezieht sich auf den Effekt der Beeinflussung analoger Signale untereinander. Nahe beieinander liegende signalführende Adernpaare stellen einen gemeinsamen elektrischen Schwingkreis dar. Die durch die entsprechenden Adern laufenden Signale werden hierdurch auf andere Adern eingekoppelt, was zu einer Veränderung des dort durchlaufenden Signals führt. Da jede Leiterbahn sowohl als Sender als auch als Empfänger fungiert, unterliegen alle Signale einer Möglichen Verfälschung. Um die Datenintegrität sicherzustellen, setzt man neben der Abschirmung der Adern auch auf eine frühzeitige Umwandlung der analogen Signale in digitale Signale und anschließende Übertragung mittels datenintegritätsgesicherten Übertragungsprotokollen.

[17]Ein Analog-Digital-Wandlerchip konvertiert eine an seinem analogen Eingang gemessene Spannung in einen digitalen Wert.

[18]12 Bit Auflösung entspricht $2^{12} = 4096$ Schritten, in die zwischen der unteren Referenzspannung (meist Masse, 0 V) und der oberen Referenzspannung (3,3 V oder wie hier 5 V) unterteilt gemessen wird.

[19]Microchip Technology Inc. MCP3208, Datenblatt siehe Anhang B

Analog-Digital-Wandlerchip sitzt direkt neben den Phototransistoren, um die analogen Signalwege kurz zu halten, ist gegenüber dem Mikrocontrollerboard in einem platzsparenden DIP-16-Gehäuse[20] untergebracht und kann mit einer Betriebsspannung zwischen 2,7 V und 5,5 V versorgt werden, sodass neben der Verwendung des Arduino Uno (5 V System) auch ein Arduino Due oder Raspberry Pi (beide 3,3 V System) als Steuerungseinheit in Frage kommt.

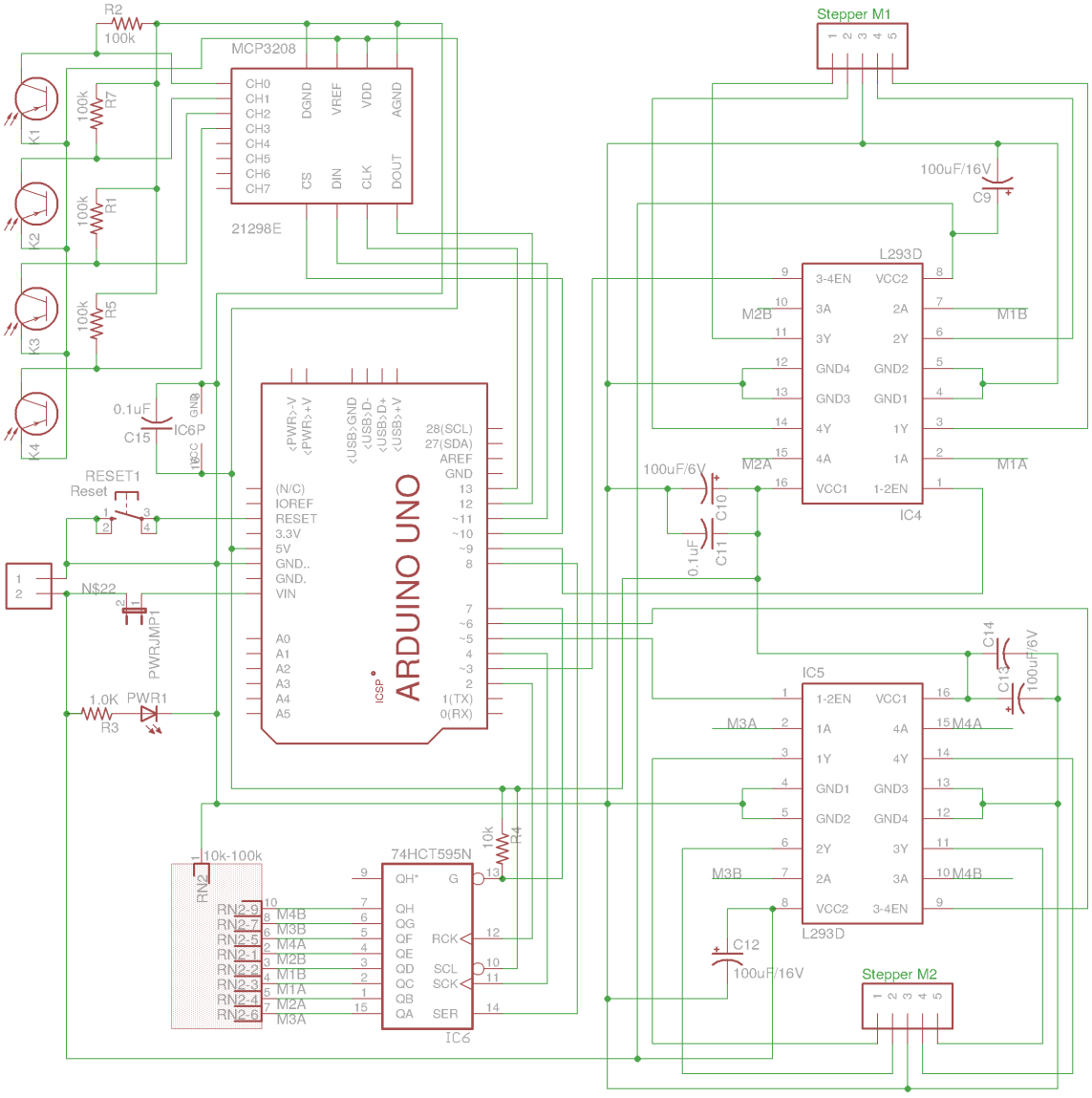


Abbildung 3.29: Schaltplan der Platinen des Phasentrenners mit Phototransistoren

[20] DIP-16 steht für ein 16-poliges *dual in-line package*, eine Gehäuseform mit zwei Anschlussreihen mit je 8 Kontakten, die zur Durchsteckmontage auf Lochrasterplatinen und in passenden Sockeln geeignet ist.

Der MCP3208 kommuniziert über *SPI*[21] mit dem Mikrocontroller und übermittelt nach seiner Initialisierung die Messwerte der einzelnen analogen Eingänge. Derzeit werden 4 der 8 analogen Eingänge für die vier Transmissionslichtschranken verwendet - eine Erweiterung der Anzahl der Lichtschranken ist dadurch jedoch möglich[22]. Auf Anfrage durch den Mikrocontroller liest der MCP3208 die Spannungswerte der einzelnen Lichtschranken aus und wandelt diese in einen Wert zwischen 0 (entspricht keinem Lichteinfall auf den Phototransistor) und 4096 (entspricht dem Maximum der Photoempfindlichkeit des Phototransistors) um, welcher an den Mikrocontroller übermittelt wird. Dieser sammelt jeweils eine vorher festgelegte Anzahl einzelner Messwerte und mittelt über diese, um eventuelle einzelne Messfehler der Phototransistoren auszugleichen. Im Folgenden wird von diesen gemittelten Messwerten der einzelnen Lichtschranken bei der Positionsvorhersage der Phasengrenze ausgegangen.

Da der Arduino Uno nur sehr kleine Ströme an seinen digitalen Ausgängen zulässt, ist er alleine nicht dazu in der Lage, größere Motoren zu betreiben. Dies wird beispielsweise mit Hilfe einer Brückenschaltung realisiert, wie sie gleich vierfach in Form des L293D-Chips[23] von Texas Instruments auf dem hier verwendeten Adafruit MotorShield[24] vorhanden ist. Diese Zusatzplatine stellt das zur Ansteuerung von Schrittmotoren nötige Taktsignal und die 12 V Spannung zur Verfügung. Sie nutzt bis auf die Pins `D2`, `D9`, `D10` und Pin `D13` alle digitalen Ein- und Ausgänge, die der Arduino Uno zur Verfügung stellt. Zur Kommunikation mit dem Analog-Digital-Wandlerchip MCP3208 über *SPI* werden jedoch die digitalen Pins `D10`, `D11`, `D12` und `D13` benötigt. Die Leiterbahnen der beiden belegten Pins `D11` und `D12` wurden auf dem MotorShield aufgetrennt und die eigentlich mit Pin `D12` verbundene Leiterbahn mit Pin `D2` verbunden und verlötet. So sind alle Pins für die *SPI*-Kommunikation frei bei gleichzeitig uneingeschränkter Funktion des MotorShields für die hier nötige Schrittmotorsteuerung. Im Zuge der Lötarbeiten wurde eine zusätzliche Buchsenleiste zum Anschluss des MCP3208 an die Pins `D10` bis `D13` gelötet (Abbildung 3.30 Mitte unten).

[21] *SPI* steht für Serial Peripheral Interface und wurde Anfang der 1980er Jahre von Motorola entwickelt. Hierbei handelt es sich um einen hauptsächlich zur geräteinternen Anbindung genutzten seriellen Datenbus, welchem das Master-Slave-Prinzip zu Grunde liegt: die Initiierung von Datentransfers findet immer durch den Master statt, welcher einen Slave explizit unter seiner Adresse abfragt.

[22] Die Anzahl von vier Transmissionslichtschranken wurde gewählt, da mehr Lichtschranken eine stärkere Fokussierung der UV-LEDs erfordern: Die verwendeten UV-LEDs haben einen Austrittswinkel von 30°, was bei einer Distanz zum gegenüberliegenden Phototransistor von 7 mm einen Lichtkegel mit dem Durchmesser von 7 mm dort erzeugt. Die Lichtschranken haben einen minimalen Abstand von 10 mm zueinander, um sowohl das direkte Einfallen des Lichts benachbarter UV-LEDs als auch das Einfallen von Streulicht benachbarter UV-LEDs zu verhindern.

[23] Texas Instruments L293D, Datenblatt siehe Anhang B

[24] Es wurde das Adafruit MotorShield[49], Product ID 81, von Adafruit Industries[48] in Version 1.2, Datenblatt siehe Anhang B, eingesetzt (Stand 2017 nicht mehr verfügbar, Nachfolger ist Product ID 1438 bzw. baugleiche Modelle wie das MotorShield von DK Electronics zu sehen in Abbildung 3.30).

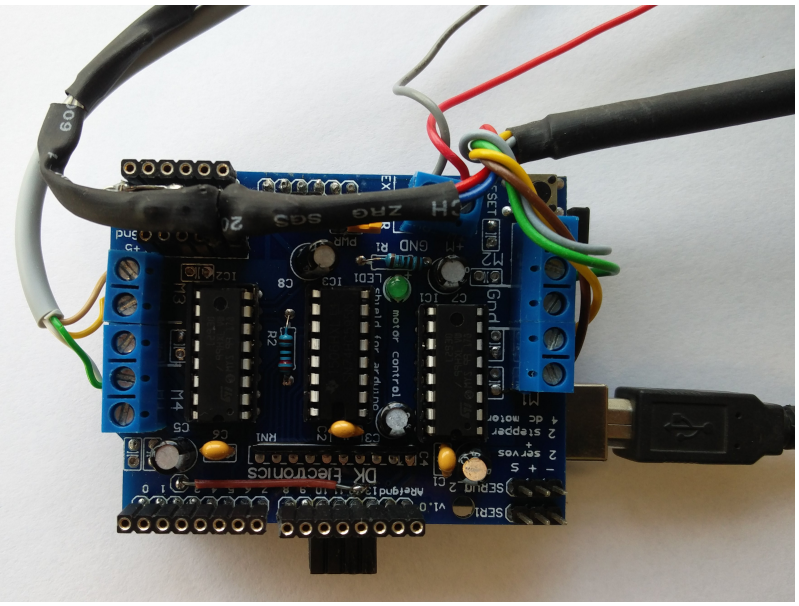


Abbildung 3.30: Modifiziertes MotorShield mit Verbindung der Leiterbahn von D12 zu D2 (horizontales braunes Kabel unten)

3.2.4 Software

Die speziell für den autonomen Phasentrenner entwickelte Software läuft lokal auf dem Arduino Uno, eine Anbindung an einen Computer ist zur Funktionsüberwachung möglich, jedoch nicht erforderlich. Die Software nutzt zur Steuerung der Ventile über die Schrittmotoren eine für das Adafruit MotorShield von Adafruit Industries entwickelte Softwarebibliothek [49]. Neben der oben beschriebenen Hardwaremodifikation des MotorShields zur parallelen Nutzung gemeinsam mit dem MCP3208 wurde die originale Softwarebibliothek (Stand 2012) durch Änderung von Zeile 138 der AFMotor.h, `#define MOTORLATCH 12` in `#define MOTORLATCH 2`, angepasst.

Der spezielle Betriebsmodus der optischen Phasengrenzflächenerkennung erfolgt auf Grund der nötigen Speicher- und Rechenleistung bei der Erkennung des Grenzflächendurchtritts durch eine der Lichtschranken nicht lokal auf dem Arduino. Hierfür wird ein separater Steuerungsrechner zur Auswertung oder ein leistungsstärkerer Ersatz für den Arduino Uno benötigt.

Der Programmablaufplan der autonomen Steuerung des Phasentrenners beginnt mit der Initialisierung der Ventile, indem diese bis zum Erreichen des jeweiligen Endabschalters in Richtung ihrer geschlossenen Position bewegt werden. An diesem Punkt ist das Ventil gerade geschlossen, entspricht dem Ventilzustand „0 Umdrehungen“ seines Datenblatts (Anhang B).

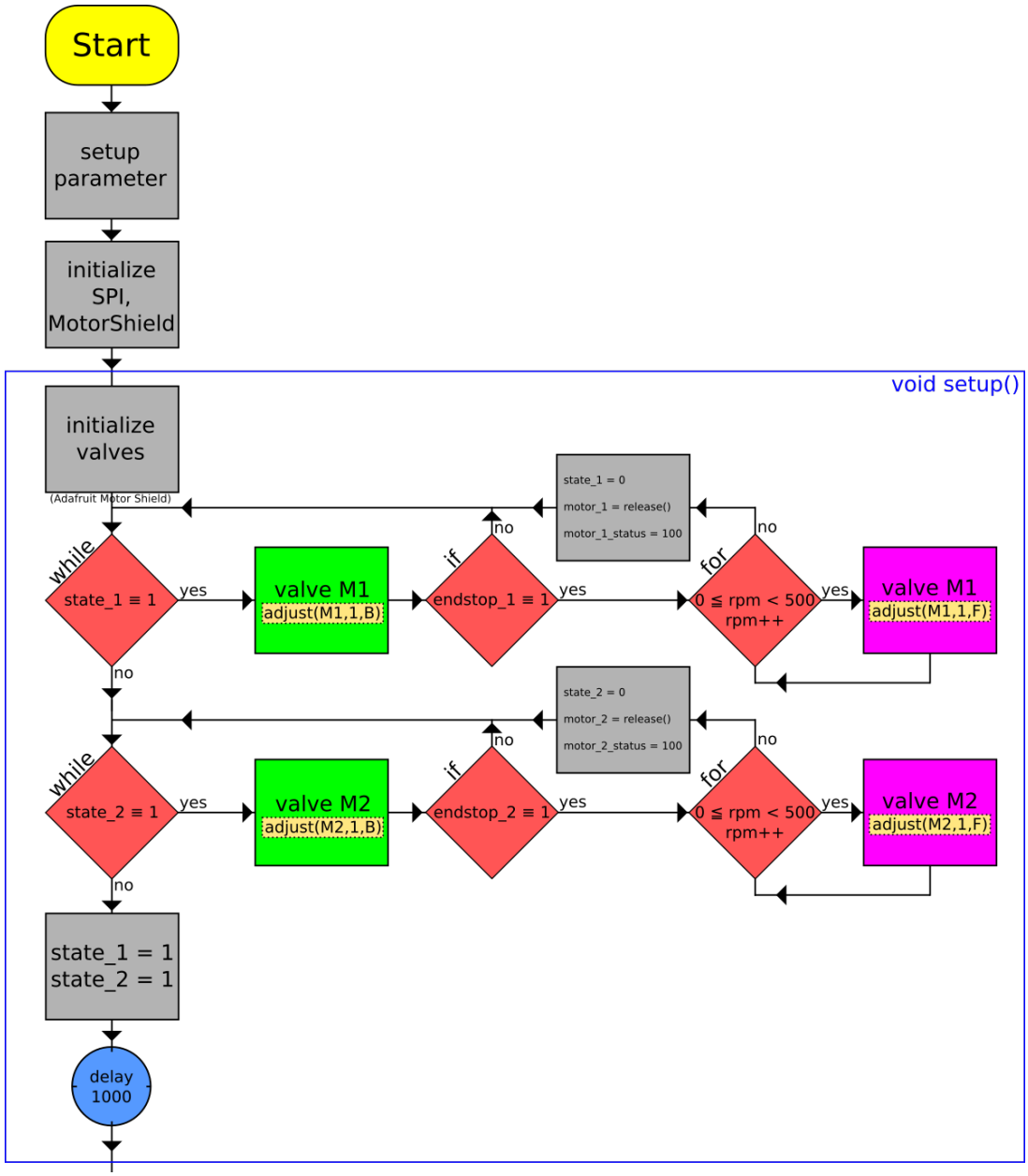


Abbildung 3.31: Programmablaufplan des Phasentrenners, Initialisierungsroutine

Abschließend werden die Ventile um eine vorher festgelegte Anzahl an Umdrehungen, umgerechnet in Motorschritten, geöffnet, um kontrolliert und kalibriert die folgenden Messungen zu beginnen. Die Anzahl der Schritte wird über die Variable `startvalue` im Kopfbereich der Software festgelegt. Hier wird auch die maximale Anzahl an Schritten, die das Ventil durch die Software geöffnet werden darf, festgelegt, um diese nicht zu überdrehen oder mit den Motoren in den Ventilanschlag zu steuern. Die Einhaltung der minimalen (Nullstellung) und der maximalen (`maxOpenValue`) Grenze der Ventilposition, gemessen in Motorschritten, wird ständig durch die Software überwacht, damit es nicht zu einem Druckaufbau innerhalb des Systems durch zu weit geschlossene Ventile kommt.

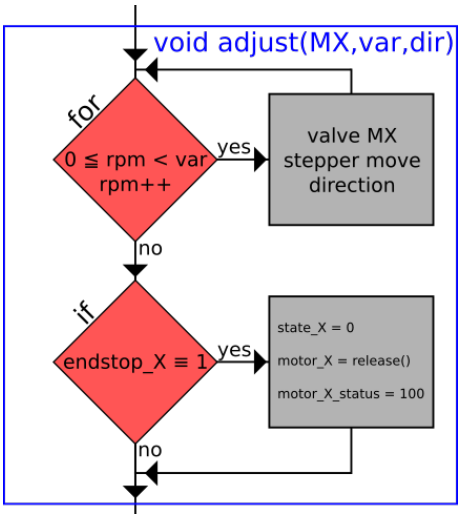
Listing 3.4: Quellcodeausschnitt: Initialisierungsroutine - Phasentrenner

```
// average of numReadings
const int numReadings = 100;

// time between measurements
const int seconds = 60;

// valve calibration on (1) / off (0)
const int valveCalibration = 1;

// valve adjustment on (1) / off (0)
const int valveActive = 1;

// threshold value {denser phase, rarer phase }
int sensorValue[4][2] = {
                          { 100, 450 },
                          { 100, 450 },
                          { 100, 450 },
                          { 100, 450 }
                        };

// modificated AdafruitMotorShield-library
// see text for details
#include <AFMotorMod.h>

// valveM1 (bottom valve)
AF_Stepper motor_1(200, 2);

// valveM2 (top valve)
AF_Stepper motor_2(200, 1);

// {MotorNumber, State, Status, EndstopPin}
int motor_array[2][4] = {
                          {1,1,0,15},
                          {2,1,0,14}
                        };

// initial steps (200 steps = 1 turn)
const int startvalue = 600;

// maximum steps (200 steps = 1 turn)
const int maxOpenValue = 1000;

long total[8] = { 0, 0, 0, 0, 0, 0, 0, 0 };
int average[8] = { 0, 0, 0, 0, 0, 0, 0, 0 };
int value_mcp3208 = 0;


```

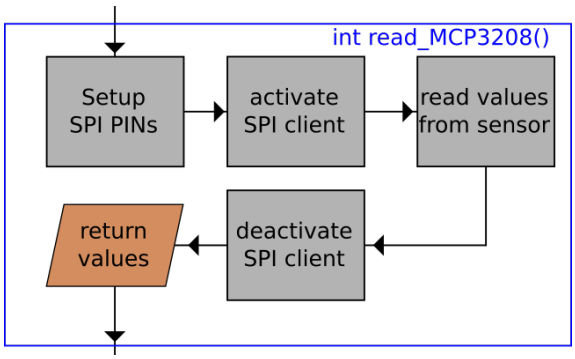
```
48
49 // SPI
50 #define SELPIN 10                          //Selection PIN
51 #define DATAOUT 11                         //MOSI
52 #define DATAIN  12                         //MISO
53 #define SPICLOCK  13                       //Clock
54 int readvalue;
55
56 void setup() {
57  pinMode(SELPIN, OUTPUT);
58  pinMode(DATAOUT, OUTPUT);
59  pinMode(DATAIN, INPUT);
60  pinMode(SPICLOCK, OUTPUT);
61  digitalWrite(SELPIN,HIGH);
62  digitalWrite(DATAOUT,LOW);
63  digitalWrite(SPICLOCK,LOW);
64  pinMode(17, INPUT_PULLUP);
65  pinMode(motor_array[0][3], INPUT);
66  pinMode(motor_array[1][3], INPUT);
67  motor_1.setSpeed(50);
68  motor_2.setSpeed(50);
69
70  Serial.begin(9600);
71
72  if (valveCalibration == 1) {
73    Serial.println("Ventile in Ausgangsposition bringen.");
74    for (int number = 1; number < ((sizeof(motor_array)/sizeof(
     motor_array[0]))+1); number++) {
75      while (motor_array[number-1][1] == 1) {
76        MoveMotor(number, 'F');
77        if (digitalRead(motor_array[number-1][3]) == 1 ) {
78          for (int rpm = 0; rpm < startvalue; rpm++) {
79            MoveMotor(number, 'B');
80          }
81          motor_array[number-1][1] = 0;
82          MoveMotor(number, 'R');
83          motor_array[number-1][2] = startvalue;
84          Serial.print("Ventil M");
85          Serial.print(number);
86          Serial.print(" bereit ( ");
87          Serial.print(startvalue);
88          Serial.println(" Schritte).");
89        }
90      }
91      motor_array[number-1][1] = 1;
92    }
93  }
94
95
```

```
96   else {
97     Serial.println("Ventilkalibrierung deaktiviert!");
98     Serial.println("(Nutzung der Ventile auch deaktiviert)");
99   }
100  Serial.println("Startphase abgeschlossen.");
101  delay(1000);
102 }
```

In der Softwareentwicklung ist es üblich, mehrfach genutzte Teile des Programmcodes in sogenannte Funktionen[25] auszulagern, welche wiederum in Klassen[26] zusammengefasst werden. Dies ermöglicht neben der besseren Übersicht und leichteren Fehlererkennung und -behebung auch die deutlich einfachere Anpassung der Software an Hardwareänderungen, beispielsweise den Wechsel auf einen Analog-Digital-Wandlerchip eines anderen Herstellers. Hierfür muss der Quellcode nur noch an einer zentralen Stelle editiert werden.



(a) Funktion zur Ventilsteuerung



(b) Funktion zum Auslesen des MCP3208

Abbildung 3.32: Programmablaufplan des Phasentrenners, ausgelagerte Funktionen

Auch in der Software des Phasentrenners sind wichtige wiederkehrende Bestandteile, wie die Funktion zum Auslesen des MCP3208, `read_mcp3208()` und auch die Funktion zum Öffnen und Schließen der Ventile, `adjust()`[27], in separate Funktionen ausgelagert.

[25] Durch die Segmentierung von Programmiercode in Funktionen lassen sich modulare Codeteile erstellen, die eine definierte Aufgabe ausführen und dann zu dem Codebereich zurückkehren, von dem die Funktion aus aufgerufen wurde. Der häufigste Fall für das Erstellen einer Funktion ist, wenn die gleiche Aktion mehrmals in einem Programm ausgeführt werden muss.

[26] Klassen zum Zusammenfassen von Funktionen werden bevorzugt in Software-Bibliotheken eingesetzt, die wiederum direkt in bestehende Softwareprojekte eingebunden werden können. Hier geschehen durch die Einbindung der modifizierten MotorShield-Bibiothek AFMotorMod.h durch *#include <AFMotorMod.h>* in Kopfteil des Quelltextes.

[27] Im Quelltext ist die Funktion *adjust()* zur weiteren Vereinfachung und besseren Lesbarkeit in zwei Funktionen *MoveMotor()* und *adaptValve()* aufgeteilt.

Die Funktion `adjust()` nimmt neben dem anzusteuernden Ventil, der Anzahl der Schritte[28], auch die Drehrichtung als Parameter entgegen. Die Drehrichtung wird für die bessere Lesbarkeit in `F` für `FORWARD`, was dem Schließen des Ventils entspricht und `B` für `BACKWARD`, was dem Öffnen des Ventils entspricht angegeben. Zusätzlich ermöglicht die Software noch den Status `R` für `RELEASE` der Schrittmotoren anzusteuern, was der Freistellung der Schrittmotorachse entspricht[29].

Listing 3.5: Quellcodeausschnitt: Motorsteuerung - Phasentrenner

```
1  // function MoveMotor() (1step)
2  int MoveMotor(int number, char direction) {
3    switch (direction) {
4      case 'F':                                  // F = FORWARD
5        switch (number) {
6          case 1:
7            motor_1.step(1, FORWARD, DOUBLE); break;
8          case 2:
9            motor_2.step(1, FORWARD, DOUBLE); break;
10       }
11       break;
12     case 'B':                                  // B = BACKWARD
13       switch (number) {
14         case 1:
15           motor_1.step(1, BACKWARD, DOUBLE); break;
16         case 2:
17           motor_2.step(1, BACKWARD, DOUBLE); break;
18       }
19       break;
20     case 'R':                                  // R = RELEASE
21       switch (number) {
22         case 1:
23           motor_1.release(); break;
24         case 2:
25           motor_2.release(); break;
26       }
27       break;
28   }
29 }
30
31
```

[28]Die hier verwendeten Schrittmotoren arbeiten mit 200 Schritten pro Umdrehung, das entspricht in Verbindung mit den Whitey SS-22RS2(-A) einer Änderung des Durchflusskoeffizienten von $\Delta K_V = 4{,}64 \cdot 10^{-6}\,m^3\,h^{-1}$ pro Schritt ($\equiv 4{,}64\,mL\,h^{-1} \equiv 0{,}077\,mL\,min^{-1} \equiv 77\,\mu L\,min^{-1}$).

[29]Schrittmotoren kennen 4 Betriebszustände: neben der Drehrichtung vorwärts und rückwärts gibt es zwei bewegungslose Zustände; normalerweise wird nach Erreichen einer Position das innere Magnetfeld so aufrecht erhalten, dass die Motorachse festgestellt und unbeweglich ist, eine Art Bremszustand. Will man hingegen eine Freistellung bzw. ein Auslaufen der Schrittmotorachse erreichen, muss man den Motor explizit in den Status *release* versetzen.

```
32 // function adaptValve() - valve adjustment
33 int adaptValve (int ValveToClose, int LimitValveToClose, int
       StepsToClose, int ValveToOpen, int LimitValveToOpen, int
       StepsToOpen, String ErrorMessage) {
34   if (motor_array[ValveToClose][2] > LimitValveToClose) {
35     for (int rpm = 0; rpm < StepsToClose; rpm++) {
36       if (digitalRead(motor_array[ValveToClose][3]) == 1 ) {
37         motor_array[ValveToClose][1] = 0;
38         motor_array[ValveToClose][2] = 0;
39       }
40       else {
41         if (motor_array[ValveToClose][1] == 1) {
42           MoveMotor(motor_array[ValveToClose][0], 'F');
43         }
44       }
45     }
46     MoveMotor(motor_array[ValveToClose][0], 'R');
47     motor_array[ValveToClose][2] = motor_array[ValveToClose][2] -
       StepsToClose;
48     Serial.print(ErrorMessage);
49     Serial.print(" - Ventil M");
50     Serial.print(motor_array[ValveToClose][0]);
51     Serial.print(" bei ");
52     Serial.println(motor_array[ValveToClose][2]);
53   }
54   if (motor_array[ValveToOpen][2] < LimitValveToOpen) {
55     for (int rpm = 0; rpm < StepsToOpen; rpm++) {
56         MoveMotor(motor_array[ValveToOpen][0], 'B');
57     }
58     MoveMotor(motor_array[ValveToOpen][0], 'R');
59     motor_array[ValveToOpen][2] = motor_array[ValveToOpen][2]+
       StepsToOpen;
60     Serial.print(ErrorMessage);
61     Serial.print(" - Ventil M");
62     Serial.print(motor_array[ValveToOpen][0]);
63     Serial.print(" bei ");
64     Serial.println(motor_array[ValveToOpen][2]);
65   }
66 }
```

Die Funktion `read_mcp3208()` aktiviert den Analog-Digital-Wandlerchip, indem Pin `D10` des Arduino, verbunden mit dem sogenannten *ChipSelect*-Eingang des MCP3208 auf Masse gelegt wird. Nun wird der auszulesende Kanal in einer Bitfolge[30] im LSB-Format[31] codiert und an den Chip gesendet. Dieser benötigt nun zwei Takte für die Messung und antwortet im dritten Takt mit dem Messwert des gewünschten Kanals. Dieser wird nach Beendigung der Verbindung zum MCP3208 der Rückgabewert der Funktion.

Listing 3.6: Quellcodeausschnitt: Auslesefunktion MCP3208 - Phasentrenner

```
1  // function read_MCP3208() - get values from lightbarriers
2  int read_MCP3208(int channel){
3   int mcpvalue = 0;
4   byte commandbits = B11000000;
5   commandbits|=((channel-1)<<3);
6
7  // select / activate MCP3208
8   digitalWrite(SELPIN,LOW);
9   for (int i=7; i>=3; i--){
10    digitalWrite(DATAOUT,commandbits&1<<i);
11    digitalWrite(SPICLOCK,HIGH);
12    digitalWrite(SPICLOCK,LOW);
13  }
14
15 // ignore two zero-bits
16  digitalWrite(SPICLOCK,HIGH);
17  digitalWrite(SPICLOCK,LOW);
18  digitalWrite(SPICLOCK,HIGH);
19  digitalWrite(SPICLOCK,LOW);
20
21 // read data from MCP3208
22  for (int i=11; i>=0; i--){
23    mcpvalue+=digitalRead(DATAIN)<<i;
24    digitalWrite(SPICLOCK,HIGH);
25    digitalWrite(SPICLOCK,LOW);
26  }
27
28 // deactivate MCP3208
29  digitalWrite(SELPIN, HIGH);
30
31 // return value read
32  return mcpvalue;
33 }
```

[30] Die Bitfolge zum Ansprechen des MCP3208 wird ausführlich im zugehörigen Datenblatt beschrieben.

[31] LSB-Format, engl. für *Least Significant Bit*, die Bits werden aufsteigend von rechts nach links gelesen, da das Bit 0 den niedrigsten Stellenwert hat.

Die sich wiederholende zentrale Programmschleife startet mit einer Zeitschleife, da, bedingt durch die meist niedrigen Flussraten, das System eine erhöhte Trägheit aufweist. So findet eine Anpassung der Ausflussmenge durch die Ventile nicht bei jeder Wiederholung der Schleife statt. Das Auslesen der Lichtschranken und die Bildung des Mittelwertes von 100 Messungen dauert circa eine Sekunde. Eine Regelung der Ventile im vorherigen Schleifendurchlauf wird folglich in den direkt folgenden Umläufen keine für das System messbare Änderung zeigen. Bei Flussraten im Bereich von einem Milliliter pro Minute hat sich empirisch ein Intervall von einer Minute zwischen der erneuten Anpassung der Ventile als vernünftig erwiesen. Ist die Bedingung der Zeitschleife erfüllt, beginnt das Auslesen der Lichtschranken in der im Initialisierungsbereich der Software festgelegten Häufigkeit, um daraus einen Mittelwert für jede Lichtschranke zu berechnen.

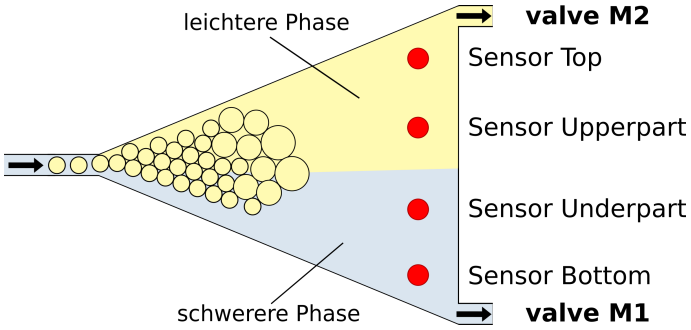


Abbildung 3.33: Benennung der Ventile und Lichtschranken im Programmablaufplan

Wird auf Höhe der untersten Lichtschranke (Sensor Bottom) die leichtere Phase detektiert, prüft die Software, ob das untere Auslassventil (M1) bereits geschlossen ist. Ist dem noch nicht so, wird das untere Auslassventil um 100 Schritte weiter geschlossen und parallel das obere Auslassventil (M2) um 100 Schritte weiter geöffnet. Das explizite Prüfen und Öffnen des anderen Ventils dient neben dem verbesserten Abfluss insbesondere der Verhinderung eines Druckaufbaus innerhalb der Trennkammer. Der dazu invertierte Ablauf findet statt, wenn an der obersten Lichtschranke (Sensor Top) die schwerere Phase gemessen wird. Befindet sich die Phasengrenzfläche dazwischen, beispielsweise ein wenig oberhalb der untersten Lichstschranke (Sensor Bottom), detektiert der zweite Sensor von unten (Sensor Underpart) die leichtere Phase, der unterste Sensor (Sensor Bottom) jedoch wie gewünscht die schwerere Phase. Jetzt findet nur eine geringe Anpassung der Ventile statt: das untere Ventil (M1) wird um 50 Schritte geschlossen, parallel, wenn es nicht schon vollständig geöffnet ist, das obere Ventil um 50 weitere Schritte geöffnet. Nach einer kurzen Wartedauer (delay) beginnt die Schleife erneut mit der Abfrage der Lichtschranken.

Im Anschluss daran sorgt eine Kaskade von Bedingungen für die korrekte Interpretation der Messwerte und die Steuerung der Ventile:

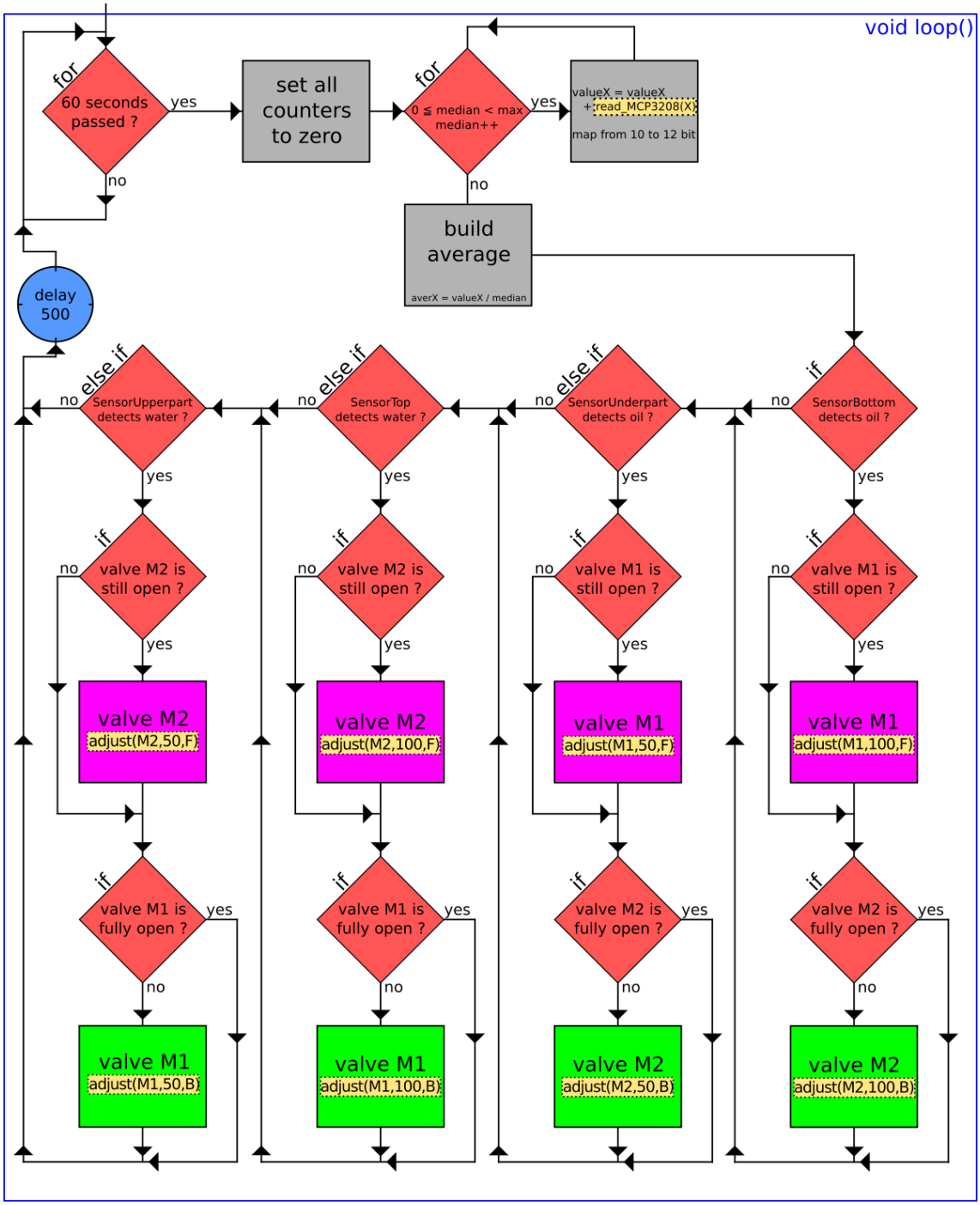


Abbildung 3.34: Programmablaufplan des Phasentrenners, Wiederholungsroutine

Listing 3.7: Quellcodeausschnitt: Wiederholungsroutine - Phasentrenner

```
void loop() {
 uint8_t i;

 for (int count = 0; count < seconds; count++) {
// only one measurement within given interval
   if (count == 0) {
     Serial.println("Messung gestartet.");
     memset(total, 0, sizeof(total));

// read values from MCP3208
     for (int thisReading = 0; thisReading < numReadings; thisReading
    ++) {
       for (i = 0; i < (sizeof(total)/sizeof(long)); i++) {
         value_MCP3208 = read_MCP3208(i+1);
         total[i] = total[i] + value_MCP3208;
       }
       delay(50);
     }

// build average of readings for every sensor
     for (i = 0; i < (sizeof(total)/sizeof(long)); i++) {
       average[i] = total[i] / numReadings;
     }

// only move valves when set to active and calibrated
     if (valveActive == 1 && valveCalibration == 1) {

// close M1 when detecting rarer phase on bottom-lightbarrier
// additionally open M2
       if (average[6] > sensorValue[3][1]) {
         adaptValve (0, 99, 100, 1, maxOpenValue, 100, "Phasengrenze
    viel zu niedrig!");
       }

// open M1 when detecting rarer phase on underpart-lightbarrier
// additionally close M2
       else if (average[4] > sensorValue[2][1]) {
         adaptValve (0, 49, 50, 1, maxOpenValue, 50, "Phasengrenze zu
     niedrig!");
       }

// close M2 when detecting denser phase on top-lightbarrier
// additionally open M1
       else if (average[0] < sensorValue[0][0]) {
         adaptValve (1, 99, 100, 0, maxOpenValue, 100, "Phasengrenze
    viel zu hoch!");
       }
```

```
44
45 // open M2 when detecting denser phase on upperpart-lightbarrier
46 // additionally close M1
47         else if (average[2] < sensorValue[1][0]) {
48           adaptValve (1, 49, 50, 0, maxOpenValue, 50, "Phasengrenze zu
       hoch!");
49         }
50       }
51       else {
52         Serial.println("Ventilanpassung deaktiviert!");
53       }
54
55 // data output on serial console
56       Serial.print("Messwerte: \t");
57       for (i = 0; i < (sizeof(total)/sizeof(long)) - 1; i++) {
58         Serial.print(average[i]);Serial.print("\t"); // - \t");
59       }
60       Serial.print(average[(sizeof(total)/sizeof(long)) - 1]);
61       Serial.println();
62     }
63
64 // additional delay for gaining one cycle per second
65     delay(500);
66     Serial.print(".");
67  }
68  Serial.println();
69 }
```

Der gesamte Softwareablauf sorgt für eine autonome Ventilsteuerung des initial kalibrierten Phasentrenners über eine beliebig lange Zeit hinweg. Selbst der thermische Shift der Phototransistoren bei längerer Laufzeit und stärkerer Erwärmung wird innerhalb gewisser Grenzen durch die Software toleriert und verursacht kein Fehlverhalten des Phasentrenners. Selbst eine Änderung des Volumenverhältnisses der zulaufenden Phasen auf 100:0 ist mit der oben vorgestellten Software abbildbar, da nach dem Ausbleiben des Zustromes einer der beiden Komponenten dessen Auslassventil nach kurzer Zeit geschlossen wird, sodass der gesamte Zustrom auch direkt über das noch offene Auslassventil abfließt.

3.2.4.1 Optische Phasenerkennung

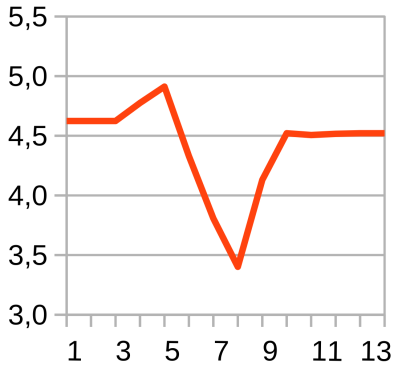
Dem Entwurf aus [40] entsprechend, operiert die auf dem Arduino laufende Software so, dass sie die Messwerte der vier Transmissionslichtschranken einzeln mit einem zuvor individuell ermittelten Schwellenwert vergleicht und so die Vorhersage trifft, welche Phase sich zwischen der jeweiligen Lichtschranke befindet. Somit lässt sich daraus die Position der Phasengrenze innerhalb der Trennkammer angeben. Der Durchfluss der Auslassventile oben und unten wird daraufhin abhängig der Postion der Phasengrenze geregelt. Sollte gegenüber der vorhergehenden Messung eine andere Position der Phasengrenze ermittelt worden sein, öffnet respektive schließt die Software durch Ansteuerung der mit dem Arduino verbundenen Motorsteuerungsplatine die schrittmotorgetriebenen Nadelventile. Die Kalibrierung der Ventile findet während der Startphase des Phasentrenners statt und wird bei einem Neustart wiederholt.

Die hier beschriebene Ermittlung der Position der Phasengrenze basiert auf dem Unterschied in der optischen Dichte der zu trennenden Phasen und erfordert dazu eine Kalibrierung der Software über den Erwartungswert jeder der beiden Phasen, welchen die Phototransistoren bei der Messung liefern. Der Mittelwert beider Messwerte entspricht dem oben genannten Schwellenwert: liegt die Messung unterhalb dessen, befindet sich zwischen der Lichtschranke die Phase mit der höheren Lichtabsorption bzw. -streuung; liegt der Messwert über dem Schwellwert, handelt es sich um die Phase mit der geringeren Lichtabsorption bzw. -streuung.

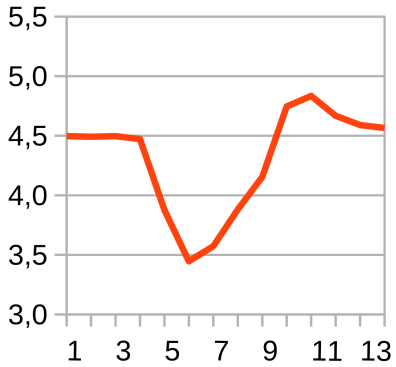
Die Kalibrierung erfolgt entweder vor der Nutzung des Phasentrenners durch Hinterlegen der Schwellenwerte in dem Kopfteil der Software (siehe Quellcode-Listing 3.4) oder durch eine externe Schwellenwertermittlung. Dies geschieht durch die Erweiterung des Phasentrenners um zwei Küvetten identischer Breite und Beschaffenheit wie die Trennkammer des Phasentrenners, beide ausgestattet mit einer eigenen Transmissionslichtschranke und befüllt mit entweder der unteren oder der oberen reinen Phase. Zusätzlich zur einmaligen externen Kalibrierung besteht die Möglichkeit der Inline-Kalibrierung durch Nutzung von Durchflussküvetten identischer Breite und Beschaffenheit wie die Trennkammer des Phasentrenners, angeschlossen an jeweils den oberen bzw. unteren Ausgang des Phasentrenners. So wird selbst bei zeitlich veränderlicher Lichtabsorption der jeweiligen Phase, beispielsweise durch variierende Produktkonzentration, laufend der Schwellenwert angepasst. Hier ist jedoch auch eine initiale Schwellwertermittlung im Vorfeld (intern oder extern) unabdingbar.

3.2.4.2 Optische Phasengrenzflächenerkennung

Die optische Phasenerkennung versagt jedoch bei der Unterscheidung zweier Phasen mit zu geringen Absorptionsunterschieden. Um dieses Problem zu umgehen, basiert die optische Phasengrenzflächenerkennung auf der Intensitätsveränderung des auf den Phototransistor einfallenden Lichts am Meniskus der Phasengrenze. Abhängig von der Oberflächenspannung der beiden zu trennenden Phasen und deren Hydrophilie (Benetzbarkeit gegenüber der Trennkammer aus Glas) entspricht der Meniskus an der Phasengrenze der Form einer senkrecht halbierten konvexen Linse. Das sorgt während des Durchtritts des Lichtstrahls durch die Phasengrenzfläche für eine Beeinflussung des Strahlengangs der Lichtschranke. Überträgt man die gemessenen Werte in ein Koordinatensystem, ergibt sich ein charakteristisches Bild, aus dem die Richtung und die Geschwindigkeit des Phasengrenzflächendurchtritts ablesbar ist.



(a) Zulauf von 1 mL/min FC-40®, steigende Phasengrenze

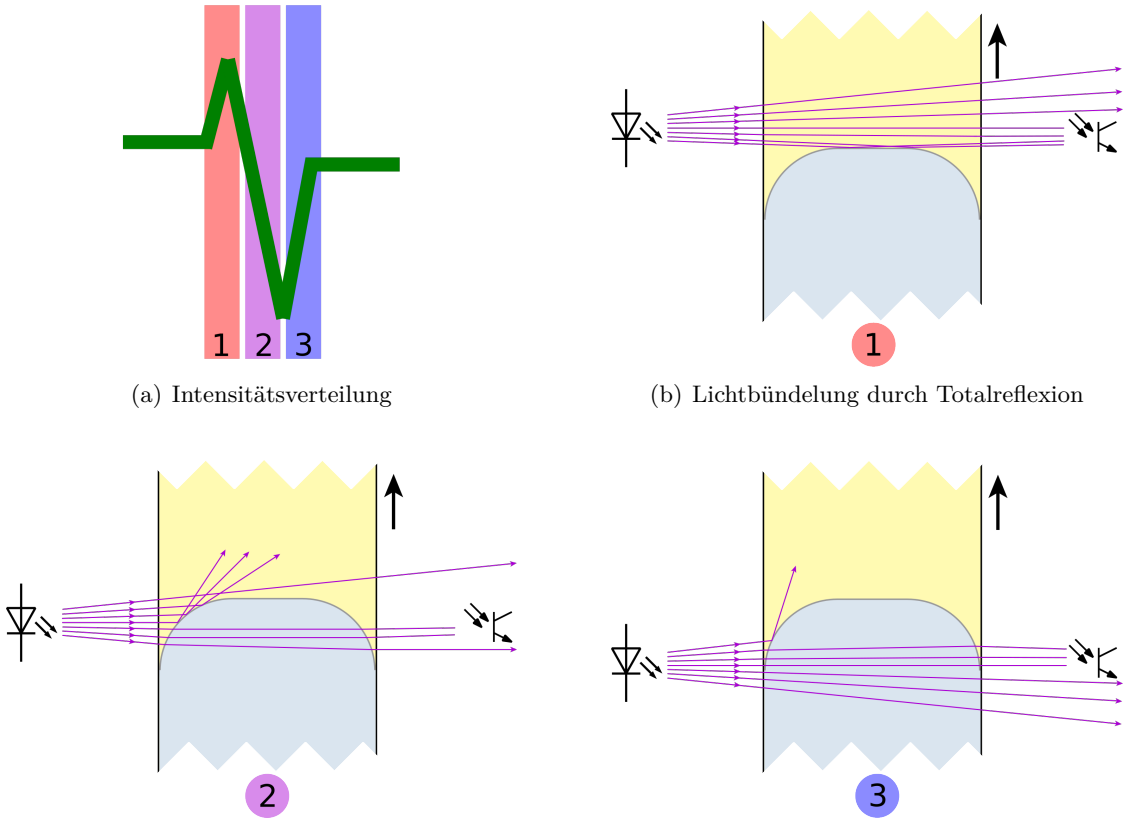


(b) Zulauf von 1 mL/min Wasser, sinkende Phasengrenze

Abbildung 3.35: Zeitlicher Lichtintensitätsverlauf (in Volt) während des Durchlaufs der Phasengrenze durch eine Lichtschranke für ein Zweiphasensystem Wasser/FC-40®

Die Messung in Abbildung 3.35 (a) zeigt eine kurze Steigerung der gemessenen Lichtintensität, die höher liegt, als die gemessene Intensität der reinen Phasen. Dies lässt sich durch die Totalreflexion des UV-Lichts am flachen Boden des Meniskus erklären. Abbildung 3.36 (b) zeigt den entsprechenden vereinfachten Strahlengang der UV-LED durch den Meniskus hin zum Phototransistor. Licht aus dem Lichtkegel der UV-LED, welches im spitzen Winkel auf die Phasengrenzfläche auf trifft, wird dort reflektiert und erreicht zusätzlich zum ungestreuten UV-Licht den Phototransistor.
Direkt darauf folgt ein Einbruch der gemessenen Intensität, hervorgerufen wird dieser durch die Brechung und Streuung des einfallenden Lichts durch den Kontaktwinkel der Phasengrenze zu den Trennkammerrändern hin, wie an Hand des vereinfachten Strahlengangs in Abbildung 3.36 (c) dargestellt. Erst wenn der Lichtkegel das Ende des Meniskus erreicht, stellt sich wieder das lineare Durchtrittsbild der Lichtstrahlen ein und die Intensität erreicht einen konstanten Wert, abhängig von der optischen Dichte des Mediums

(Abbildung 3.36 (d)).



(a) Intensitätsverteilung

(b) Lichtbündelung durch Totalreflexion

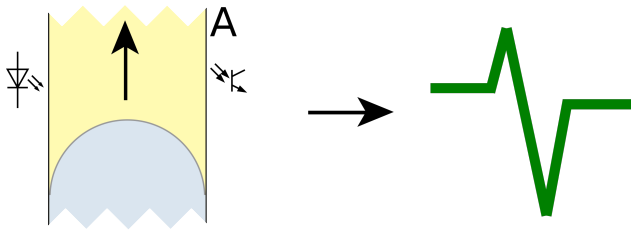
(c) Intensitätsverringerung durch Streuung

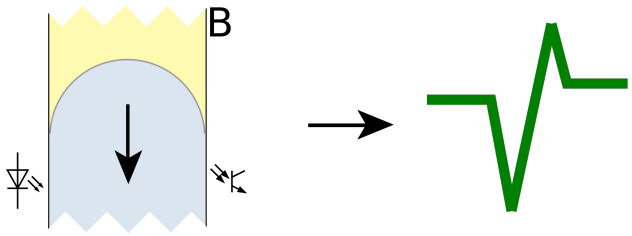
(d) Anstieg entsprechend der optischen Dichte

Abbildung 3.36: Intensitätsverteilung durch Reflexion und Streuung während des Lichtdurchtritts durch den Meniskus

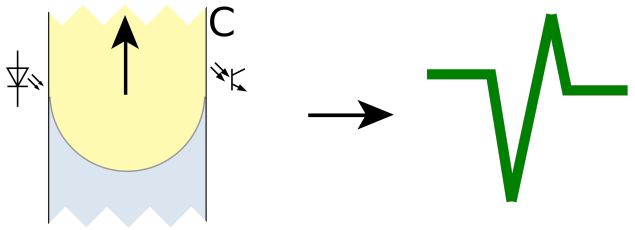
Neben der Erkennung des Durchtritts der Phasengrenze durch die Lichtschranke lässt sich mit dieser Methode auch direkt die Durchtrittsrichtung (steigende oder fallende Phasengrenze) innerhalb der Trennkammer ablesen, sofern bekannt ist, ob sich zwischen den zu trennenden Phasen ein konkaver (nach unten gewölbter) oder konvexer (nach oben gewölbter) Meniskus ausbildet[32]. Abbildung 3.37 zeigt eine Übersicht über die möglichen Figuren, die eine Auftragung der Messung in einem Koordinatensystem ergibt, sofern das obere Medium optisch dichter als das untere Medium ist. Ein Beispiel für den Fall (a) ist das System Wasser - FC-40®, wobei Wasser als Medium geringerer Dichte die obere Phase darstellt. Das typische Beispiel für Abbildung 3.37 (b) stellt das System Wasser - Toluol dar.

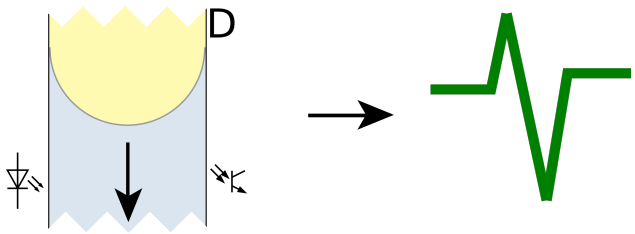
[32]Die Ausbildung des Meniskus ist von der Adhäsion der Phasen zu der Wand der Trennkammer abhängig. In dem verwendeten Glaskörper verhält sich die Wasserphase wandadhäsiv, sodass deren Position die Vorhersage des Meniskus erlaubt.





(a) Fall A (bspw. Wasser - FC-40®)





(b) Fall B (bspw. Toluol - Wasser)

Abbildung 3.37: Übersicht über die vier möglichen Messverläufe, die sich ergeben können. Detaillierte Erläuterungen dazu befinden sich im Text.

Abbildung 3.38 zeigt eine Messung der Spannung einer Lichtschranke im Verlauf der Füllung des Phasentrenners mit unterschiedlichen Flüssigkeiten: vorgelegt im Phasentrenner war FC-40®, welches mit $1\,\mathrm{mL\,min^{-1}}$ Toluol verdrängt wurde. Nach Befüllung mit Toluol wurde dieses mit $1\,\mathrm{mL\,min^{-1}}$ Wasser aus dem Phasentrenner verdrängt. Die gelbe Linie zeigt die gemittelte Spannung für FC-40® (4,63 Volt), die grüne Linie die gemittelte Spannung für Toluol (4,83 Volt) und die blaue Linie die gemittelte Spannung für Wasser (4,65 Volt). Man erkennt deutlich, dass der Spannungsunterschied zwischen Wasser und Toluol ausreichend ist, um die Phasen sowohl durch den signifikanten Unterschied der optischen Dichte leicht voneinander zu unterscheiden, als auch mittels des charakteristischen

Durchtrittsmusters der Phasengrenzfläche diese Unterscheidung zu treffen. Da der gemessene Spannungsunterschied zwischen der Wasserphase und der FC-40®-Phase jedoch nur sehr gering ist, ist eine Phasenerkennung über die optische Dichte beider Substanzen im Rahmen der Messungenauigkeit nicht sicher möglich und muss über die Erkennung des Phasengrenzflächendurchtritts erfolgen. Ein weiteres Beispiel hierfür ist die Trennung von Wasser und Octanol, bei denen sich die Messwerte der Lichtabsorption bei den verwendeten UV-LEDs um weniger als 0,5 % voneinander unterscheiden.

Abbildung 3.38: Vergleich der gemessenen Spannungen (in Volt) für Wasser, Toluol und FC-40® anhand einer Messreihe (siehe Text)

Zusammenfassend lässt sich festhalten:

- Die optische Phasenerkennung ermöglicht den einfachen Zugang zur dynamischen Phasenseparation, jedoch lassen sich hiermit nicht alle Begebenheiten abdecken.
- Erst die Methode der optischen Phasengrenzflächenerkennung erlaubt, gerade bei ähnlicher Lichtabsorption beider Phasen im Wellenlängenbereich der verwendeten UV-LEDs, die exakte Erkennung der Position der Phasengrenze. Zusätzlich erhält man noch die wichtige Information, in welche Richtung sich die Phasengrenze gerade bewegt. So können auch Phasen mit geringsten Absorptionsunterschieden zuverlässig unterschieden werden.
- Einzig zur Differenzierung zwischen zwei lichtundurchlässigen Phasen sind beide Erkennungsmethoden gleichermaßen ungeeignet.

3.3 Fraktionierer

Neben der im vorherigen Kapitel beschriebenen Trennung zweier oder mehrerer Phasen ist zusätzlich eine zeitliche Trennung verschiedener Fraktionen eines Reaktionslaufes von Interesse, insbesondere dann, wenn die Reaktionsbedingungen (Temperatur, Konzentrationsverhältnisse, Verweizeit, etc.) automatisiert zeitlich variiert werden. Auf dem Markt sind unterschiedliche Systeme zur Fraktionssammlung erhältlich. Die meisten Systeme eint, dass diese keine individuelle Anpassung der Intervalldauer einer jeden Fraktion zulassen. In der Mikrofluidik ist es aber unumgänglich, flexible Intervalle beim Sammeln der Proben zu realisieren, da beispielsweise beim Parameterscreening einer Reaktion eine Änderung der Gesamtflussrate um die Hälfte die Dauer zum Sammeln eines vergleichbaren Fraktionsvolumens verdoppelt. Auch wird normalerweise die erste Fraktion nach dem Anfahren bis zur Equilibrierung der Reaktion separiert und verworfen. Gleichsam kann eine Parameteränderung im Lauf der Reaktion stattfinden, was eine Trennung des Übergangs von dem vorherigen Lauf erfordert, bis sich das Reaktionsgleichgewicht wieder eingestellt hat.

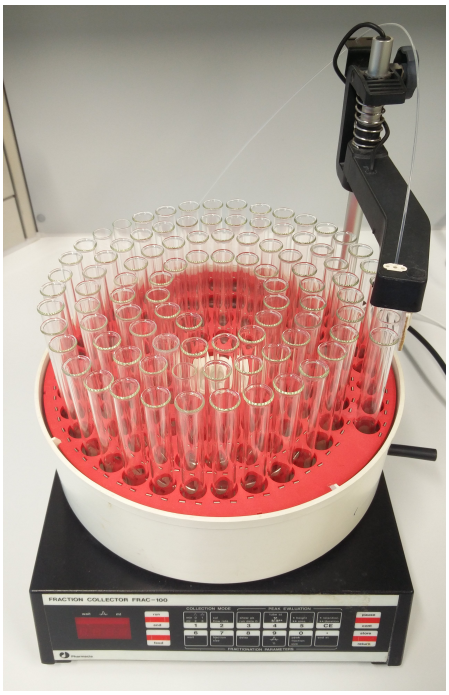


Abbildung 3.39: Modifizierter Säulenautomat FRAC-100

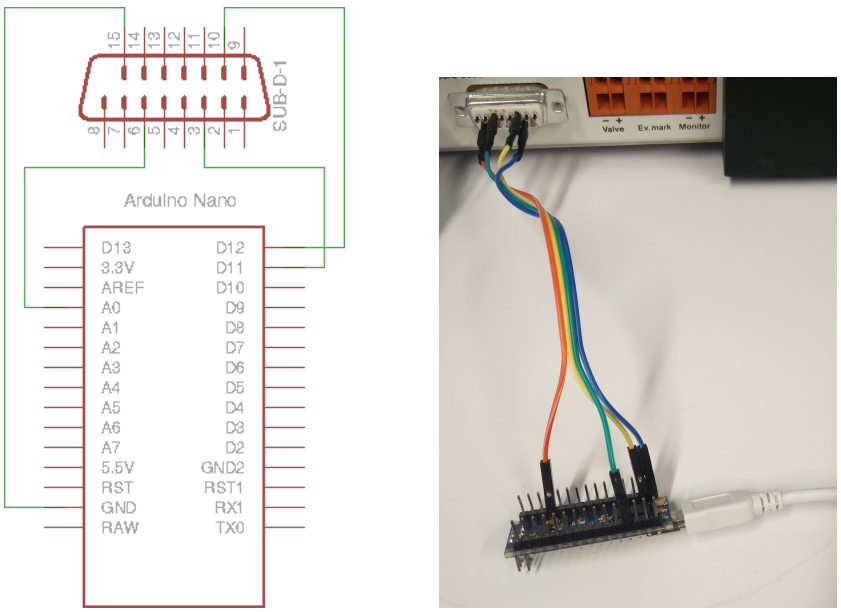
Die häufigste Lösung dieses mikrofluidischen Fraktionierungsproblems besteht im manuellen Austausch des Auffanggefäßes am Ausgang des Reaktionssystems. Um ein System mit möglichst wenig erforderlicher Benutzereinbindung beziehungsweise vollständig autonom laufen zu lassen, ist ein automatisches Fraktionierungssystem notwendig, das direkt durch die zentrale Kontrolleinheit gesteuert wird, welche auch die Reaktionsparameter überwacht.

3.3.1 Funktionsweise

Es wurde ein im Labor vorhandener Säulenautomat[33] um einen Mikrocontroller erweitert, der dessen Steuerung übernimmt und entweder in vorher festgelegten Zeitintervallen oder auf entsprechende Befehle durch die zentrale Steuerungseinheit eine neue Fraktion beginnt. Der Säulenautomat FRAC-100 verfügt über eine TTL[34]-fähige 15-polige Schnittstelle, die durch den Hersteller vollständig dokumentiert ist[50].

3.3.2 Technische Realisierung

Der verwendete Arduino Nano[35] bildet das nötige Gegenstück zu der Schnittstelle des Säulenautomaten und übersetzt die Befehle der zentralen Steuerungseinheit in elektronische Signale. Durch seine kleine Bauform passt er in handelsübliche Gehäuse, welche nicht breiter als der 15-polige Sub-D-Stecker sind.



(a) Schaltplan

(b) Arduino Nano verbunden über die 15-polige Sub-D-Buchse mit dem Säulenautomaten

Abbildung 3.40: Schaltplan und Aufbau der Arduino-Steuerungseinheit (ohne Gehäuse)

[33]Bei dem verwendeten Säulenautomaten handelt es sich um einen Fraction Collector FRAC-100 von Amersham Biosciences AB, Schweden, Datenblatt siehe Anhang B, der über einen Drehteller bis zu 95 schneckenförmig angeordnete Reagenzgläser (⌀ 10-18 mm, 50-180 mm Länge) nacheinander befüllt.

[34]TTL: Transistor-Transistor-Logik; elektrotechnischer Standard für logische Schaltungen in der seriellen Übertragung von *low*- und *high*-Pegeln auf 5 V-Basis.

[35]Der Arduino Nano verfügt auch über einen Atmel ATMEGA328P vergleichbar dem Arduino Uno, die Platine wurde aber auf eine möglichst kleine Bauform optimiert.

Für die Steuerung des Säulenautomaten sind nur 3 der 15 möglichen Leitungen des Sub-D-Steckers nötig. Neben der unumgänglichen Masseleitung (Ground, Pin `15`, orange) handelt es sich zusätzlich noch um den Pin `3` (Feed, gelb), um den Fraktionswechsel zu initiieren und Pin `10` (Operable, blau), der der Säulenautomaten in den Modus der externen Steuerung versetzt. Mittels dem im Folgenden zusätzlich verwendeten, optionalen Pin `6` (Event mark, grün) erfolgt die Rückmeldung des Säulenautomaten und wird durch den Mikrocontroller verarbeitet.

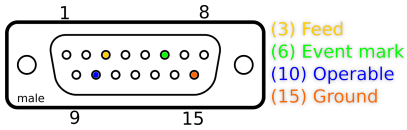


Abbildung 3.41: Steckerbelegung (siehe Text)

Um eine neue Fraktion zu beginnen, muss zuerst Pin `10` mit Masse (logisch *low*) verbunden werden, um den Säulenautomaten für den Empfang externer Steuerungssignale vorzubereiten. Nun wird durch ein 5 V-Signal von mindestens 50 ms Dauer an Pin `3` der Wechsel des Reagenzglases veranlasst. Dies erfolgt mittels eines Pulses an einem der digitalen Ausgänge des Arduino Nano, in diesem Falle Pin `D11` (siehe Abbildung 3.40 (a)). Der steuernde Arduino Nano misst fortwährend über seinen analogen Eingang `A0` das *Event mark*-Signal des Säulenautomaten. Im Falle des erfolgreichen Reagenzglaswechsels wird der standardmäßige *high*-Pegel des Signals für 200 ms zum *low*-Pegel. Diese Rückmeldung des Säulenautomaten wird entweder vom Arduino Nano direkt oder an einer zentralen Steuerungseinheit verarbeitet oder an einen Computer zur Anzeige weitergeleitet.

3.3.3 Software

Im Folgenden wird die auf dem Arduino Nano laufende Software zur Intervall-Steuerung des Säulenautomaten vorgestellt. Neben der Anzahl der Reagenzgläser (tube), die beim einprogrammierten Lauf genutzt werden sollen, wird im Kopfteil des Programms jeweils noch die Zeitspanne (interval) in Sekunden angegeben, in der im entsprechenden Reagenzglas eine Fraktion gesammelt wird. Nach Ablauf der Zeit wird das Karussell des Säulenautomaten um genau ein Reagenzglas weiter gedreht und die nächste vorher festgelegte Zeit abgewartet.

Für den Arduino entwickelte Programme gliedern sich in einen setup()-Teil und einen loop()-Teil. Der setup()-Teil wird einmalig beim Start der Software ausgeführt, der loop()-Teil, wie der Name schon sagt, dauerhaft bis zu einer expliziten Austrittsbedingung wiederholt. Die Software zur Steuerung des Säulenautomaten benötigt nur den setup()-Teil, da nach der Abarbeitung aller Programmschritte die Programmausführung beendet wird.

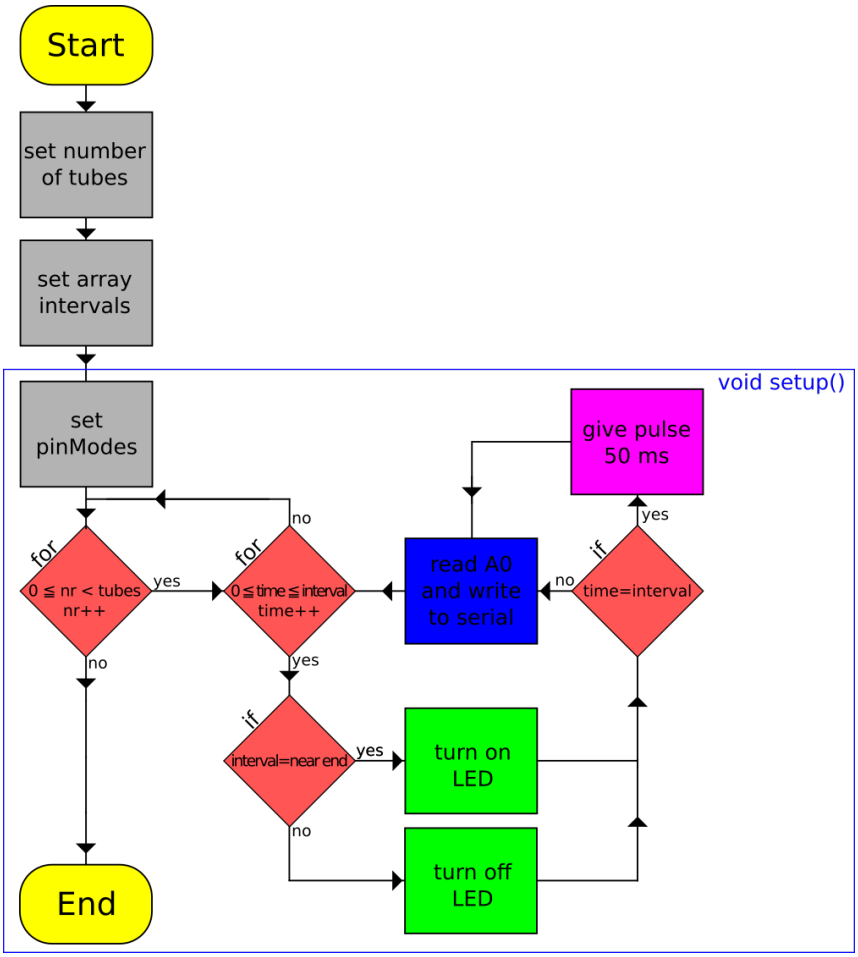


Abbildung 3.42: Programmablaufplan der Software zur Steuerung des Säulenautomaten

Zuerst wird zur Rückmeldung und Programmablaufüberwachung eine serielle Konsole mit 9600 Baud geöffnet. An diese wird im weiteren Verlauf des Programms direkt der Pegel des Pulses und der Pegel des ersten analogen Eingangs `A0` mit dem Rückmeldesignal des Säulenautomaten weitergeleitet[36]. Die eigentliche Softwareroutine prüft zuerst, ob nicht bereits die gewünschte Anzahl Reagenzgläser abgearbeitet wurde. Ist dem nicht so, startet die Zeitschleife für die vorgegebene Zeit. Mit dem Erreichen des Endes des Intervalls wird Pin `D11` für 60 ms mit 5 V versorgt (logisch *high*), um das Probenkarussell ein Reagenzglas weiter zu drehen. Zur visuellen Rückmeldung eines Pulses wird einige Takte davor die eingebaute LED eingeschaltet, da die kurze Aktivierung der LED nur für die 60 ms Dauer des Pulses für das menschliche Auge nicht wahrnehmbar ist. Am Fuß der Schleife erfolgt die Rückmeldung über die serielle Schnittstelle.

[36] Über den in die Arduino IDE integrierten seriellen Plotter erfolgt eine grafische Überwachung der Programmfunktion zur Laufzeit. Sie steht seit Version 1.6.6 der Arduino IDE zur Verfügung.

Listing 3.8: Programmbeispiel zur Ansteuerung des Fraktionierers

```
// Number of tubes, maximum 90
int tube = 10;

// Interval in seconds, minimum 5 seconds
int interval[90] = { 10,10,10,10,10,10,10,10,10,10,
                     10,10,10,10,10,10,10,10,10,10,
                     10,10,10,10,10,10,10,10,10,10,
                     10,10,10,10,10,10,10,10,10,10,
                     10,10,10,10,10,10,10,10,10,10,
                     10,10,10,10,10,10,10,10,10,10,
                     10,10,10,10,10,10,10,10,10,10,
                     10,10,10,10,10,10,10,10,10,10,
                     10,10,10,10,10,10,10,10,10,10 };
int return_int = 0;
int return_ext = 0;

void setup() {
  Serial.begin(9600);
  pinMode(11, OUTPUT);        // FEED-Pin (3)
  pinMode(12, OUTPUT);        // OPERABLE-Pin (10)
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(11, LOW);
  digitalWrite(12, LOW);      // keep OPERABLE-Pin active (low)
  delay(1000);
  for (int nr = 0; nr < tube; nr++) {
    for (long time = 0; time <= (interval[nr]*10); time++) {
      // just for visual feedback turn on LED
      if (time > ((interval[nr]*10)-3)) {
        digitalWrite(LED_BUILTIN, HIGH);
      }
      else {
        digitalWrite(LED_BUILTIN, LOW);
      }
      // pulse high for 60ms
      if (time == (interval[nr]*10)) {
        digitalWrite(11, HIGH);
        return_int = 1024;
      }
      else {
        digitalWrite(11, LOW);
        delay(40);
        return_int = 0;
      }
      delay(60);
```

```
48
49          return_ext = analogRead(A0);
50          // write to serial port for serial plotter in Arduino IDE
51          Serial.print(return_int);
52          Serial.print(" ");
53          Serial.println(return_ext);
54        }
55      }
56      exit;
57  }
58  void loop() { }
```

Vergleichbar mit der Software zur Intervallsteuerung des Fraktionierers nimmt die interaktive Software zur individuellen Steuerung des Fraktionierers über die integrierte serielle Schnittstelle des Arduino Nano neben der Ausgabe von Rückmeldewerten auch ein Signal zum Probenwechsel entgegen und verarbeitet es entsprechend. Alternativ, beispielsweise beim Anschluss an andere Mikrocontroller, erfolgt eine Übertragung des Signals auch über die direkte Verbindung zweier Pins der Mikrocontroller. Auf das Auslesen und Verarbeiten von Daten der seriellen Schnittstelle wird dadurch zu Gunsten performanterer in Echtzeit arbeitender Software verzichtet.

Die Erweiterung des FRAC-100 um einen Mikrocontroller öffnet das Feld der flexiblen Probensammlung, in dem nicht nur im Rahmen des Parameterscreenings einer Reaktion die Zeiten zum Sammeln gewünschter Probenvolumina an Spülintervalle oder veränderliche Flussraten angepasst wird, sondern auch erstmals die vollständige Integration der abschließenden Probensammlung in autonome mikrofluidische Systeme wie das Internet of Lab erfolgt.

Kapitel 4

Steuerungssoftware

Neben den bereits in den vorherigen Kapiteln vorgestellten Softwareentwicklungen, welche direkt auf den jeweiligen Arduino Mikrocontrollern lauffähig sind, war die Entwicklung einer zentralen Steuerungssoftware notwendig, welche neben der Steuerung und Überwachung der Aktoren, die von den Sensoren bereitgestellten Informationen ausliest und verarbeitet, sowie die zentrale Steuerung der angeschlossenen Pumpen übernimmt. In diesem Kapitel wird der aktuelle Releasestand einer solchen zentralen Steuerungssoftware, deren überwiegende Aufgabe derzeit noch in der erweiterten Pumpensteuerung liegt, vorgestellt.

4.1 Pumpensteuerung

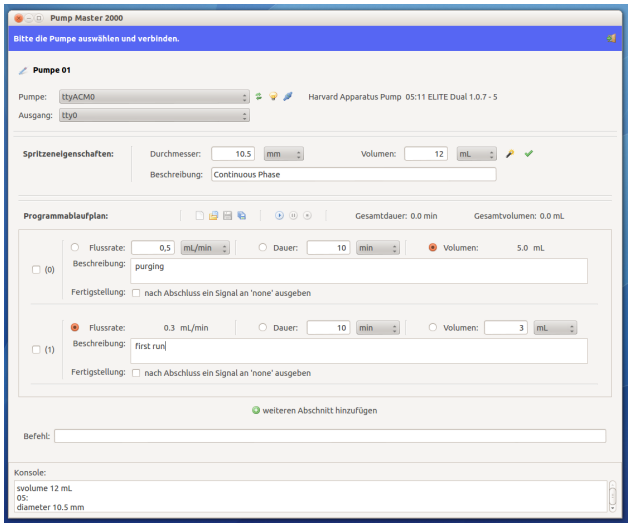


Abbildung 4.1: Frontend der Pumpensteuerung

Um über die gesamte Anlagenlaufzeit gleichbleibende Umsätze und Ausbeuten bei chemischen Reaktionen im kontinuierlichen Fluss zu erhalten, liegt der Fokus neben dem Reaktor, in dem die Reaktion stattfindet, besonders auf der Rohstofflieferkette, die entsprechend einer vorherigen Definition ohne jegliche Abweichung alle zur Reaktion benötigten Edukte anliefern muss. Hierzu finden in der Mikrofluidik im Labormaßstab neben HPLC-Pumpen immer öfter die aus der Medizintechnik bekannten Spritzenpumpen Verwendung, welche mit frei wählbarer Flussrate ein bestimmtes Volumen aus einer Spritze befördern. Viele der im Laborbedarf erhältlichen Modelle erlauben neben dem Einsatz spezialisierter Spritzen aus Glas oder

Edelstahl auch die Nutzung von Einmalspritzen aus PP/PE, wie sie in der Medizintechnik verwendet werden und welche in unterschiedlichen Volumina im Handel erhältlich sind.

Für alle in dieser Arbeit durchgeführten Reaktionen und Versuche kamen entweder Spritzenpumpen von Harvard Apparatus[1] oder aus dem Hause IMM[2] in Kombination mit Einmalspritzen[3] zum Einsatz. Beide Typen von Spritzenpumpen erlauben eine externe Steuerung durch einen Computer über den USB-Anschluss und bieten sich damit für einen Einsatz in mikrofluidischen Systemen, die zentral durch einen Computer gesteuert und überwacht werden, an. Die Rohstofflieferkette stellt bei kontinuierlichen Systemen immer den Startpunkt mindestens eines Teils des Systems dar und gehört zu den Komponenten des Gesamtsystems, an denen ohne technischen Umbau der Anlage im laufenden Betrieb Reaktionsparameter verändert werden können. So zählt neben computergesteuerten Ventilen auch die Pumpensteuerung zu den Aktoren eines mikrofluidischen Systems. Gerade bei chemischen Reaktionen, für die im labortechnischen Versuchsaufbau ein Parameterscreening stattfinden soll, um möglichst die Reaktionsparameter[4] zu ermitteln, bei denen einen Verfahren am effizientesten läuft. Dies sind beispielsweise die Reaktionsparameter, bei denen der maximale Umsatz ohne Nebenreaktionen möglich ist oder die Temperatur, bei der die Reaktion gerade noch stattfindet. Für ein solches Parameterscreening ist es wichtig, dass eben diese direkt beeinflussbaren Reaktionsparameter an zentraler Stelle gesetzt werden und mittels entsprechender Algorithmen ebenda im Laufe des Screenings direkt angepasst werden.

Für diverse Spritzenpumpentypen existiert herstellereigene Software, mit welcher die Spritzenpumpen in engen Grenzen betrieben werden können. Diese Software genügt aber nicht vollumfänglich den für eine zentrale Softwaresteuerung notwendigen Kriterien:

- Programmierbarkeit durch variable Programmablaufpläne
- Möglichkeit, externe Ereignisse auszulösen
- Ansteuerung unterschiedlicher Pumpentypen (Herstellerunabhängigkeit)
- freie Betriebssystemwahl des zentralen Steuerungscomputers
- Einbindung weiterer Sensormesswerte in die Protokolldatei (Datenlogger)
- Client-Server-Architektur, Trennung von Frontend und Backend

[1] Harvard Apparatus Pump 11 Elite, Datenblatt siehe Anhang B

[2] Eigenentwicklung des ehem. Institut für Mikrotechnik Mainz (IMM), heute Fraunhofer IMM

[3] Injekt®Solo-Einmalspritzen von B.Braun; 2, 5, 10 und 20 mL; zentrischer Luer-Lock-Konus

[4] Zu den wichtigsten direkt beeinflussbaren Reaktionsparametern zählen die Reaktionstemperatur, der Druck, das Verhältnis der eingesetzten Edukte zueinander und die Verweilzeit.

Im Rahmen dieser Arbeit wurde die Programmierung einer Software, die die oben genannten Kriterien erfüllt, begonnen. Als Programmiersprache wurde Python[5] gewählt, da diese Programmiersprache auf allen gängigen Betriebssystemen (Stand 2018) interpretiert werden kann und ihre volle Funktion auch auf speziellen Kleinstcomputern wie dem Raspberry Pi gewährleistet ist. So läuft auf einem solchen Kleinstcomputer das Backend in Form der zentralen Steuereinheit des mikrofluidischen Reaktionssystems, benutzerseitig wird das Frontend beispielsweise per Webinterface bereitgestellt. Dies ermöglicht eine sogenannte Remotesteuerung, bei der der Bediener nicht mehr zwingend am Ort des Geschehens sein muss, sondern das System von einem beliebigen im gleichen Netzwerk oder mit dem Internet verbundenen Gerät aus steuert.

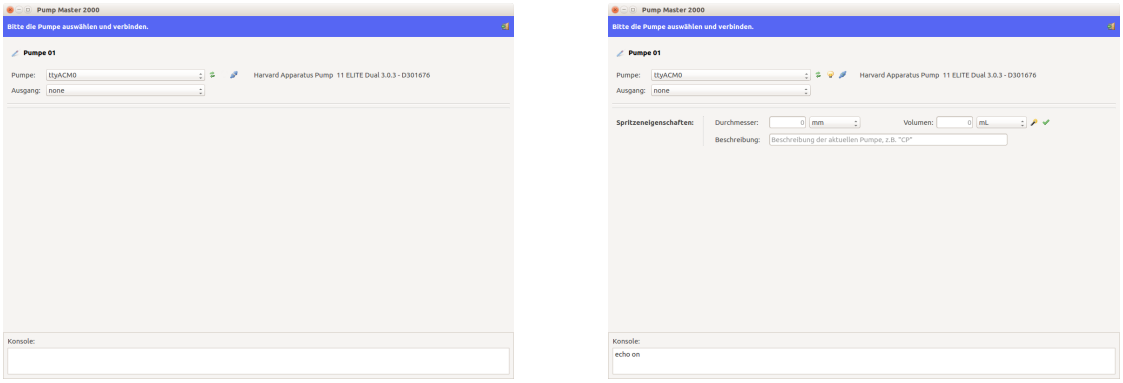
Die hier vorgestellte Software bindet beliebige Spritzenpumpentypen, welche eine Ansteuerung per serieller Schnittstelle ermöglichen, ein, indem in der Konfiguration des Backends im Kopfteil des Backendprozessquellcodes von der standardmäßig verwendeten Harvard Apparatus Pump 11 Elite abweichende Parameter zur seriellen Steuerung hinterlegt werden können. Zum Ausführen der Software ist die Installation einer Python3-Umgebung erforderlich. Zusätzlich müssen die Python3-Module `numpy`, `serial`, `pyside`, `pyside-qtgui` installiert sein, um die Verbindung zu den Spritzenpumpen via serieller Schnittstelle und die grafische Ausgabe des Frontends zu ermöglichen. Der aktuelle Releasestand der Spritzenpumpensoftware *Pump Master 2000* ist ohne weitere Anpassungen derzeit direkt unter Linux oder einem vergleichbaren unixoiden Betriebssystem[6] läuffähig, zur Nutzung unter Microsoft Windows muss die Abfrage der seriellen Schnittstelle angepasst werden.

Auf Grund der Komplexität und des Umfangs des gesamten Quellcodes der Pumpensteuerung ist diese vollständig und zusammenhängend im Anhang abgedruckt. Im Folgenden zeigen einige Screenshots die Funktionsweise und den Umfang des aktuellen Releasestands von *Pump Master 2000*. Nach dem Start bietet die Software alle mit dem System verbunden Spritzenpumpen zur Auswahl. Zusätzlich lässt sich ein Empfänger für Triggersignale[7] auswählen, sofern entsprechende Geräte am PC angeschlossen sind.

[5]Die Programmiersprache Python wurde ursprünglich in den Niederlanden entwickelt und ist eine interpretierte höhere Programmiersprache, deren Syntax auf Übersichtlichkeit optimiert und die leichte Verständlichkeit hin reduziert ist. Der Name geht auf die englische Komikergruppe Monty Python zurück.

[6]Zu den unixoiden oder unixähnlichen Betriebssystemen gehören alle Betriebssysteme, die die Funktionsweise von UNIX nachahmen und damit grundlegende Systemstrukturen und Befehle implementieren, wie sie mit UNIX seit 1969 eingeführt wurden. Neben Linux mit allen seinen verschiedenen Distributionen gehören auch alle BSD-Varianten und auch das aus der BSD-Linie hervorgegangene macOS von Apple Inc. dazu.

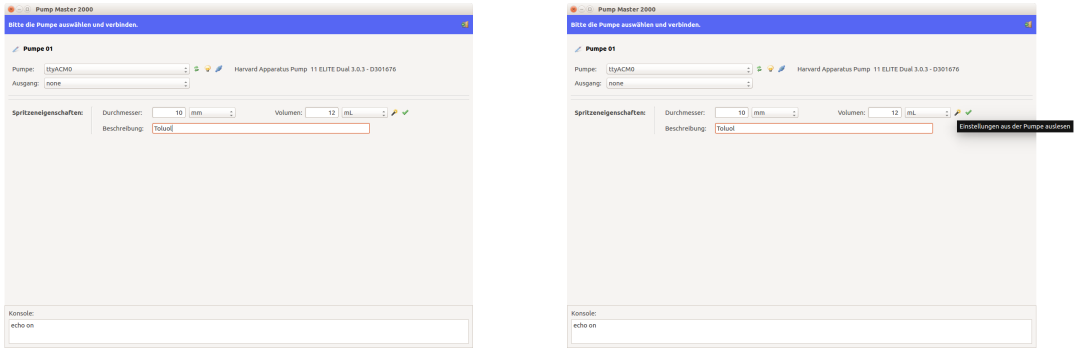
[7]Als Empfänger für Triggersignale dient beispielsweise ein Ardunio, wie in Kapitel 3.3, Seite 91, in Form des Fraktionierers vorgestellt: Bei nach Abschluss eines Pumpzyklus wird dadurch der Fraktionierer zum Wechsel des Auffangbehälters veranlasst.



(a) Spritzenpumpenauswahl (b) Verbindung mit der Pumpe

Abbildung 4.2: Nach dem Start: Auswahl der zu steuernden Spritzenpumpe und Herstellen der Verbindung mit dieser

Wurde die gewünschte Spritzenpumpe ausgewählt, stellt man die serielle Verbindung zu dieser her. Spritzenpumpen von Harvard Apparatus signalisieren die Verbindungsherstellung durch ein kurzes, von *Pump Master 2000* initiiertes, Blinken des Displays der Spritzenpumpe, welches auch durch Klick auf das Glühbirnensymbol erneut erfolgen kann.
Wenn die Verbindung mit der Spritzenpumpe hergestellt ist, müssen die Parameter der verwendeten Spritze der Software mitgeteilt werden. Dies geschieht entweder durch Eingabe der Parameter durch den Anwender oder durch Auslesen aus der Spritzenpumpe (Zauberstab-Symbol), sofern die Parameter bereits in der Spritzenpumpe von der vorherigen Nutzung hinterlegt sind. Zusätzlich lässt sich der Pumpe eine Beschreibung zuteilen, die bei der Ansteuerung mehrerer Spritzenpumpen hilft, die Übersicht zu wahren.

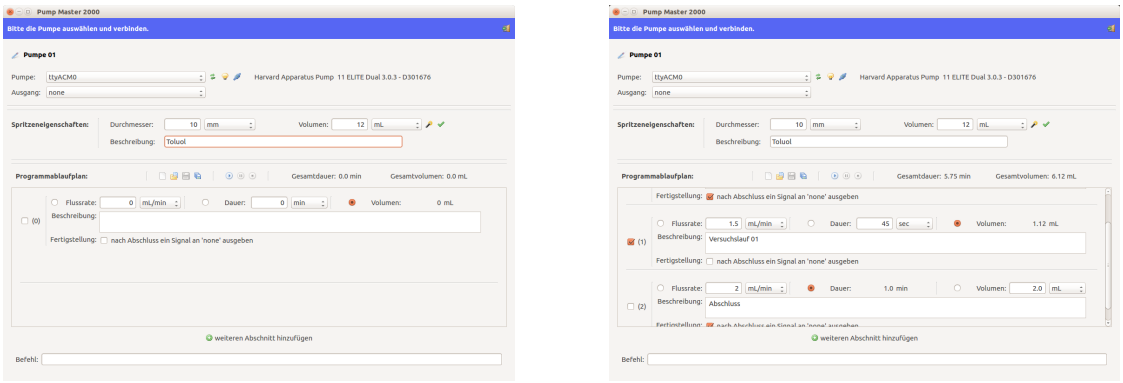


(a) Manuelle Parametereinstellung (b) Auslesen der Pumpenparameter

Abbildung 4.3: Einstellen der Spritzenparameter

Nach Übernahme der Spritzenparameter ist es möglich, Programmablaufpläne anzulegen. Des Weiteren können bereits angelegte und zuvor gespeicherte Programmablaufpläne an dieser Stelle geladen werden. Da die Programmablaufpläne in einem generischen Format vorliegen, also unabhängig von dem zum Einsatz kommenden Spritzenpumpentyp oder der verwendeten Spritze sind, können auf diesem Wege auch komplexe Programmablaufpläne

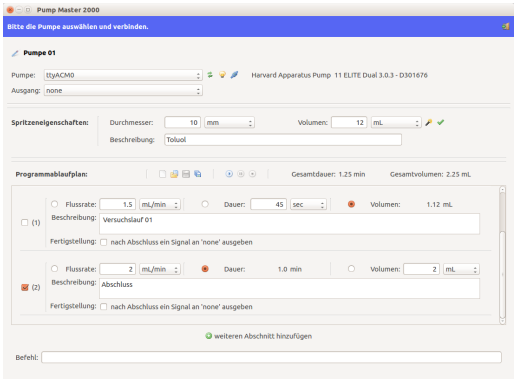
auf andere Pumpen-Spritzen-Konstellationen übertragen werden, ohne diese neu entwerfen zu müssen. Diese Kompatibilität wird jedoch durch die Güte der Spritzenpumpen begrenzt: nur wenige Spritzenpumpen sind dazu in der Lage, beispielsweise Flussraten von Nanolitern pro Minute bei Verwendung von 20 mL-Spritzen zu ermöglichen. Entsprechende Grenzen müssen dem Datenblatt der Spritzenpumpe entnommen und separat beachtet werden. Aufgrund der Vielfalt an Spritzenpumpentypen, verbunden mit der Dynamik des Marktes ist eine softwareseitige Prüfung auf eine solche Konsistenz des Programmablaufplans derzeit nicht möglich.



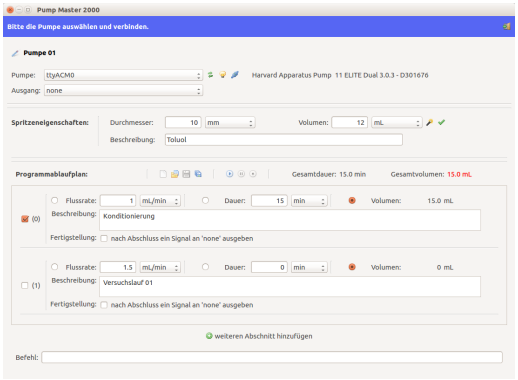
(a) Erster PAP-Abschnitt (b) Weitere PAP-Abschnitte

Abbildung 4.4: Erstellen eines Programmablaufplanes

Pump Master 2000 erlaubt das Hinzufügen beliebig vieler Abschnitte innerhalb des Programmablaufplans, welche alle einzeln aktiviert und auch deaktiviert werden können, damit auch umfangreichere Programmablaufpläne weiterhin flexibel eingesetzt werden können. Jeder Abschnitt erwartet die Eingabe von zwei aus drei möglichen Parametern: wahlweise können Flussrate und Eluationsdauer, Eluationsdauer und Eluationsvolumen oder Flussrate und Eluationsvolumen vorgegeben werden. Der entsprechend dritte Parameter wird dynamisch aus den beiden anderen Parametern berechnet. Zusätzlich wird die Eluationsdauer und das Eluationsvolumen aller aktivierten Abschnitte des Programmablaufplans aufaddiert und gezeigt. Sobald entweder das Eluationsvolumen eines Abschnittes oder das gesamte Eluationsvolumen das voreingestellte Gesamtvolumen der Spritze überschreitet, gibt die Software entsprechenden Hinweis aus und färbt die Volumenangaben rot ein. Neben den Eluationsparametern kann jeder Abschnitt optional mit einer Beschreibung versehen werden, um eine bessere Übersicht über den gesamten Programmablaufplan zu behalten. So kann im Beschreibungstext die Nummer es Reagenzglases im Karussell des Fraktionierers hinterlegt werden, um eine spätere schnelle Zuordnung der Einzelfraktionen zu ermöglichen. Darüber hinaus kann für jeden Abschnitt ein Trigger gesetzt werden. Wurde zuvor ein Empfänger für Triggersignale definiert, verschickt die Software nach Beendigung eines Abschnitts ein Triggersignal. So wird beispielsweise der Fraktionierer zum Wechsel der Vorlage aufgefordert.



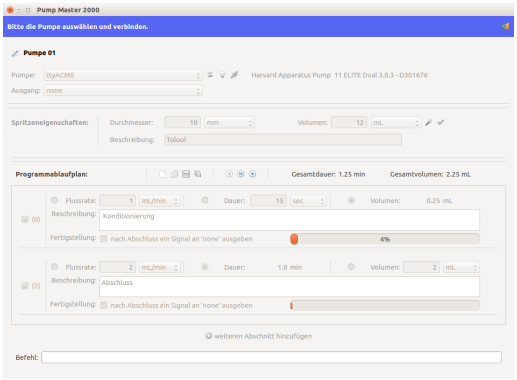
(a) Inaktiver PAP-Abschnitt



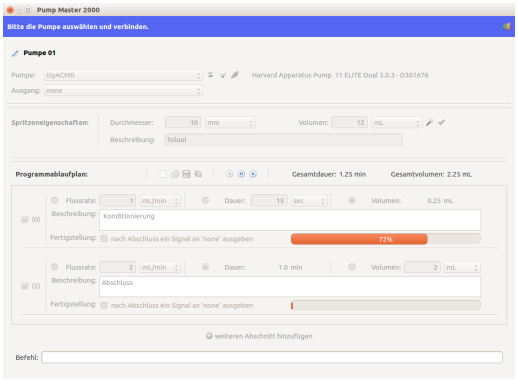
(b) Eluationsvolumen übersteigt Spritzenvolumen

Abbildung 4.5: Fehlerbehandlung im Programmablaufplanes

Wurden alle gewünschten Abschnitte des Programmablaufplans definiert, wird die serielle Abarbeitung aller aktiven Abschnitte durch Anklicken des Play-Symbols gestartet. *Pump Master 2000* arbeitet nun alle aktiven Abschnitte ab, wobei inaktive Abschnitte während der Ausführung ausgeblendet werden, um nur den wirklichen Programmablaufplan zu sehen.



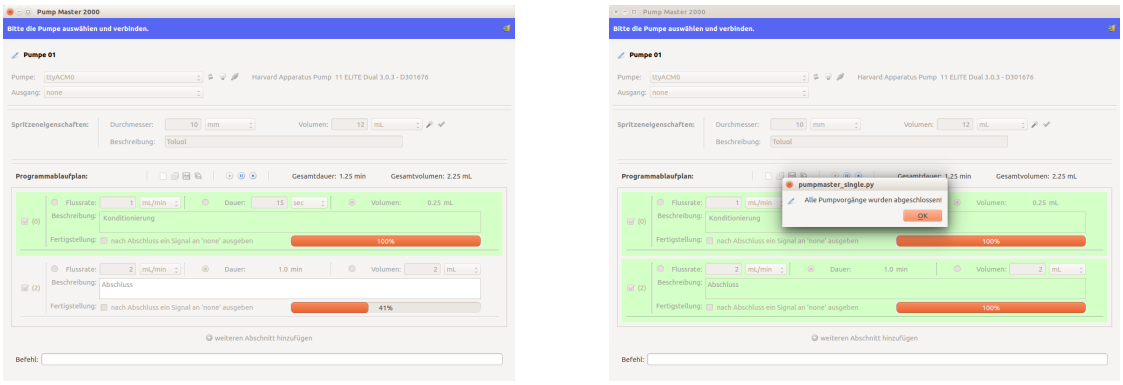
(a) Beginn



(b) Abschnitt 1

Abbildung 4.6: Durchlauf des Programmablaufplanes - Teil 1

In dem Abschnitt, welcher gerade ausgeführt wird, zeigt ein dynamisch wachsender Balken den Prozessfortschritt in Echtzeit an. Bereits erfolgreich abgearbeitete Abschnitte werden mit grünem Hintergrund versehen. Beides trägt dazu bei, dass man auch während des Versuchslaufs stets den Überblick über den aktuellen Fortschritt des Pumpvorgangs behält. Nach Abschluss des gesamten Programmablaufplans signalisiert ein Popup-Fenster die erfolgreiche Beendigung des Pumpvorgangs. Ein Klick auf *OK* führt wieder zurück zum vollständigen, editierbaren Programmablaufplan, wie er von vor dem Start bekannt ist.



(a) Abschnitt 2 (b) Ende

Abbildung 4.7: Durchlauf des Programmablaufplanes - Teil 2

Aufgund der Nutzung von Multiprozessorfähigkeiten ist eine simultane individuelle Steuerung mehrerer Spritzenpumpen aus einem Frontend heraus realisierbar und für ein späteres Softwarerelease angedacht. Durch die Verwendung von Python3 als Programmiersprache des Backends ist die Implementierung eines Webservers zum einfachen Aufruf eines Webfrontends mittels Webbrowser problemlos möglich und Teil kommender Softwarereleases. Zusätzlich geplant ist, sofern der verwendete Spritzenpumpentyp entsprechende Messung zulässt, eine Anzeige des aktuellen Füllstandes entsprechend des eingestellten Spritzenvolumens im Verhältnis zum bereits eluierten Volumen. Der häufig in anderer Software verwendete idealisierte Füllstand der Spritze unter der Annahme einer vollen Spritze zu Beginn entspricht einer Vermutung ohne Beweis und ist daher ohne Aussagekraft und nur von kosmetischer Natur.

Kapitel 5

Internet of Lab

Die in dieser Arbeit vorgestellten Entwicklungen sind allesamt für sich autarke Module, welche zur Zusammenstellung eines kontinuierlichen mikrofluidischen Systems je nach Bedarf verwendet werden können. Alle Module sind dazu in der Lage, mit einem Steuerungscomputer zu kommunizieren - manche teilen der Steuerungseinheit Messwerte mit, andere warten auf entsprechende Befehle, um die ablaufende Reaktion zu beeinflussen. Eine zentrale Steuerungssoftware, wie in Kapitel 4 ansatzweise vorgestellt, kann die einzelnen Module auslesen und zeit- oder eventgesteuert eine Anpassung von Reaktionsparametern vornehmen. Das System erlaubt nicht nur eine lokale Überwachung und Steuerung durch den Bediener, sondern darüber hinaus einen Zugriff über das Internet, welcher beispielsweise über eine App oder ein Webinterface vonstatten gehen kann. Da dieses mit dem Internet verbundene Netzwerk aus Sensoren und Aktoren, gesteuert durch eine zentrale Steuerungseinheit, allgemein unter dem Begriff *Internet of Things (IoT)* bekannt ist, bietet es sich an, hier von einem **Internet of Lab (IoL)** zu sprechen.

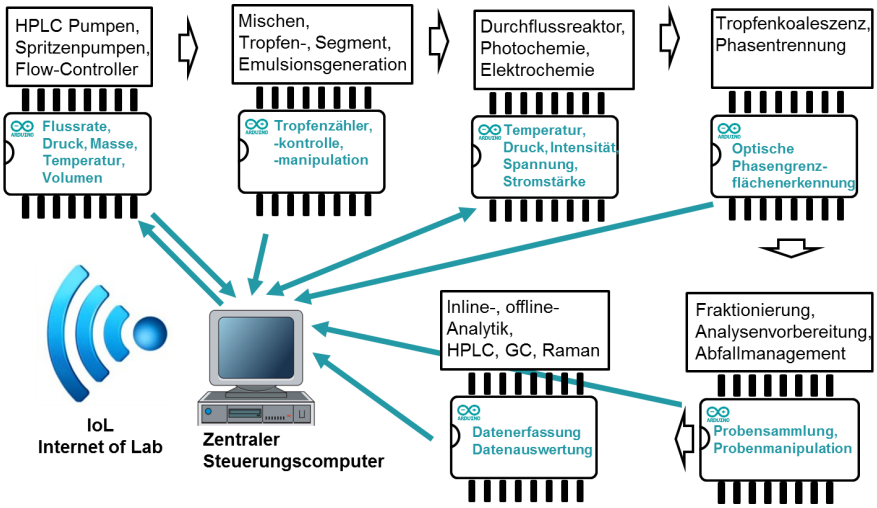


Abbildung 5.1: Schema der zentralen Steuerung des Internet of Lab mit den in dieser Arbeit vorgestellten unterschiedlichen Modulen in einem gemeinsamen System

Das Internet of Lab stellt ein neues, modulares Konzept des Anlagenbaus im Labormaßstab dar: alle Komponenten, die für die Umsetzung einer chemischen Reaktion im kontinuierlichen Fluss benötigt werden, werden wie gehabt als kontinuierliches System verbunden. Zusätzlich findet eine Verbindung über Datenaustauschleitungen, überwiegend via USB, statt. Zentraler Steuerungspunkt ist die Pumpensteuerungssoftware, die neben ihrer Grundfunktionalität, der Pumpensteuerung, auch Signale aus anderen Modulen aufnehmen und verarbeiten kann und zusätzlich weitere Einflussgrößen auf das System variieren kann. Im einfachsten Falle sorgt ein Triggersignal für das Auslösen einer Aktion, beispielsweise den Reagenzglaswechsel am Ende eines Pumpvorgangs.

Die folgende Abbildung zeigt das Anlagenkonzept eines kontinuierlichen zweiphasigen Systems, welches neben der Rückführung der abgetrennten kontinuierlichen Phase auch die optionale Rückgewinnung des Katalysators durch elektrochemische Regeneration erlaubt (9-11 in Abbildung 5.2). Denkbar ist die Oxidation eines Alkohols zu einem Aldehyd unter Verwendung von 2,2,6,6-Tetramethylpiperidinyloxyl (TEMPO) und der anschließenden elektrochemischen Regeneration des TEMPO.

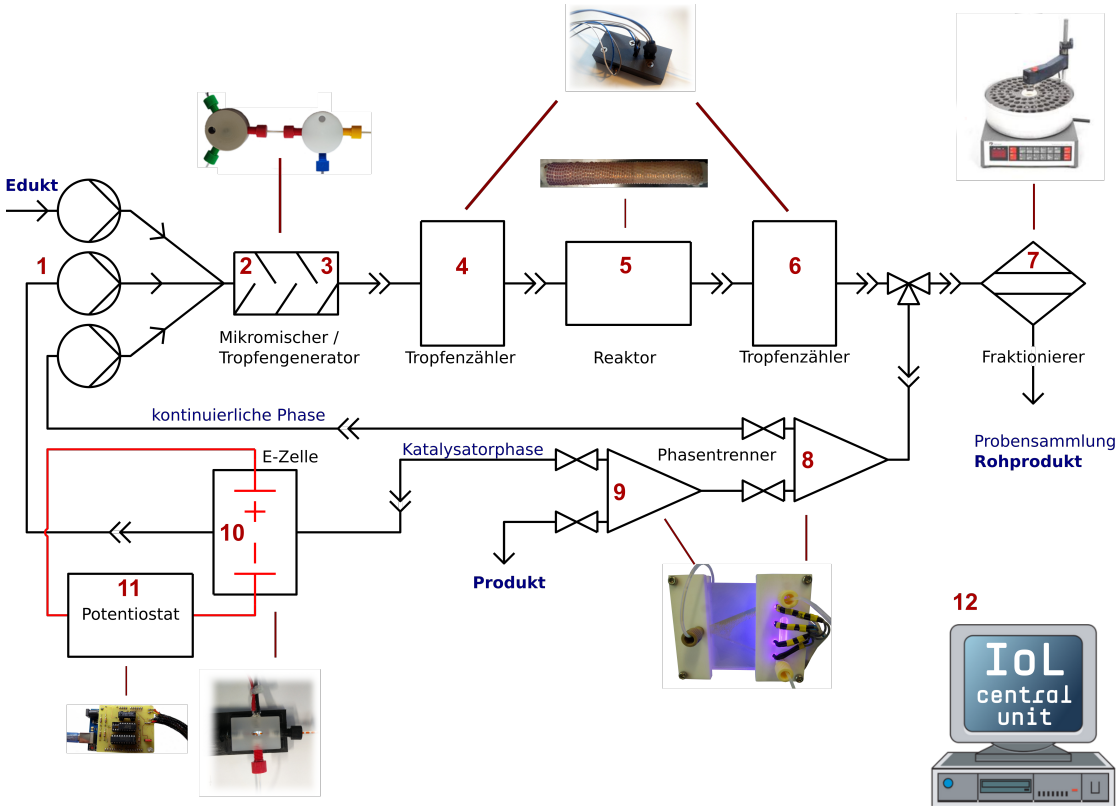


Abbildung 5.2: Konzept für ein kontinuierliches, tropfenbasiertes System mit einer elektrochemischen Katalysatorregeneration innerhalb des Katalysatorkreislaufs (siehe Text)

1 Pumpen: Sie versorgen das System mit Edukt/Agens, sorgen für die Zirkulation der kontinuierlichen Phase und des Katalysators und erlauben eine granulare Flussmengenvariation

2 Mikromischer: Edukt und Katalysator/Agens werden gemischt (homogene Lösung)

3 Tropfengenerator: Hier entstehen Tropfen oder Segmente der Edukt-Katalysator-Mischung innerhalb der kontinuierlichen Phase

4 Tropfenzähler:

- homogene Lösung: Bestimmung der Lichtabsorption
- heterogene Lösung: Prüfung der Qualität und Frequenz der Tropfen/Segemente

5 Reaktor: Er besteht im einfachsten Falle aus einem Verweilelement (gewickelte Stahl- oder PTFE-Kapillare oder gefräster Reaktor) und befindet sich innerhalb eines Thermostaten zur Temperatursteuerung.

6 Tropfenzähler:

- homogene Lösung: Vergleiche die Lichtabsorption mit **4** als Indikator für den Umsatz
- heterogene Lösung: Prüfung der Qualität und Frequenz der Tropfen/Segemente, Detektion von Koaleszenz

7 Probensammler: Sorgt für ein automatisches Sammeln von Proben des Rohprodukts gemäß der Festlegung im DoE

8 Phasentrenner: Trennung von Rohprodukt und kontinuierlicher Phase, Kreislaufführung der kontinuierlichen Phase

9 Phasentrenner (optional): Trennung von Rohprodukt und verbrauchtem/unverbrauchtem Katalysator/Agens

10 Elektrochemische Zelle (optional): Zur Regeneration von Katalysator/Agens

11 Potentiostat: Elektrochemische Umsetzung

12 Iol zentrale Steuerungseinheit mit Zugang zum Internet

Ziel des Systems ist ein in sich geschlossenes mikrofluidisches Laborsystem, welches die gängigsten im Labor veränderbaren Reaktionsparameter autonom rastern kann, also beispielsweise eine bestimmte Reaktion bei unterschiedlichen Temperaturen mit verschiedenen Flussraten durchführt, sodass bei Anbindung eines Inline-Analysengerätes (z.B. Inline-IR, Inline-GC, Inline-Raman) direkt die Umsatz- und Ausbeutenabhängigkeit der veränderten Reaktionsparameter sichtbar wird. So ist der Bediener dazu in der Lage, nach der Vorgabe der zu testenden Reaktionsparameterintervalle, das System autark die Messungen in der gewünschten Granularität durchführen zu lassen und sich am Ende das Ergebnis von unzähligen Einzelmessungen kombiniert und wahlweise auch grafisch aufbereitet ausgeben zu lassen.

Bereits im praktischen Einsatz ist eine Anlage zur Umsetzung von 4-(4-Methoxybenzyl)-triphenylphosphoniumchlorid mit Terephthalaldehyd im Mikroreaktor [51]. Das folgende Schaubild zeigt den Systemaufbau:

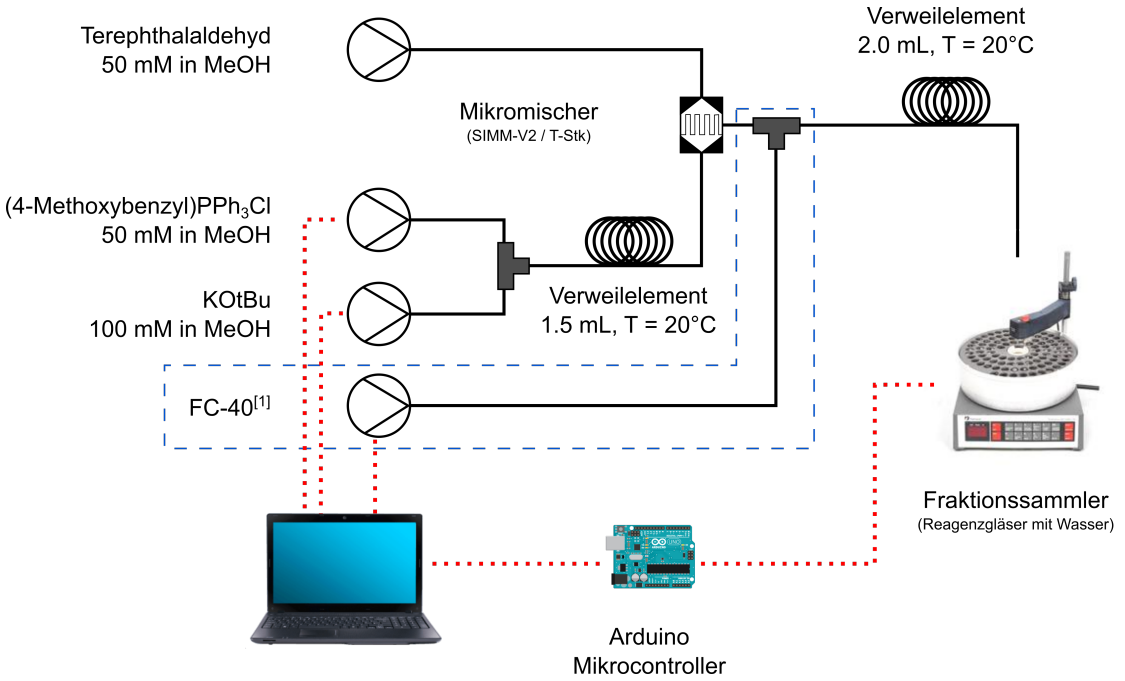


Abbildung 5.3: Analgenschema der automatisierten Anlage zur Durchführung und Probensammlung der Wittig-Reaktionen im kontinuierlichen Fluss unter Verwendung der in dieser Arbeit beschriebenen Pumpensteuerung kombiniert mit dem daraus getriggerten Fraktionssammler.
Quelle: Dissertation D. Karl [51]
[1]Nur für zweiphasige Versuche eingebaut.

In dieser Anlage konnte nach einmaliger Programmierung der zentralen Steuerungseinheit ohne weitere Interaktionen des Experimentators ein vollständiges Design of Experiment (DoE) für Flussraten[1] zwischen 0,51 mL min^{-1} und 2,67 mL min^{-1} abgefahren werden, mitsamt der Abtrennung des Reaktionsvorlaufs und aller Übergänge zwischen einer Flussratenänderung und der Gleichgewichtseinstellung der Reaktion. Die Messung erstreckte sich über 14 unterschiedliche Flussraten, welche für zwei verschiedene Verweilzeiten und dies einmal unter Verwendung eines SIMM-V2-Mischers und einmal unter Verwendung eines T-Stücks ermittelt wurden. Daraus ergeben sich fast 60 Einzelmessungen, denen jeweils ein Reinigungs- und Konditionierabschnitt voraus geht, sodass über 120 Reaktionsabschnitte durchgeführt werden mussten. Die gesamte Messreihe konnte mit Hilfe des Internet of Lab nicht nur an einem Arbeitstag durchgeführt werden, sondern erforderte nach dem Start des Systems keine weitere Interaktion des Experimentators. Dieser führte in der Zwischenzeit die Analytik bereits beendeter Messungen durch.

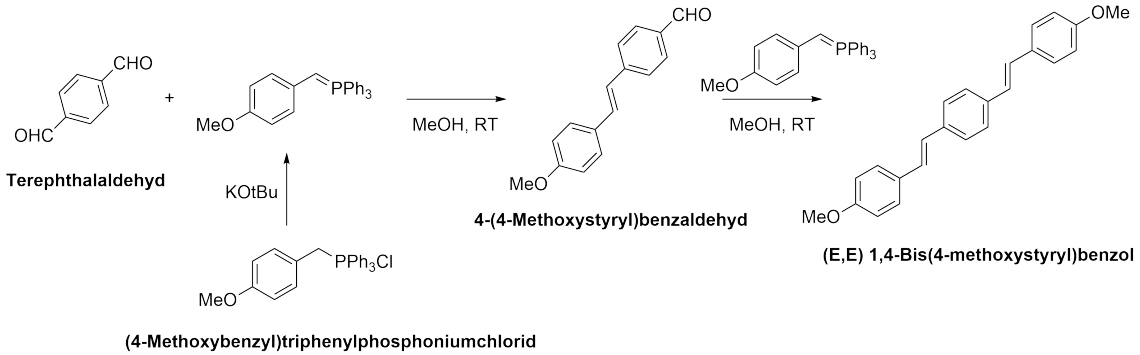


Abbildung 5.4: Reaktionsgleichung der Reaktion von Terephthalaldehyd mit (4-Methoxybenzyl)triphenylphosphoniumchlorid

Die Verbindung der Mikrofluidik mit intelligenter Computertechnologie erlaubt es, ein früher sehr zeit- und arbeitsintensives Rastern von Reaktionsparametern zur Ermittlung optimaler Reaktionsparameter automatisiert durch ein computergesteuertes System durchführen zu lassen und dadurch die menschliche Arbeitskapazität zu schonen und eine Beeinflussung durch menschliche Fehler zu vermeiden. Durch die hier vorgestellte Client-Server-Konzeption ist darüber hinaus eine externe Kontrolle bzw. Einflussnahme möglich, sodass sich das System auch für Versuche mit chemischen Reaktionen, die hochexplosiv, stark mutagen oder auf eine andere Art als stark lebensfeindlich gelten, geeignet sind.

[1] Die unterste ($0,51\,mL\,min^{-1}$) und oberste ($2,67\,mL\,min^{-1}$) Grenze der für die Reaktion möglichen Flussraten war bereits vorher bekannt, sodass eine Ermittlung durch die Anlage nicht benötigt wurde.

Kapitel 6

Ausblick

Das in dieser Arbeit vorgestellte Konzept des *Internet of Lab* mitsamt der dazu bereits entwickelten Komponenten zeigt nur den Anfang der Möglichkeiten, welche die intelligente Vernetzung von Sensor- und Aktor-Komponenten zu einem Gesamtsystem bietet. Die hier gezeigten Komponenten decken nur einen Teil derjenigen Komponenten eines mikrofluidischen Systems ab, die durch elektronische Bauteile zu smarten Komponenten (steuerbar bzw. auslesbar) werden können. So ist beispielsweise unter Zuhilfenahme von Magnetventilen die Entwicklung einer smarten Weiche denkbar, welche in Kombination mit dem hier vorgestellten Tropfenzähler eine bestimmte Anzahl Tropfen bzw. ein bestimmtes Reaktionsvolumen aus dem kontinuierlichen Fluss entnimmt, um dieses während des fortlaufenden Prozesses zu entnehmen. So können in einem laufenden System von definierten Zwischenschritten oder Intermediaten Proben zur Analyse entnommen werden, ohne den kontinuierlichen Fluss zu beeinträchtigen. Auch die Eingliederung weiterer bereits bestehender computerisierter Komponenten, allen voran Inline-Analysegeräte in das bestehende *Internet of Lab* stellt einen wichtigen Schritt für die Zukunft dar.

Neben der Erweiterung der Zahl der Komponenten und der Weiterentwicklung dieser liegt ein besonderes Augenmerk auf dem zentralen Steuerungscomputer, der alle Komponenten verbindet. Dieser verarbeitet derzeit die vorher vom Experimentator festgelegten Programmablaufpläne und steuert die einzelnen Komponenten, sofern diese nicht wie der Phasentrenner autonom sind. Einzig im Fehlerfall ändert die zentrale Steuerung semi-eigenständig den aktuellen Programmablaufplan und wechselt auf den Fehlerstatus, der, wie auch zuvor festgelegt, alle Komponenten stoppt. Zukünftig ist der Einbau einer erweiterten Intelligenz bzw. grundlegenden Eingenverantwortlichen Steuerung durch die zentrale Steuereinheit denkbar: Ein Parameterscreening über mehrere Reaktionsparameter stellt mathematisch ein n-dimensionales Problem dar, bei dem n für die Anzahl der veränderten Parameter steht. Möchte man beispielsweise das Ausbeutemaximum für eine Reaktion (Temperaturbereich von circa 4°C bis 90°C) bei unterschiedlichen Eduktzusammensetzungen und verschiedenen Verweilzeiten (Gesamtflussraten) ermitteln, so ergibt sich ein 3-dimensionales

Problem, bei welchem man zu jedem vorher festgelegten Eduktverhältnis für unterschiedliche Verweilzeiten und jeweils jede Temperatur im gewählten Bereich einen eigenen Versuch durchführen muss. Selbst unter Verwendung der hier vorgestellten tropfenbasierten kontinuierlichen Reaktionsverfahren stellt dies einen sehr langwierigen Prozess dar. Hier ist der Ansatzpunkt für intelligente Algorithmen, die, vergleichbar den Näherungsberechnungen von Potentialhyperflächen in der physikalischen Chemie, mit deutlich weniger Messpunkten, also durchgeführten Reaktionen, eine Vorhersage des gesuchten Ausbeutemaximums für eine vorgegebene Reaktion erlauben.

Einen weiteren vielversprechenden Ansatz liefert die Entwicklung von neuronalen Systemen bzw. ganzen neuronalen Netzen [52, 53], welche nach dem Erlernen für diese eine Reaktion wichtigen Grundparameter und durch das bereits vorhandene Wissen über optimale Parameter anderer Reaktionen Voraussagen zur Eingrenzung der zu prüfenden Parameterbereiche treffen, um die Anzahl der notwendigen Screeningreaktionen weiter zu verringern.

Gerade die genannten neuen Entwicklungen zur künstlichen Intelligenz können die Arbeit und den Alltag in chemischen Laboratorien einfacher und sicherer gestalten und lassen hoffen, dass diese durch den derzeit offenen Entwicklungsansatz keine Frage des Laborbudgets sein werden.

Um den freien Gedanken weiter zu geben, stehen auch alle in dieser Arbeit von mir entwickelten Hard- und Software-Komponenten unter der GNU General Public License v3.0 [54], welche die freie Verwendung meiner Arbeit zulässt und ausdrücklich wünscht. Alle Grafiken und Bilder, sofern urheberrechtlich nicht anderes gekennzeichnet, stammen von mir und unterliegen der Creative Commons Lizenz *CC BY-NC-SA* [55].

Anhang A

Quelltexte

A.1 Pyrometer

Die hier gezeigte Bibliothek (`mlx90621.h` und `mlx90621.cpp`) wird neben den Standardbibliotheken `SPI.h` und `TFT.h` zur Ansteuerung des IR-Array-Chips Melexis MLX90621 benötigt.

Listing A.1: Quelltext: Bibliothek mlx90621.h - Pyrometer

```
1  /* Quelle: Make 3/2017, Termika Fotilo, F. Pfeifer und F. Schäffer */
2  #ifndef __MLX90621__
3  #define __MLX90621__
4  #include <Wire.h>
5  #include <utility/twi.h>
6
7  #if BUFFER_LENGTH < 64
8    #error "BUFFER_LENGTH in Wire.h nicht auf 64 gesetzt."
9  #endif
10 #if TWI_BUFFER_LENGTH < 64
11   #error "TWI_BUFFER_LENGTH in twi.h nicht auf 64 gesetzt."
12 #endif
13
14 #include <Arduino.h>
15
16 class MLX90621 {
17   private:
18     uint8_t eeprom_dump_address = 0x50;
19     uint8_t chip_address = 0x60;
20     uint16_t configreg;
21     int16_t vcp;
22     int16_t acp;
23     int16_t acommon;
24     int16_t ksta;
25     int8_t bcpee;
26     float alphacp;
```

```
27      int8_t tgc;
28      uint8_t aiscale;
29      uint8_t biscale;
30      uint16_t alpha0;
31      uint8_t alpha0scale;
32      uint8_t deltaalphascale;
33      uint16_t epsilon;
34      uint8_t irpixels[128];
35      float ta;
36      int8_t ks4ee;
37      uint8_t ksscale;
38      uint8_t eepromMLX[256];      // Buffer
39      uint8_t read_eeprom_64 (uint8_t);
40      uint8_t read_eeprom (void);
41      void write_trim (uint8_t);
42      void write_config (uint8_t, uint8_t);
43      int32_t read_config (void);
44      uint8_t test_por (void);
45      int32_t read_ptat (void);
46      int16_t read_compensation (void);
47      uint8_t read_ir (void);
48
49    public:
50      MLX90621 (void);
51    uint8_t init (void);
52      void read_all_irfield (float [16][4]);
53      float get_ptat (void);
54  };
55  #endif
```

Listing A.2: Quellcodeausschnitt: Bibliothek mlx90621.h - Pyrometer

```
1   /* Quelle: Make 3/2017, Termika Fotilo, F. Pfeifer und F. Schäffer */
2   #include "mlx90621.h"
3
4   MLX90621::MLX90621(void) {
5     Wire.begin();
6   }
7
8   uint8_t MLX90621::init (void) {
9     delay(5);
10    if (!read_eeprom())
11      return 0;
12    write_trim (eepromMLX[0xF7]);
13    write_config (eepromMLX[0xF5], eepromMLX[0xF6]);
14    configreg = read_config();
15    ta = get_ptat();
16    if (ta > 300 || ta < -20)
17      return 0;
```

```
18    vcp = read_compensation();
19    acp = (int16_t)( eepromMLX[0xD4] << 8 ) | eepromMLX[0xD3];
20    bcpee = (int8_t)eepromMLX[0xD5];
21    tgc = (int8_t)eepromMLX[0xD8];
22    aiscale = eepromMLX[0xD9] >> 4;
23    biscale = eepromMLX[0xD9] & 0x0F;
24    alpha0 = ( eepromMLX[0xE1] << 8 ) | eepromMLX[0xE0];
25    alpha0scale = eepromMLX[0xE2];
26    deltaalphascale = eepromMLX[0xE3];
27    alphacp = (( eepromMLX[0xD7] << 8 ) | eepromMLX[0xD6]) / (float) (
        pow (2, alpha0scale) * pow (2, 3 - ( (configreg >> 4) & 0x03)));
28    epsilon = ( eepromMLX[0xE5] << 8 ) | eepromMLX[0xE4] ;
29    acommon = (int16_t)(( eepromMLX[0xD1] << 8 ) | eepromMLX[0xD0]);
30    ksta = (int16_t)(( eepromMLX[0xD7] << 8 ) | eepromMLX[0xD6]);
31    ks4ee = (int8_t)eepromMLX[0xC4];
32    ksscale = eepromMLX[0xC0];
33    return 1;
34  }
35
36  void MLX90621::read_all_irfield (float temperatures[16][4]) {
37    uint8_t x, y, i;
38    int16_t vir;
39    uint8_t deltaai;
40    int8_t bieeprom;
41    float bi;
42    float bcp;
43    uint8_t deltaalpha;
44    float viroffsetcompensated;
45    float virtgccompensated;
46    float vircpoffsetcompensated;
47    float vircompensated;
48    float ai;
49    float alphacomp;
50    float alpha;
51    float ks4;
52    uint64_t tak4;
53    float sx;
54    float to;
55    while (test_por ()) {
56      delay (10);
57      init ();
58    }
59    if (!read_ir ())
60      return;
61    for (i = 0; i < 64; i++) {
62      vir = (int16_t)(irpixels[i * 2 + 1] << 8 | irpixels[i * 2]);
63      deltaai = eepromMLX[i];
64      bi = (int8_t)eepromMLX[0x40 + i];
65      deltaalpha = eepromMLX[0x80 + i];
```

```
66      ai = (acommon + deltaai * pow (2, aiscale)) / pow (2, ( (
        configreg >> 4) & 0x03));
67      bi = bi / (pow (2, biscale) * pow (2, 3 - ( (configreg >> 4) & 0
        x03)));
68      viroffsetcompensated = vir - (ai + bi * (ta - 25.0));
69      bcp = bcpee / (pow (2, biscale) * pow (2, 3 - ( (configreg >> 4)
        & 0x03)));
70      vircpoffsetcompensated = vcp - (acp + bcp * (ta - 25.0) );
71      virtgccompensated = viroffsetcompensated - ( (tgc / 32.0) *
        vircpoffsetcompensated);
72      vircompensated = virtgccompensated / (epsilon / 32768);
73      alpha = ( ( alpha0 / pow (2, alpha0scale) ) + ( deltaalpha / pow
        (2, deltaalphascale) ) ) / (float) pow (2, 3 - ( (configreg >> 4)
        & 0x03)) ;
74      alphacomp = (1 + (ksta/pow (2, 20)) * (ta - 25.0)) * (alpha - tgc
        * alphacp);
75      ks4 = ks4ee / pow (2, ksscale + 8);
76      tak4 = pow ( (ta + 273.15), 4);
77      sx = ks4 * pow ( ( ( pow (alphacomp, 3) * vircompensated ) + (
        pow  ( alphacomp, 4 ) * tak4 ) ), (1 / 4.0) );
78      to = pow ( (vircompensated / ( alphacomp * (1 - ks4 * 273.15) +
        sx ) ) + tak4, (1 / 4.0) ) - 273.15;
79      temperatures[i%16][(uint8_t)i/16] = to;
80    }
81  }
82
83  uint8_t MLX90621::read_eeprom_64 (uint8_t start) {
84    uint16_t i;
85    Wire.beginTransmission(eeprom_dump_address);
86    Wire.write (start);
87    Wire.endTransmission(0);
88    if (!Wire.requestFrom(eeprom_dump_address, 64) )
89      return 0;
90    for (i = start; i < (start + 64); i++)
91      eepromMLX[i] = Wire.read();
92    return 1;
93  }
94
95  uint8_t MLX90621::read_eeprom (void) {
96    uint16_t i;
97    if (!read_eeprom_64 (0x00))
98      return 0;
99    if (!read_eeprom_64 (0x40))
100     return 0;
101   if (!read_eeprom_64 (0x80))
102     return 0;
103   if (!read_eeprom_64 (0xC0))
104     return 0;
105   return 1;
```

```
106 }
107
108 void MLX90621::write_trim (uint8_t trimvalue) {
109   Wire.beginTransmission(chip_address);
110   Wire.write (0x04);
111   Wire.write (trimvalue - 0xAA);
112   Wire.write (trimvalue);
113   Wire.write (0x00 - 0xAA);
114   Wire.write (0x00);
115   Wire.endTransmission();
116 }
117
118 void MLX90621::write_config (uint8_t lsb, uint8_t hsb) {
119   Wire.beginTransmission(chip_address);
120   Wire.write (0x03);
121   Wire.write (lsb - 0x55);
122   Wire.write (lsb);
123   Wire.write (hsb - 0x55);
124   Wire.write (hsb);
125   Wire.endTransmission();
126 }
127
128 int32_t MLX90621::read_config (void) {
129   uint16_t configreg;
130   Wire.beginTransmission(chip_address);
131   Wire.write (0x02);
132   Wire.write (0x92);
133   Wire.write (0x00);
134   Wire.write (0x01);
135   Wire.endTransmission(0);
136   if (!Wire.requestFrom(chip_address, 2))
137     return -1;
138   configreg = Wire.read();
139   configreg |= (Wire.read() << 8);
140   return configreg;
141 }
142
143 uint8_t MLX90621::test_por (void) {
144   return (read_config () & 0x400);
145 }
146
147 int32_t MLX90621::read_ptat (void) {
148   int32_t ptat = 0;
149   Wire.beginTransmission(chip_address);
150   Wire.write (0x02);
151   Wire.write (0x40);
152   Wire.write (0x00);
153   Wire.write (0x01);
154   Wire.endTransmission(0);
```

```
155   if (!Wire.requestFrom(chip_address, 2))
156     return -1;
157   ptat = Wire.read();
158   ptat |= Wire.read() << 8;
159   return ptat;
160 }
161
162 float MLX90621::get_ptat (void) {
163   uint8_t exp1, exp2;
164   float kt1, kt2, vth, tmp;
165   int32_t ptat;
166   do {
167     ptat = read_ptat();
168     if (ptat == -1) {
169       delay (1000);
170       init();
171     }
172   } while (ptat == -1);
173   exp2 = ( (configreg >> 4) & 0x03);
174   vth = ( eepromMLX[0xDB] << 8 ) | eepromMLX[0xDA];
175   if (vth > 32767)
176     vth -= 65536;
177   vth = vth / pow (2, 3-exp2);
178   kt1 = (( eepromMLX[0xDD] << 8 ) | eepromMLX[0xDC]);
179   if (kt1 > 32767)
180     kt1 -= 65536;
181   exp1 = ( eepromMLX[0xD2] >> 4);
182   kt1 = kt1 / ( pow (2, exp1) * pow (2, 3-exp2) );
183
184   kt2 = (( eepromMLX[0xDF] << 8 ) | eepromMLX[0xDE]);
185   if (kt2 > 32767)
186     kt2 -= 65536;
187   exp1 = (eepromMLX[0xD2] & 0x0F);
188   kt2 = kt2 / ( pow (2, exp1+10) * pow (2, 3-exp2) );
189   return (((kt1 * -1.0) + sqrt( kt1*kt1 - (4 * kt2) * (vth - ptat) ))
        / (2 * kt2) ) + 25.0;
190 }
191
192 int16_t MLX90621::read_compensation (void) {
193   int8_t cp = 0;
194   Wire.beginTransmission(chip_address);
195   Wire.write (0x02);
196   Wire.write (0x41);
197   Wire.write (0x00);
198   Wire.write (0x01);
199   Wire.endTransmission(0);
200   if (!Wire.requestFrom(chip_address, 2))
201     return -1;
202   cp = Wire.read();
```

```
203   return ( (int16_t) Wire.read() << 8 | cp);
204 }
205
206 uint8_t MLX90621::read_ir (void) {
207   uint8_t i;
208   Wire.beginTransmission(chip_address);
209   Wire.write (0x02);
210   Wire.write (0x00);
211   Wire.write (0x04);
212   Wire.write (0x10);
213   Wire.endTransmission(0);
214   if (!Wire.requestFrom(chip_address, 32))
215     return 0;
216   for (i=0; i < 32; i++)
217     irpixels[i] = Wire.read();
218   Wire.beginTransmission(chip_address);
219   Wire.write (0x02);
220   Wire.write (0x01);
221   Wire.write (0x04);
222   Wire.write (0x10);
223   Wire.endTransmission(0);
224   if (!Wire.requestFrom(chip_address, 32))
225     return 0;
226   for (i=0; i < 32; i++)
227     irpixels[i+32] = Wire.read();
228   Wire.beginTransmission(chip_address);
229   Wire.write (0x02);
230   Wire.write (0x02);
231   Wire.write (0x04);
232   Wire.write (0x10);
233   Wire.endTransmission(0);
234   if (!Wire.requestFrom(chip_address, 32))
235     return 0;
236   for (i=0; i < 32; i++)
237     irpixels[i+64] = Wire.read();
238   Wire.beginTransmission(chip_address);
239   Wire.write (0x02);
240   Wire.write (0x03);
241   Wire.write (0x04);
242   Wire.write (0x10);
243   Wire.endTransmission(0);
244   if (!Wire.requestFrom(chip_address, 32))
245     return 0;
246   for (i=0; i < 32; i++)
247     irpixels[i+96] = Wire.read();
248   return 1;
249 }
```

A.2 Pumpensteuerung

Der im Folgenden gezeigte Python3-Quelltext benötigt zusätzlich zu einer Python3-Umgebung auf einem unixoiden Betriebssystem die Installation der Python3-Module `numpy`, `serial`, `pyside` und `pyside-qtgui` inklusive deren Abhängigkeiten. Der Quelltext ist in drei Teile unterteilt, die eigentlich ausführbare Datei `pumpmaster_single.py`, dem Quelltextbereich des Backends (`backend.py`) und dem des QT-Frontends (`frontend.py`).

Listing A.3: Quelltext: pumpmaster2000.py - Pumpensteuerung

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import os
import sys
import time
import serial
import platform
import numpy as np
from multiprocessing import Process
from functools import partial
from PySide.QtGui import *
from PySide.QtCore import *
from PySide.QtCore import Signal as pyqtSignal
from PySide.QtCore import Slot as pyqtSlot
from backend import Monitor
from frontend import PumpMaster
if __name__ == "__main__":
 app = QApplication([])
 QCoreApplication.setOrganizationName("JULRIC")
 QCoreApplication.setOrganizationDomain("julric.de")
 QCoreApplication.setApplicationName("PumpMaster2000")
 thread = Monitor()
 thread.sendEvent.this.connect(PumpMaster.receiveEvent)
 thread.start()
 window = PumpMaster()
 app.exec_()
```

Listing A.4: Quelltext: backend.py - Pumpensteuerung

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import os
import sys
import time
import serial
import platform
import numpy as np
```

```
10 from multiprocessing import Process
11 from frontend import PumpMaster
12 from functools import partial
13 from PySide.QtGui import *
14 from PySide.QtCore import *
15 global device
16 global interrupt_device
17 global papCounter
18 global status
19 global start_pump
20 global pause_pump
21 global stop_pump
22 global finished_pump
23 global end_threads
24 global can_exit
25 global running_number
26 global running_status
27 global output
28 device = False
29 interrupt_device = False
30 papCounter = 1
31 status = False
32 start_pump = False
33 pause_pump = False
34 stop_pump = False
35 finished_pump = False
36 end_threads = False
37 can_exit = False
38 running_number = 0
39 running_status = 0
40 output = False
41 pap = [[0 for x in range(12)] for x in range(20)]
42 unit = [["mL/min","mL/sec","mL/hour","uL/sec","uL/min","uL/hour","nL/
       sec","nL/min","nL/hour","pL/sec","pL/min","pL/hour"],
43   ["min","sec","hour",0,0,0,0,0,0,0,0,0],
44   ["mL","uL","nL","pL",0,0,0,0,0,0,0,0]]
45 Loewe01 = ["11 ELITE Dual 3.0.3 - Loewe01","20.05",0,"20",0]
46 for number in range(20):#set comment field in all pap
47  pap[number][11] = ' '
48 import frontend
49
50 class connection(QObject):
51  this = Signal(str)
52 class Monitor(QThread):
53  sendEvent = connection()
54  def run(self):
55   # monitor values of pump while running
56   time.sleep(5) #start after initialisation of gui
57   global device
```

```
58    global start_pump
59    global pause_pump
60    global stop_pump
61    global finished_pump
62    global end_threads
63    global can_exit
64    global status
65    global pap
66    global output
67    global running_number
68    while end_threads == False:
69     if start_pump == True and status == 0:#start new pump run
70      if device != False and device != 'Loewe01' and os.path.exists(
      device) == True:
71       ser = serial.Serial(port=device,baudrate=115200,parity=serial.
      PARITY_NONE,stopbits=serial.STOPBITS_TWO,bytesize=serial.
      EIGHTBITS)
72       ser.isOpen()
73       if pap[running_number][0] == 1 and pap[running_number][2] == 0
      and pap[running_number][5] != 0:#pap active and not already
      processed
74        self.sendEvent.this.emit("15_"+str(running_number).zfill(2))
75        # clear pap parameter tvolume
76        ser.write(('ctvolume\r\n').encode('UTF-8'))
77        output = ''
78        out = ''
79        # wait one second and give device time to answer
80        time.sleep(0.1)
81        while ser.inWaiting() > 0:
82         out += ser.read(1).decode('UTF-8')
83        if out != '':
84         output = out
85        # clear pap parameter ivolume
86        ser.write(('civolume\r\n').encode('UTF-8'))
87        out = ''
88        # wait one second and give device time to answer
89        time.sleep(0.1)
90        while ser.inWaiting() > 0:
91         out += ser.read(1).decode('UTF-8')
92        if out != '':
93         output = out + '\n' + output
94        # clear pap parameter time
95        ser.write(('ctime\r\n').encode('UTF-8'))
96        out = ''
97        # wait one second and give device time to answer
98        time.sleep(0.1)
99        while ser.inWaiting() > 0:
100        out += ser.read(1).decode('UTF-8')
101       if out != '':
```

```
102             output = out + '\n' + output
103           # set pap parameter flowrate
104           ser.write(('irate '+str(pap[running_number][5])+' '+str(unit
        [0][int(pap[running_number][6])])+'\r\n').encode('UTF-8'))
105           out = ''
106           # wait one second and give device time to answer
107           time.sleep(0.1)
108           while ser.inWaiting() > 0:
109             out += ser.read(1).decode('UTF-8')
110           if out != '':
111             output = out + '\n' + output
112           # set pap parameter volume
113           ser.write(('tvolume '+str(pap[running_number][9])+' '+str(unit
        [2][int(pap[running_number][10])])+'\r\n').encode('UTF-8'))
114           out = ''
115           # wait one second and give device time to answer
116           time.sleep(0.1)
117           while ser.inWaiting() > 0:
118             out += ser.read(1).decode('UTF-8')
119           if out != '':
120             output = out + '\n' + output
121           # start run
122           ser.write(('run\r\n').encode('UTF-8'))
123           out = ''
124           # wait one second and give device time to answer
125           time.sleep(0.5)
126           while ser.inWaiting() > 0:
127             out += ser.read(1).decode('UTF-8')
128           if out != '':
129             output = out + '\n' + output
130             self.sendEvent.this.emit("10")
131           status = 1
132         elif pap[running_number][5] == 0:#pap active but with zero
        flowrate
133           if papCounter > running_number:
134             running_number = running_number+1
135           else:
136             start_pump = False
137             pause_pump = False
138             stop_pump = False
139             finished_pump = True
140         elif pap[running_number][0] == 1 and pap[running_number][2] ==
        1:#pap active and processed
141           if papCounter > running_number:
142             running_number = running_number+1
143           else:
144             start_pump = False
145             pause_pump = False
146             stop_pump = False
```

```
147             finished_pump = True
148           elif pap[running_number][0] != 1:#pap not active
149            if papCounter > running_number:
150             running_number = running_number+1
151            else:
152             start_pump = False
153             pause_pump = False
154             stop_pump = False
155             finished_pump = True
156           ser.close()
157           self.sendEvent.this.emit("12")
158          else:
159           start_pump = False
160           pause_pump = False
161           stop_pump = True
162           device = False
163           frontend.refresh_clicked()
164         if start_pump == True and status == 1:#do monitoring while running
165          pump_status_full = self.get_pump_status()
166          if len(pump_status_full) > 1:
167           pump_status = pump_status_full[1].split()
168           if len(pump_status) > 3:
169            if pump_status[3][:1] == "I" or pump_status[3][:1] == "i":
170             flowrate = float(pump_status[0])/1000000000000*60
171             volume = float(pump_status[2])/1000000000000
172             if pump_status[3][-4:-3] == "S":
173              self.sendEvent.this.emit("17")
174              status = 0
175              start_pump = False
176              pause_pump = False
177              stop_pump = True
178             elif pump_status[3][-1:] == ".":
179              volume_unit=1000 if pap[running_number][10]==1 else 1000000
     if pap[running_number][10]==2 else 1000000000 if pap[
     running_number][10]==3 else 1
180              running_status = int(100*(round(volume, 2)/pap[running_number
     ][9]*volume_unit))
181              self.sendEvent.this.emit("11_"+str(running_number).zfill(2)+"
     _"+str(running_status))
182             elif pump_status[3][-1:] == "T":
183              running_status = 100
184              self.sendEvent.this.emit("11_"+str(running_number).zfill(2)+"
     _"+str(running_status))
185              pap[running_number][2] = 1
186              status = 0
187            else:
188             status = 0
189         if start_pump == True and status == 2:#start paused pump run
190          if device != False and device != 'Loewe01' and os.path.exists(
```

```
    device) == True:
191   ser = serial.Serial(port=device,baudrate=115200,parity=serial.
    PARITY_NONE,stopbits=serial.STOPBITS_TWO,bytesize=serial.
    EIGHTBITS)
192   ser.isOpen()
193   # start run
194   ser.write(('run\r\n').encode('UTF-8'))
195   output = ''
196   out = ''
197   # wait one second and give device time to answer
198   time.sleep(0.1)
199   while ser.inWaiting() > 0:
200    out += ser.read(1).decode('UTF-8')
201   if out != '':
202    output = out
203    self.sendEvent.this.emit("10")
204   status = 1
205   ser.close()
206   self.sendEvent.this.emit("12")
207  else:
208   start_pump = False
209   pause_pump = False
210   stop_pump = True
211   device = False
212   frontend.refresh_clicked()
213 if pause_pump == True and status == 1:#pause running pump
214  if device != False and device != 'Loewe01' and os.path.exists(
    device) == True:
215   ser = serial.Serial(port=device,baudrate=115200,parity=serial.
    PARITY_NONE,stopbits=serial.STOPBITS_TWO,bytesize=serial.
    EIGHTBITS)
216   ser.isOpen()
217   # stop run
218   ser.write(('stp\r\n').encode('UTF-8'))
219   output = ''
220   out = ''
221   # wait one second and give device time to answer
222   time.sleep(0.1)
223   while ser.inWaiting() > 0:
224    out += ser.read(1).decode('UTF-8')
225   if out != '':
226    output = out
227    self.sendEvent.this.emit("10")
228   status = 2
229   ser.close()
230   self.sendEvent.this.emit("13")
231  else:
232   start_pump = False
233   pause_pump = False
```

```
234         stop_pump = True
235         device = False
236         frontend.refresh_clicked()
237       if stop_pump == True:#stop running pump
238        if device != False and device != 'Loewe01' and os.path.exists(
          device) == True:
239         ser = serial.Serial(port=device,baudrate=115200,parity=serial.
          PARITY_NONE,stopbits=serial.STOPBITS_TWO,bytesize=serial.
          EIGHTBITS)
240         ser.isOpen()
241         # stop run
242         ser.write(('stp\r\n').encode('UTF-8'))
243         output = ''
244         out = ''
245         # wait one second and give device time to answer
246         time.sleep(0.1)
247         while ser.inWaiting() > 0:
248          out += ser.read(1).decode('UTF-8')
249         if out != '':
250          output = out
251         # clear pap parameter tvolume
252         ser.write(('ctvolume\r\n').encode('UTF-8'))
253         out = ''
254         # wait one second and give device time to answer
255         time.sleep(0.1)
256         while ser.inWaiting() > 0:
257          out += ser.read(1).decode('UTF-8')
258         if out != '':
259          output = out + '\n' + output
260         # clear pap parameter ivolume
261         ser.write(('civolume\r\n').encode('UTF-8'))
262         out = ''
263         # wait one second and give device time to answer
264         time.sleep(0.1)
265         while ser.inWaiting() > 0:
266          out += ser.read(1).decode('UTF-8')
267         if out != '':
268          output = out + '\n' + output
269         # clear pap parameter time
270         ser.write(('ctime\r\n').encode('UTF-8'))
271         out = ''
272         # wait one second and give device time to answer
273         time.sleep(0.1)
274         while ser.inWaiting() > 0:
275          out += ser.read(1).decode('UTF-8')
276         if out != '':
277          output = out + '\n' + output
278          self.sendEvent.this.emit("10")
279         ser.close()
```

```
280         status = 0
281         self.sendEvent.this.emit("14")
282         for number in range(papCounter):#reset all pap not to be
            finished
283           pap[number][2] = 0
284       else:
285         start_pump = False
286         pause_pump = False
287         stop_pump = True
288         device = False
289         frontend.refresh_clicked()
290       stop_pump = False
291     if finished_pump == True:#all jobs finished on pump
292       self.sendEvent.this.emit("16")
293       start_pump = False
294       pause_pump = False
295       stop_pump = True
296       finished_pump = False
297   can_exit = True
298  def get_pump_status(self):
299   # receive/read version and serial from syringe-pump, set pump-
      address to zero if necessary
300   global device
301   if device != False and device != 'Loewe01' and os.path.exists(
      device) == True:
302     ser_status = serial.Serial(port=device,baudrate=115200,parity=
      serial.PARITY_NONE,stopbits=serial.STOPBITS_TWO,bytesize=serial.
      EIGHTBITS)
303     ser_status.isOpen()
304     ser_status.write('status\r\n'.encode('UTF-8'))
305     status = ''
306     time.sleep(2)
307     while ser_status.inWaiting() > 0:
308       status += ser_status.read(1).decode('UTF-8')
309     pump_status_full = status.splitlines()
310     ser_status.close()
311   else:
312     status = "status\n0 0 0 ......\n:"
313     pump_status_full = status.splitlines()
314   return pump_status_full
```

Listing A.5: Quelltext: frontend.py - Pumpensteuerung

```
1 #!/usr/bin/python3
2 # -*- coding: utf-8 -*-
3
4 import os
5 import sys
6 import time
```

```
7  import serial
8  import platform
9  import numpy as np
10 from multiprocessing import Process
11 from functools import partial
12 from PySide.QtGui import *
13 from PySide.QtCore import *
14 from PySide.QtCore import Signal as pyqtSignal
15 from PySide.QtCore import Slot as pyqtSlot
16 global fname_pap
17 pap_layout = False
18 pap_area = False
19 main_area = False
20 fname_pap = False
21 import backend
22 class PumpMaster(QWidget, object):
23  device_label = None
24  device_edit = None
25  serial_label = None
26  pump_label = None
27  pump_version = None
28  settings_area = None
29  main_area = None
30  def __init__(self):
31   super(PumpMaster, self).__init__()
32   self.setWindowIcon(QIcon(os.path.join('icons', 'syringe.png')))
33   global layout
34   layout = QBoxLayout(QBoxLayout.TopToBottom)
35   layout.setContentsMargins(0, 0, 0, 0)
36   self.setting_dirty = True
37   global somesignal
38   somesignal = Signal()
39   # Header bar
40   header_bar = QFrame()
41   header_bar.setStyleSheet("QWidget { background: rgb(86, 102, 243);
       } " +
42         "QLabel {color: white; font-weight: bold; font-size:4em;}" +
43         "QToolTip { color: rgb(255, 255, 255); background-color: rgb
     (20, 20, 20); " +
44         "border: 1px solid white; }")
45   header_bar.setAutoFillBackground(True)
46   header_bar.setFixedHeight(45)
47   header_bar_layout = QBoxLayout(QBoxLayout.LeftToRight)
48   header_bar.setLayout(header_bar_layout)
49   layout.addWidget(header_bar)
50   self.create_header_bar(header_bar_layout)
51   header_bar_layout.addStretch()
52   self.create_exit_button(header_bar_layout)
53   # Pump 01 area
```

```
54      global pump_01_area
55      pump_01_area = QFrame()
56      pump_01_area.setStyleSheet("QToolTip { color: rgb(255, 255, 255);
          background-color: rgb(20, 20, 20); padding: 2px; }")
57      pump_01_area_layout = QBoxLayout(QBoxLayout.TopToBottom)
58      pump_01_area.setLayout(pump_01_area_layout)
59      layout.addWidget(pump_01_area)
60      self.create_pump_01_area(pump_01_area_layout)
61      # Pump 01 Header
62      pump_01_header = QFrame()
63      pump_01_header_layout = QBoxLayout(QBoxLayout.LeftToRight)
64      pump_01_header.setLayout(pump_01_header_layout)
65      pump_01_area_layout.addWidget(pump_01_header)
66      self.create_pump_01_header(pump_01_header_layout)
67      # Serial Connection area
68      global serial_connection_area
69      serial_connection_area = QFrame()
70      serial_connection_area.setObjectName("serial_connection_area")
71      serial_connection_area_layout = QGridLayout()
72      serial_connection_area_layout.setColumnMinimumWidth(0, 60)
73      serial_connection_area_layout.setColumnMinimumWidth(1, 20)
74      serial_connection_area_layout.setColumnMinimumWidth(2, 20)
75      serial_connection_area_layout.setColumnMinimumWidth(3, 20)
76      serial_connection_area_layout.setColumnMinimumWidth(4, 20)
77      serial_connection_area_layout.setColumnMinimumWidth(5, 20)
78      serial_connection_area_layout.setColumnMinimumWidth(6, 20)
79      serial_connection_area_layout.setColumnMinimumWidth(7, 370)
80      serial_connection_area.setLayout(serial_connection_area_layout)
81      pump_01_area_layout.addWidget(serial_connection_area)
82      self.create_serial_connection_area(serial_connection_area_layout)
83      # horizontal line
84      hline_01 = QFrame()
85      hline_01.setFrameShape(QFrame.HLine)
86      hline_01.setFrameShadow(QFrame.Sunken)
87      pump_01_area_layout.addWidget(hline_01)
88      # horizontal line
89      hline_02 = QFrame()
90      hline_02.setFrameShape(QFrame.HLine)
91      hline_02.setFrameShadow(QFrame.Sunken)
92      pump_01_area_layout.addWidget(hline_02)
93      # Settings area
94      global settings_area
95      settings_area = QFrame()
96      settings_area.setObjectName("settings_area")
97      settings_area_layout = QGridLayout()
98      settings_area_layout.setColumnMinimumWidth(0, 180)
99      settings_area_layout.setColumnMinimumWidth(1, 20)
100     settings_area_layout.setColumnMinimumWidth(4, 40)
101     settings_area_layout.setColumnMinimumWidth(5, 150)
```

```
102     settings_area_layout.setColumnMinimumWidth(7, 40)
103     settings_area_layout.setColumnMinimumWidth(8, 10)
104     settings_area_layout.setColumnMinimumWidth(9, 150)
105     settings_area_layout.setRowMinimumHeight(0, 10)
106     settings_area_layout.setRowMinimumHeight(1, 10)
107     settings_area_layout.setRowStretch(0, 0)
108     settings_area_layout.setRowStretch(1, 0)
109     settings_area_layout.setRowStretch(2, 3) # invisible 3rd row can
          stretch
110     settings_area_layout.setVerticalSpacing(10)
111     settings_area.setLayout(settings_area_layout)
112     pump_01_area_layout.addWidget(settings_area)
113     settings_area.setVisible(False)
114     self.create_settings_area(settings_area_layout)
115     # Widget area
116     global main_area
117     main_area = QFrame()
118     main_layout = QBoxLayout(QBoxLayout.TopToBottom)
119     main_area.setLayout(main_layout)
120     pump_01_area_layout.addWidget(main_area)
121     main_area.setVisible(False)
122     self.create_main_area(main_layout)
123     # horizontal line
124     hline_03 = QFrame()
125     hline_03.setFrameShape(QFrame.HLine)
126     hline_03.setFrameShadow(QFrame.Sunken)
127     main_layout.addWidget(hline_03)
128     # horizontal line
129     hline_04 = QFrame()
130     hline_04.setFrameShape(QFrame.HLine)
131     hline_04.setFrameShadow(QFrame.Sunken)
132     main_layout.addWidget(hline_04)
133     # Main area header
134     global main_area_header
135     main_area_header = QFrame()
136     main_header_layout = QGridLayout()
137     main_header_layout.setColumnMinimumWidth(0, 100)
138     main_header_layout.setColumnMinimumWidth(1, 20)
139     main_header_layout.setColumnMinimumWidth(5, 36)
140     main_header_layout.setColumnMinimumWidth(6, 20)
141     main_header_layout.setColumnMinimumWidth(9, 36)
142     main_header_layout.setColumnMinimumWidth(10, 20)
143     main_header_layout.setColumnMinimumWidth(11, 120)
144     main_header_layout.setColumnMinimumWidth(12, 100)
145     main_header_layout.setColumnMinimumWidth(13, 120)
146     main_header_layout.setColumnMinimumWidth(14, 100)
147     main_header_layout.setColumnMinimumWidth(15, 20)
148     main_area_header.setLayout(main_header_layout)
149     main_layout.addWidget(main_area_header)
```

```
150     main_area_header.setVisible(True)
151     self.create_main_area_header(main_header_layout)
152     # Progam flow chart area
153     global pap_layout
154     global pap_area
155     pap_area = QFrame()
156     pap_layout = QBoxLayout(QBoxLayout.TopToBottom)
157     pap_area.setLayout(pap_layout)
158     main_layout.addWidget(pap_area)
159     self.create_pap_area(pap_layout)
160     # Build PAP-Items
161     self.build_item_area('0')
162     # PAP scrolling area
163     global pap_sc_area
164     pap_sc_area = QScrollArea()
165     pap_sc_layout = QBoxLayout(QBoxLayout.TopToBottom)
166     pap_sc_area.setLayout(pap_sc_layout)
167     pap_sc_area.setMinimumHeight(300)
168     pap_sc_area.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
169     pap_sc_area.setVerticalScrollBarPolicy(Qt.ScrollBarAsNeeded)
170     pap_sc_area.setWidget(pap_area)
171     pap_sc_area.setWidgetResizable(True)
172     main_layout.addWidget(pap_sc_area)
173     self.create_pap_sc_area(pap_sc_layout)
174     # Progam flow chart area add-button
175     pap_add = QFrame()
176     pap_add_layout = QBoxLayout(QBoxLayout.TopToBottom)
177     pap_add.setLayout(pap_add_layout)
178     main_layout.addWidget(pap_add)
179     self.create_pap_add(pap_add_layout)
180     # Input area
181     input_area = QFrame()
182     input_layout = QBoxLayout(QBoxLayout.LeftToRight)
183     input_area.setLayout(input_layout)
184     main_layout.addWidget(input_area)
185     self.create_input_area(input_layout)
186     layout.addStretch()
187     # Output area
188     global output_area
189     output_area = QScrollArea()
190     output_layout = QBoxLayout(QBoxLayout.TopToBottom)
191     output_area.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
192     output_area.setVerticalScrollBarPolicy(Qt.ScrollBarAsNeeded)
193     output_area.setLayout(output_layout)
194     output_area.setVisible(True)
195     output_area.setMinimumHeight(100)
196     output_area.setMaximumHeight(100)
197     layout.addWidget(output_area)
198     self.create_output_area(output_layout)
```

```
199     # Window layout
200     main_layout.addStretch()
201     pump_01_header_layout.addStretch()
202     self.setLayout(layout)
203     settings = QSettings()
204     size = settings.value("MainWindow/size")
205     if not size:
206       size = QSize(1100, 780)
207     self.resize(size)
208     position = settings.value("MainWindow/pos")
209     if not position:
210       position = QPoint(100, 100)
211     self.move(position)
212     self.setWindowTitle("Pump Master 2000")
213     self.show()
214   def build_item_area(self, number):
215     # Progam flow chart item
216     global item_layout
217     global item_area
218     item_area = QFrame()
219     item_layout = QGridLayout()
220     item_area.setLayout(item_layout)
221     item_area.setObjectName("item_"+number+"_area")
222     item_layout.setColumnMinimumWidth(2, 20)
223     item_layout.setColumnMinimumWidth(3, 50)
224     item_layout.setColumnMinimumWidth(4, 80)
225     item_layout.setColumnMinimumWidth(5, 90)
226     item_layout.setColumnMinimumWidth(6, 30)
227     item_layout.setColumnMinimumWidth(7, 20)
228     item_layout.setColumnMinimumWidth(8, 50)
229     item_layout.setColumnMinimumWidth(9, 80)
230     item_layout.setColumnMinimumWidth(10, 80)
231     item_layout.setColumnMinimumWidth(11, 30)
232     item_layout.setColumnMinimumWidth(12, 20)
233     item_layout.setColumnMinimumWidth(13, 50)
234     item_layout.setColumnMinimumWidth(14, 80)
235     item_layout.setColumnMinimumWidth(15, 80)
236     item_layout.setRowStretch(0, 0)
237     item_layout.setRowStretch(1, 0)
238     item_layout.setRowStretch(2, 0)
239     item_layout.setRowStretch(3, 0)
240     item_layout.setRowStretch(4, 3) # invisible 5th row can stretch
241     pap_layout.addWidget(item_area)
242     self.create_item_area(item_layout, number)
243   # noinspection PyUnresolvedReferences
244   def create_header_bar(self, layout):
245     # Header Line
246     self.headerlabel = QLabel()
247     self.headerlabel.setText("Bitte die Pumpe auswählen und verbinden."
```

```
    )
248   layout.addWidget(self.headerlabel)
249  # noinspection PyUnresolvedReferences
250  def create_exit_button(self, layout):
251   # Quit-Button
252   self.quit_button = QToolButton()
253   self.quit_button.setIconSize(QSize(16, 16))
254   self.quit_button.setIcon(QIcon(os.path.join("icons", "door_in.png")
      ))
255   self.quit_button.setStyleSheet("border: 0px;")
256   self.quit_button.setToolTip("Beenden")
257   self.quit_button.clicked.connect(self.close)
258   self.quit_button.setVisible(True)
259   layout.addWidget(self.quit_button)
260  # noinspection PyUnresolvedReferences
261  def create_pump_01_area(self, layout):
262   global fname_pap
263  # noinspection PyUnresolvedReferences
264  def create_pump_01_header(self, layout):
265   self.pump_01_icon = QLabel()
266   pixmap = QPixmap('icons/syringe_small.png')
267   self.pump_01_icon.setPixmap(pixmap)
268   layout.addWidget(self.pump_01_icon)
269   self.pump_01_label = QLabel()
270   self.pump_01_label.setStyleSheet("QLabel { color:rgb(0, 0, 0); font
      -weight: bold; }")
271   self.pump_01_label.setText("Pumpe 01")
272   layout.addWidget(self.pump_01_label)
273  # noinspection PyUnresolvedReferences
274  def create_serial_connection_area(self, layout):
275   # List of Serial Ports
276   device_label = QLabel("Pumpe:")
277   self.device_edit = QComboBox()
278   self.device_edit.setEditable(False)
279   self.device_edit.textChanged.connect(self.pump_device_changed)
280   self.device_edit.currentIndexChanged.connect(self.
      pump_device_changed)
281   self.device_edit.setMaximumHeight(28)
282   self.device_edit.setMinimumContentsLength(35)
283   filenames = list(filter(lambda x: x.startswith('ttyACM'), os.
      listdir('/dev/')))
284   if len(filenames) == 0:
285    self.device_edit.addItem("keine Pumpe gefunden, bitte
      aktualisieren")
286    backend.device = False
287   else:
288    for name in filenames:
289       self.device_edit.addItem(name)
290    backend.device = '/dev/' + filenames[0]
```

```
291     device_label.setBuddy(self.device_edit)
292     layout.addWidget(device_label, 0, 0)
293     layout.addWidget(self.device_edit, 0, 1)
294     # Refresh-Button
295     self.refresh_button = QToolButton()
296     self.refresh_button.setIconSize(QSize(16, 16))
297     self.refresh_button.setIcon(QIcon(os.path.join("icons", "
          arrow_refresh_small.png")))
298     self.refresh_button.setStyleSheet("border: 0px;")
299     self.refresh_button.setToolTip("Verfügbare Pumpen aktualisieren")
300     self.refresh_button.clicked.connect(self.refresh_clicked)
301     self.refresh_button.setVisible(True)
302     layout.addWidget(self.refresh_button, 0, 2)
303     # Identify-Button
304     self.identify_button = QToolButton()
305     self.identify_button.setIconSize(QSize(16, 16))
306     self.identify_button.setIcon(QIcon(os.path.join("icons", "lightbulb
          .png")))
307     self.identify_button.setStyleSheet("border: 0px;")
308     self.identify_button.setToolTip("Ausgewähle Pumpe sichtbar machen")
309     if backend.device == False:
310      self.identify_button.setVisible(False)
311     else:
312      self.identify_button.setVisible(True)
313     self.identify_button.clicked.connect(self.identify_pump)
314     layout.addWidget(self.identify_button, 0, 3)
315     # Connect-Button
316     self.connect_button = QToolButton()
317     self.connect_button.setObjectName("connect_button")
318     self.connect_button.setIconSize(QSize(16, 16))
319     self.connect_button.setIcon(QIcon(os.path.join("icons", "disconnect
          .png")))
320     self.connect_button.setStyleSheet("border: 0px;")
321     self.connect_button.setToolTip("Mit der gewählten Pumpe verbinden")
322     if backend.device == False:
323      self.connect_button.setVisible(False)
324     else:
325      self.connect_button.setVisible(True)
326     self.connect_button.clicked.connect(self.connect_pump)
327     layout.addWidget(self.connect_button, 0, 4)
328     # Disconnect-Button
329     self.disconnect_button = QToolButton()
330     self.disconnect_button.setIconSize(QSize(16, 16))
331     self.disconnect_button.setIcon(QIcon(os.path.join("icons", "connect
          .png")))
332     self.disconnect_button.setStyleSheet("border: 0px;")
333     self.disconnect_button.setToolTip("Verbindung zur Pumpe Trennen")
334     self.disconnect_button.clicked.connect(self.disconnect_pump)
335     self.disconnect_button.setVisible(False)
```

```
336     layout.addWidget(self.disconnect_button, 0, 4)
337     self.pump_label = QLabel("Harvard Apparatus Pump")
338     self.pump_label.setVisible(False)
339     layout.addWidget(self.pump_label, 0, 6)
340     self.pump_version = QLabel("11 ELITE")
341     self.pump_version.setVisible(False)
342     layout.addWidget(self.pump_version, 0, 7)
343     self.get_pump_version()
344 
345     # List of Serial Ports
346     interrupt_device_label = QLabel("Ausgang:")
347     self.interrupt_device_edit = QComboBox()
348     self.interrupt_device_edit.setEditable(False)
349     self.interrupt_device_edit.textChanged.connect(self.
          interrupt_device_changed)
350     self.interrupt_device_edit.currentIndexChanged.connect(self.
          interrupt_device_changed)
351     self.interrupt_device_edit.setMaximumHeight(28)
352     self.interrupt_device_edit.setMinimumContentsLength(35)
353     self.interrupt_device_edit.addItem("none")
354     interrupts = list(filter(lambda x: x.startswith('tty'), os.listdir(
          '/dev/')))
355     if len(interrupts) == 0:
356       self.interrupt_device_edit.addItem("none")
357       backend.interrupt_device = False
358     else:
359       for name in interrupts:
360         if backend.device != name:
361           self.interrupt_device_edit.addItem(name)
362       backend.interrupt_device = 'none'
363     interrupt_device_label.setBuddy(self.interrupt_device_edit)
364     layout.addWidget(interrupt_device_label, 1, 0)
365     layout.addWidget(self.interrupt_device_edit, 1, 1)
366   # noinspection PyUnresolvedReferences
367   def create_settings_area(self, layout):
368     self.settingsarealabel = QLabel()
369     self.settingsarealabel.setStyleSheet("QLabel { font-weight: bold; }
          ")
370     self.settingsarealabel.setText("Spritzeneigenschaften:")
371     self.settingsarealabel.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
372     layout.addWidget(self.settingsarealabel, 0, 0)
373     #
374     self.settings_vline_01 = QFrame()
375     self.settings_vline_01.setFrameShape(QFrame.VLine)
376     self.settings_vline_01.setFrameShadow(QFrame.Sunken)
377     layout.addWidget(self.settings_vline_01, 0, 1, 2, 1)
378     #
379     self.settings_diameter_label = QLabel()
380     self.settings_diameter_label.setText("Durchmesser:")
```

```
381     layout.addWidget(self.settings_diameter_label, 0, 2)
382     self.settings_diameter_edit = QLineEdit()
383     self.settings_diameter_edit.setMaximumHeight(28)
384     self.settings_diameter_edit.setFixedWidth(80)
385     self.settings_diameter_edit.setVisible(True)
386     self.settings_diameter_edit.setPlaceholderText("0")
387     self.settings_diameter_edit.setAlignment(Qt.AlignRight | Qt.
          AlignVCenter)
388     self.settings_diameter_label.setBuddy(self.settings_diameter_edit)
389     layout.addWidget(self.settings_diameter_edit, 0, 3)
390     self.settings_diameter_box = QComboBox()
391     self.settings_diameter_box.setEditable(False)
392     self.settings_diameter_box.setMaximumHeight(28)
393     self.settings_diameter_box.setMinimumContentsLength(5)
394     self.settings_diameter_box.addItem("mm")
395     layout.addWidget(self.settings_diameter_box, 0, 4)
396     self.settings_volume_label = QLabel()
397     self.settings_volume_label.setText("Volumen:")
398     self.settings_volume_label.setAlignment(Qt.AlignRight | Qt.
          AlignVCenter)
399     layout.addWidget(self.settings_volume_label, 0, 5)
400     self.settings_volume_edit = QLineEdit()
401     self.settings_volume_edit.setObjectName("settings_volume_edit")
402     self.settings_volume_edit.setMaximumHeight(28)
403     self.settings_volume_edit.setFixedWidth(80)
404     self.settings_volume_edit.setVisible(True)
405     self.settings_volume_edit.setPlaceholderText("0")
406     self.settings_volume_edit.setAlignment(Qt.AlignRight | Qt.
          AlignVCenter)
407     self.settings_volume_label.setBuddy(self.settings_volume_edit)
408     layout.addWidget(self.settings_volume_edit, 0, 6)
409     self.settings_volume_box = QComboBox()
410     self.settings_volume_box.setObjectName("settings_volume_unit")
411     self.settings_volume_box.setEditable(False)
412     self.settings_volume_box.setMaximumHeight(28)
413     self.settings_volume_box.setMinimumContentsLength(5)
414     self.settings_volume_box.addItem("mL")
415     self.settings_volume_box.addItem("uL")
416     self.settings_volume_box.addItem("nL")
417     self.settings_volume_box.addItem("pL")
418     self.settings_volume_box.addItem("fL")
419     layout.addWidget(self.settings_volume_box, 0, 7)
420     # Readsettings-Button
421     self.read_settings_button = QToolButton()
422     self.read_settings_button.setIconSize(QSize(16, 16))
423     self.read_settings_button.setIcon(QIcon(os.path.join("icons", "wand
          .png")))
424     self.read_settings_button.setStyleSheet("border: 0px;")
425     self.read_settings_button.setToolTip("Einstellungen aus der Pumpe
```

```
      auslesen")
426   self.read_settings_button.clicked.connect(self.get_pump_settings)
427   layout.addWidget(self.read_settings_button, 0, 8)
428   # Acceptsettings-Button
429   self.accept_settings_button = QToolButton()
430   self.accept_settings_button.setIconSize(QSize(16, 16))
431   self.accept_settings_button.setIcon(QIcon(os.path.join("icons", "
        tick.png")))
432   self.accept_settings_button.setStyleSheet("border: 0px;")
433   self.accept_settings_button.setToolTip("Eingegebene Werte ü
        bernehmen")
434   self.accept_settings_button.clicked.connect(self.set_pump_settings)
435   layout.addWidget(self.accept_settings_button, 0, 9)
436   # Comment-Button
437   self.settings_comment_label = QLabel()
438   self.settings_comment_label.setText("Beschreibung:")
439   self.settings_comment_label.setToolTip("Freifeld zur Beschreibung
        der aktuellen Pumpe, z.B. \"CP\"")
440   layout.addWidget(self.settings_comment_label, 1, 2)
441   self.settings_comment_edit = QLineEdit("")
442   self.settings_comment_edit.setObjectName("settings_comment")
443   self.settings_comment_edit.setToolTip("Freifeld zur Beschreibung
        der aktuellen Pumpe, z.B. \"CP\"")
444   self.settings_comment_edit.setPlaceholderText("Beschreibung der
        aktuellen Pumpe, z.B. \"CP\"")
445   self.settings_comment_edit.setMaximumHeight(28)
446   self.settings_comment_edit.setFixedWidth(513)
447   self.settings_comment_edit.setVisible(True)
448   self.settings_comment_label.setBuddy(self.settings_comment_edit)
449   layout.addWidget(self.settings_comment_edit, 1, 3, 1, 5)
450 # noinspection PyUnresolvedReferences
451 def create_main_area(self, layout):
452   global fname_pap
453 # noinspection PyUnresolvedReferences
454 def create_main_area_header(self, layout):
455   self.main_label = QLabel()
456   self.main_label.setStyleSheet("QLabel { font-weight: bold; }")
457   self.main_label.setText("Programmablaufplan:")
458   layout.addWidget(self.main_label, 0, 0)
459   #
460   self.main_vline_01 = QFrame()
461   self.main_vline_01.setFrameShape(QFrame.VLine)
462   self.main_vline_01.setFrameShadow(QFrame.Sunken)
463   layout.addWidget(self.main_vline_01, 0, 1)
464   #
465   # NewPAP-Button
466   self.new_button = QToolButton()
467   self.new_button.setIconSize(QSize(16, 16))
468   self.new_button.setIcon(QIcon(os.path.join("icons", "page_white.png
```

```
    "))))
469     self.new_button.setStyleSheet("border: 0px;")
470     self.new_button.setObjectName("new_button")
471     self.new_button.setToolTip("Programmablaufplan leeren")
472     self.new_button.setVisible(True)
473     self.new_button.clicked.connect(self.new_pap)
474     layout.addWidget(self.new_button, 0, 2)
475     # OpenPAP-Button
476     self.open_button = QToolButton()
477     self.open_button.setIconSize(QSize(16, 16))
478     self.open_button.setIcon(QIcon(os.path.join("icons", "folder_page.
        png")))
479     self.open_button.setStyleSheet("border: 0px;")
480     self.open_button.setObjectName("open_button")
481     self.open_button.setToolTip("Programmablaufplan öffnen")
482     self.open_button.setVisible(True)
483     self.open_button.clicked.connect(self.open_pap)
484     layout.addWidget(self.open_button, 0, 3)
485     # SavePAP-Button
486     self.save_button = QToolButton()
487     self.save_button.setIconSize(QSize(16, 16))
488     self.save_button.setIcon(QIcon(os.path.join("icons", "disk.png")))
489     self.save_button.setStyleSheet("border: 0px;")
490     self.save_button.setObjectName("save_button")
491     self.save_button.setToolTip("Programmablaufplan speichern")
492     self.save_button.setVisible(True)
493     self.save_button.setEnabled(False)
494     self.save_button.clicked.connect(self.save_pap)
495     layout.addWidget(self.save_button, 0, 4)
496     # SaveToPAP-Button
497     self.saveto_button = QToolButton()
498     self.saveto_button.setIconSize(QSize(16, 16))
499     self.saveto_button.setIcon(QIcon(os.path.join("icons", "
        disk_multiple.png")))
500     self.saveto_button.setStyleSheet("border: 0px;")
501     self.saveto_button.setObjectName("saveto_button")
502     self.saveto_button.setToolTip("Programmablaufplan speichern unter")
503     self.saveto_button.setVisible(True)
504     self.saveto_button.clicked.connect(self.saveto_pap)
505     layout.addWidget(self.saveto_button, 0, 5)
506     #
507     self.main_vline_02 = QFrame()
508     self.main_vline_02.setFrameShape(QFrame.VLine)
509     self.main_vline_02.setFrameShadow(QFrame.Sunken)
510     layout.addWidget(self.main_vline_02, 0, 6)
511     #
512     # StartPAP-Button
513     self.start_button = QToolButton()
514     self.start_button.setIconSize(QSize(16, 16))
```

```
515     self.start_button.setIcon(QIcon(os.path.join("icons", "
          control_play_blue.png")))
516     self.start_button.setStyleSheet("border: 0px;")
517     self.start_button.setObjectName("start_button")
518     self.start_button.setToolTip("Starten")
519     self.start_button.setVisible(True)
520     self.start_button.clicked.connect(self.pap_start)
521     layout.addWidget(self.start_button, 0, 7)
522     # PausePAP-Button
523     self.pause_button = QToolButton()
524     self.pause_button.setIconSize(QSize(16, 16))
525     self.pause_button.setIcon(QIcon(os.path.join("icons", "
          control_pause_blue.png")))
526     self.pause_button.setStyleSheet("border: 0px;")
527     self.pause_button.setObjectName("pause_button")
528     self.pause_button.setToolTip("Pausieren")
529     self.pause_button.setVisible(True)
530     self.pause_button.setEnabled(False)
531     self.pause_button.clicked.connect(self.pap_pause)
532     layout.addWidget(self.pause_button, 0, 8)
533     # StopPAP-Button
534     self.stop_button = QToolButton()
535     self.stop_button.setIconSize(QSize(16, 16))
536     self.stop_button.setIcon(QIcon(os.path.join("icons", "
          control_stop_blue.png")))
537     self.stop_button.setStyleSheet("border: 0px;")
538     self.stop_button.setObjectName("stop_button")
539     self.stop_button.setToolTip("Stoppen")
540     self.stop_button.setVisible(True)
541     self.stop_button.setEnabled(False)
542     self.stop_button.clicked.connect(self.pap_stop)
543     layout.addWidget(self.stop_button, 0, 9)
544     #
545     self.main_vline_03 = QFrame()
546     self.main_vline_03.setFrameShape(QFrame.VLine)
547     self.main_vline_03.setFrameShadow(QFrame.Sunken)
548     layout.addWidget(self.main_vline_03, 0, 10)
549     #
550     self.total_duration_name = QLabel("Gesamtdauer:")
551     self.total_duration_name.setMaximumHeight(28)
552     self.total_duration_name.setFixedWidth(120)
553     self.total_duration_name.setVisible(True)
554     self.total_duration_name.setAlignment(Qt.AlignRight | Qt.
          AlignVCenter)
555     layout.addWidget(self.total_duration_name, 0, 11)
556     self.total_duration_view = QLabel()
557     self.total_duration_view.setObjectName("total_duration_view")
558     self.total_duration_view.setMaximumHeight(28)
559     self.total_duration_view.setFixedWidth(100)
```

```
560     self.total_duration_view.setVisible(True)
561     self.total_duration_view.setAlignment(Qt.AlignLeft | Qt.
          AlignVCenter)
562     self.total_duration_view.setText("0.0 min")
563     layout.addWidget(self.total_duration_view, 0, 12)
564     self.total_volume_name = QLabel("Gesamtvolumen:")
565     self.total_volume_name.setMaximumHeight(28)
566     self.total_volume_name.setFixedWidth(120)
567     self.total_volume_name.setVisible(True)
568     self.total_volume_name.setAlignment(Qt.AlignRight | Qt.AlignVCenter
          )
569     layout.addWidget(self.total_volume_name, 0, 13)
570     self.total_volume_view = QLabel()
571     self.total_volume_view.setObjectName("total_volume_view")
572     self.total_volume_view.setToolTip("Addiertes Volumen aller
          Abschnitte - Volumen in <font color=red>roter Schrift</font>
          bedeutet eine Überschreitung des eingestellten Spritzenvolumens!"
          )
573     self.total_volume_view.setMaximumHeight(28)
574     self.total_volume_view.setFixedWidth(100)
575     self.total_volume_view.setVisible(True)
576     self.total_volume_view.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
577     self.total_volume_view.setText("0.0 mL")
578     layout.addWidget(self.total_volume_view, 0, 14)
579 # noinspection PyUnresolvedReferences
580 def create_pap_area(self, layout):
581     global fname_pap
582 # noinspection PyUnresolvedReferences
583 def create_pap_sc_area(self, layout):
584     global fname_pap
585 # noinspection PyUnresolvedReferences
586 def create_pap_add(self, layout):
587     self.pap_addbutton = QPushButton()
588     self.pap_addbutton.setObjectName("pap_addbutton")
589     self.pap_addbutton.setStyleSheet("QPushButton#addButton {
          background-color:red; border-style:outset; }")
590     self.pap_addbutton.setText("weiteren Abschnitt hinzufügen")
591     self.pap_addbutton.setIconSize(QSize(16, 16))
592     self.pap_addbutton.setIcon(QIcon(os.path.join("icons", "add.png")))
593     self.pap_addbutton.setStyleSheet("border: 0px;")
594     self.pap_addbutton.setToolTip("weiteren Abschnitt hinzufügen (
          maximal 20 Abschnitte)")
595     self.pap_addbutton.clicked.connect(self.add_pap)
596     self.pap_addbutton.setVisible(True)
597     layout.addWidget(self.pap_addbutton)
598 # noinspection PyUnresolvedReferences
599 def create_item_area(self, layout, number):
600     self.itemarea_label = QCheckBox("("+number+")")
601     self.itemarea_label.setObjectName("pap_"+number+"_active")
```

```
602     if str(backend.pap[int(number)][0]) != "0":
603      self.itemarea_label.setCheckState(Qt.Checked)
604     else:
605      self.itemarea_label.setCheckState(Qt.Unchecked)
606     layout.addWidget(self.itemarea_label, 0, 0, 3, 1)
607     #
608     self.itemarea_vline_01 = QFrame()
609     self.itemarea_vline_01.setFrameShape(QFrame.VLine)
610     self.itemarea_vline_01.setFrameShadow(QFrame.Sunken)
611     layout.addWidget(self.itemarea_vline_01, 0, 1, 3, 1)
612     #
613     self.itemarea_flowrate_button = QRadioButton(self)
614     self.itemarea_flowrate_button.setObjectName("pap_"+number+"
          _flowrate_ro")
615     self.itemarea_flowrate_button.setToolTip("Flussrate automatisch
          berechnen")
616     layout.addWidget(self.itemarea_flowrate_button, 0, 2)
617     self.itemarea_flowrate_label = QLabel()
618     self.itemarea_flowrate_label.setText("Flussrate:")
619     layout.addWidget(self.itemarea_flowrate_label, 0, 3)
620     self.itemarea_flowrate_edit = QLineEdit()
621     self.itemarea_flowrate_edit.setObjectName("pap_"+number+"
          _flowrate_value_edit")
622     self.itemarea_flowrate_edit.setMaximumHeight(28)
623     self.itemarea_flowrate_edit.setFixedWidth(80)
624     self.itemarea_flowrate_edit.setVisible(True)
625     self.itemarea_flowrate_edit.setAlignment(Qt.AlignRight | Qt.
          AlignVCenter)
626     self.itemarea_flowrate_edit.setText(str(backend.pap[int(number)
          ][5]))
627     self.itemarea_flowrate_label.setBuddy(self.itemarea_flowrate_edit)
628     layout.addWidget(self.itemarea_flowrate_edit, 0, 4)
629     self.itemarea_flowrate_view = QLabel()
630     self.itemarea_flowrate_view.setObjectName("pap_"+number+"
          _flowrate_value_view")
631     self.itemarea_flowrate_view.setMaximumHeight(28)
632     self.itemarea_flowrate_view.setFixedWidth(80)
633     self.itemarea_flowrate_view.setVisible(False)
634     self.itemarea_flowrate_view.setAlignment(Qt.AlignRight | Qt.
          AlignVCenter)
635     self.itemarea_flowrate_view.setText(str(backend.pap[int(number)
          ][5]))
636     layout.addWidget(self.itemarea_flowrate_view, 0, 4)
637     self.itemarea_flowrate_box = QComboBox()
638     self.itemarea_flowrate_box.setObjectName("pap_"+number+"
          _flowrate_unit_edit")
639     self.itemarea_flowrate_box.setEditable(False)
640     self.itemarea_flowrate_box.setMaximumHeight(28)
641     self.itemarea_flowrate_box.setMinimumContentsLength(6)
```

```
642     self.itemarea_flowrate_box.setCurrentIndex(int(backend.pap[int(
          number)][6]))
643     layout.addWidget(self.itemarea_flowrate_box, 0, 5)
644     self.itemarea_flowrate_view_unit = QLabel()
645     self.itemarea_flowrate_view_unit.setObjectName("pap_"+number+"
          _flowrate_unit_view")
646     self.itemarea_flowrate_view_unit.setMaximumHeight(28)
647     self.itemarea_flowrate_view_unit.setVisible(False)
648     self.itemarea_flowrate_view_unit.setText(backend.unit[0][int(
          backend.pap[int(number)][6])])
649     layout.addWidget(self.itemarea_flowrate_view_unit, 0, 5)
650     #
651     self.itemarea_vline_02 = QFrame()
652     self.itemarea_vline_02.setFrameShape(QFrame.VLine)
653     self.itemarea_vline_02.setFrameShadow(QFrame.Sunken)
654     layout.addWidget(self.itemarea_vline_02, 0, 6)
655     #
656     self.itemarea_duration_button = QRadioButton(self)
657     self.itemarea_duration_button.setObjectName("pap_"+number+"
          _duration_ro")
658     self.itemarea_duration_button.setToolTip("Programmdauer automatisch
          berechnen")
659     layout.addWidget(self.itemarea_duration_button, 0, 7)
660     self.itemarea_duration_label = QLabel()
661     self.itemarea_duration_label.setText("Dauer:")
662     layout.addWidget(self.itemarea_duration_label, 0, 8)
663     self.itemarea_duration_edit = QLineEdit()
664     self.itemarea_duration_edit.setObjectName("pap_"+number+"
          _duration_value_edit")
665     self.itemarea_duration_edit.setMaximumHeight(28)
666     self.itemarea_duration_edit.setFixedWidth(80)
667     self.itemarea_duration_edit.setVisible(True)
668     self.itemarea_duration_edit.setAlignment(Qt.AlignRight | Qt.
          AlignVCenter)
669     self.itemarea_duration_edit.setText(str(backend.pap[int(number)
          ][7]))
670     self.itemarea_duration_label.setBuddy(self.itemarea_duration_edit)
671     layout.addWidget(self.itemarea_duration_edit, 0, 9)
672     self.itemarea_duration_view = QLabel()
673     self.itemarea_duration_view.setObjectName("pap_"+number+"
          _duration_value_view")
674     self.itemarea_duration_view.setMaximumHeight(28)
675     self.itemarea_duration_view.setFixedWidth(80)
676     self.itemarea_duration_view.setVisible(False)
677     self.itemarea_duration_view.setAlignment(Qt.AlignRight | Qt.
          AlignVCenter)
678     self.itemarea_duration_view.setText(str(backend.pap[int(number)
          ][7]))
679     layout.addWidget(self.itemarea_duration_view, 0, 9)
```

```
680     self.itemarea_duration_box = QComboBox()
681     self.itemarea_duration_box.setObjectName("pap_"+number+"
          _duration_unit_edit")
682     self.itemarea_duration_box.setEditable(False)
683     self.itemarea_duration_box.setMaximumHeight(28)
684     self.itemarea_duration_box.setMinimumContentsLength(5)
685     self.itemarea_duration_box.setCurrentIndex(int(backend.pap[int(
          number)][8]))
686     layout.addWidget(self.itemarea_duration_box, 0, 10)
687     self.itemarea_duration_view_unit = QLabel()
688     self.itemarea_duration_view_unit.setObjectName("pap_"+number+"
          _duration_unit_view")
689     self.itemarea_duration_view_unit.setMaximumHeight(28)
690     self.itemarea_duration_view_unit.setVisible(False)
691     self.itemarea_duration_view_unit.setText(backend.unit[1][int(
          backend.pap[int(number)][8])])
692     layout.addWidget(self.itemarea_duration_view_unit, 0, 10)
693     #
694     self.itemarea_vline_03 = QFrame()
695     self.itemarea_vline_03.setFrameShape(QFrame.VLine)
696     self.itemarea_vline_03.setFrameShadow(QFrame.Sunken)
697     layout.addWidget(self.itemarea_vline_03, 0, 11)
698     #
699     self.itemarea_volume_button = QRadioButton(self)
700     self.itemarea_volume_button.setObjectName("pap_"+number+"_volume_ro
          ")
701     self.itemarea_volume_button.setToolTip("Volumen automatisch
          berechnen")
702     self.itemarea_volume_button.setChecked(True)
703     layout.addWidget(self.itemarea_volume_button, 0, 12)
704     self.itemarea_volume_label = QLabel()
705     self.itemarea_volume_label.setText("Volumen:")
706     layout.addWidget(self.itemarea_volume_label, 0, 13)
707     self.itemarea_volume_edit = QLineEdit()
708     self.itemarea_volume_edit.setObjectName("pap_"+number+"
          _volume_value_edit")
709     self.itemarea_volume_edit.setMaximumHeight(28)
710     self.itemarea_volume_edit.setFixedWidth(80)
711     self.itemarea_volume_edit.setVisible(False)
712     self.itemarea_volume_edit.setAlignment(Qt.AlignRight | Qt.
          AlignVCenter)
713     self.itemarea_volume_edit.setText(str(backend.pap[int(number)][9]))
714     self.itemarea_volume_label.setBuddy(self.itemarea_volume_edit)
715     layout.addWidget(self.itemarea_volume_edit, 0, 14)
716     self.itemarea_volume_view = QLabel()
717     self.itemarea_volume_view.setObjectName("pap_"+number+"
          _volume_value_view")
718     self.itemarea_volume_view.setMaximumHeight(28)
719     self.itemarea_volume_view.setFixedWidth(80)
```

```
720     self.itemarea_volume_view.setVisible(True)
721     self.itemarea_volume_view.setAlignment(Qt.AlignRight | Qt.
          AlignVCenter)
722     self.itemarea_volume_view.setText(str(backend.pap[int(number)][9]))
723     layout.addWidget(self.itemarea_volume_view, 0, 14)
724     self.itemarea_volume_box = QComboBox()
725     self.itemarea_volume_box.setObjectName("pap_"+number+"
          _volume_unit_edit")
726     self.itemarea_volume_box.setEditable(False)
727     self.itemarea_volume_box.setMaximumHeight(28)
728     self.itemarea_volume_box.setVisible(False)
729     self.itemarea_volume_box.setMinimumContentsLength(5)
730     self.itemarea_volume_box.setCurrentIndex(int(backend.pap[int(number
          )][10]))
731     layout.addWidget(self.itemarea_volume_box, 0, 15)
732     self.itemarea_volume_view_unit = QLabel()
733     self.itemarea_volume_view_unit.setObjectName("pap_"+number+"
          _volume_unit_view")
734     self.itemarea_volume_view_unit.setMaximumHeight(28)
735     self.itemarea_volume_view_unit.setVisible(True)
736     self.itemarea_volume_view_unit.setText(backend.unit[2][int(backend.
          pap[int(number)][10])])
737     layout.addWidget(self.itemarea_volume_view_unit, 0, 15)
738     #
739     self.itemarea_comment_label = QLabel()
740     self.itemarea_comment_label.setText("Beschreibung:")
741     self.itemarea_comment_label.setToolTip("Freifeld zur Beschreibung
          des Programmablaufplans, z.B. \"Spülen\"")
742     self.itemarea_comment_label.setAlignment(Qt.AlignRight | Qt.
          AlignTop)
743     layout.addWidget(self.itemarea_comment_label, 1, 2, 1, 2)
744     self.itemarea_comment_edit = QTextEdit("")
745     self.itemarea_comment_edit.setObjectName("pap_"+number+"_comment")
746     self.itemarea_comment_edit.setToolTip("Freifeld zur Beschreibung
          des Programmablaufplans, z.B. \"Spülen\"")
747     self.itemarea_comment_edit.setMaximumHeight(45)
748     self.itemarea_comment_edit.setFixedWidth(820)
749     self.itemarea_comment_edit.setVisible(True)
750     if str(backend.pap[int(number)][11]) != "0":
751      self.itemarea_comment_edit.setText(backend.pap[int(number)][11])
752     self.itemarea_comment_label.setBuddy(self.itemarea_comment_edit)
753     layout.addWidget(self.itemarea_comment_edit, 1, 4, 1, 12)
754     #
755     self.itemarea_interrupt_label = QLabel()
756     self.itemarea_interrupt_label.setText("Fertigstellung:")
757     self.itemarea_interrupt_label.setToolTip(" ")
758     self.itemarea_interrupt_label.setAlignment(Qt.AlignRight | Qt.
          AlignTop)
759     layout.addWidget(self.itemarea_interrupt_label, 2, 2, 1, 2)
```

```
760     self.itemarea_interrupt_box = QCheckBox("nach Abschluss ein Signal
          an '"+backend.interrupt_device+"' ausgeben")
761     self.itemarea_interrupt_box.setObjectName("pap_"+number+"_signal")
762     if str(backend.pap[int(number)][1]) != "0":
763      self.itemarea_interrupt_box.setCheckState(Qt.Checked)
764     else:
765      self.itemarea_interrupt_box.setCheckState(Qt.Unchecked)
766     layout.addWidget(self.itemarea_interrupt_box, 2, 4, 1, 6)
767     self.itemarea_progressbar = QProgressBar()
768     self.itemarea_progressbar.setObjectName("pap_"+number+"_progressbar
          ")
769     self.itemarea_progressbar.setMinimum(0)
770     self.itemarea_progressbar.setMaximum(100)
771     self.itemarea_progressbar.setVisible(False)
772     self.itemarea_progressbar.setOrientation(Qt.Horizontal)
773     layout.addWidget(self.itemarea_progressbar, 2, 10, 1, 6)
774     #
775     for i in range(len(backend.unit)):
776      for j in range(len(backend.unit[0])):
777       if i == 0 and backend.unit[i][j] != 0:
778        userData = '6'+number
779        self.itemarea_flowrate_box.addItem(backend.unit[i][j], userData)
780       elif i == 1 and backend.unit[i][j] != 0:
781        userData = '8'+number
782        self.itemarea_duration_box.addItem(backend.unit[i][j], userData)
783       elif i == 2 and backend.unit[i][j] != 0:
784        userData = '10'+number
785        self.itemarea_volume_box.addItem(backend.unit[i][j], userData)
786     self.itemarea_label_callback = partial(self.
          itemarea_parameter_changed_line, '0', number)
787     self.itemarea_label.stateChanged.connect(self.
          itemarea_label_callback)
788     self.itemarea_interrupt_box_callback = partial(self.
          itemarea_parameter_changed_line, '1', number, )
789     self.itemarea_interrupt_box.stateChanged.connect(self.
          itemarea_interrupt_box_callback)
790     self.itemarea_flowrate_edit_callback = partial(self.
          itemarea_parameter_changed_line, '5', number)
791     self.itemarea_flowrate_edit.textEdited.connect(self.
          itemarea_flowrate_edit_callback)
792     self.itemarea_flowrate_box.activated.connect(self.
          itemarea_parameter_changed_box)
793     self.itemarea_flowrate_button_callback = partial(self.
          itemarea_parameter_changed_radio_flowrate, '5', number)
794     self.itemarea_flowrate_button.clicked.connect(self.
          itemarea_flowrate_button_callback)
795     self.itemarea_duration_edit_callback = partial(self.
          itemarea_parameter_changed_line, '7', number)
796     self.itemarea_duration_edit.textEdited.connect(self.
```

```
      itemarea_duration_edit_callback)
797   self.itemarea_duration_box.activated.connect(self.
      itemarea_parameter_changed_box)
798   self.itemarea_duration_button_callback = partial(self.
      itemarea_parameter_changed_radio_duration, '7', number)
799   self.itemarea_duration_button.clicked.connect(self.
      itemarea_duration_button_callback)
800   self.itemarea_volume_edit_callback = partial(self.
      itemarea_parameter_changed_line, '9', number, )
801   self.itemarea_volume_edit.textEdited.connect(self.
      itemarea_volume_edit_callback)
802   self.itemarea_volume_box.activated.connect(self.
      itemarea_parameter_changed_box)
803   self.itemarea_volume_button_callback = partial(self.
      itemarea_parameter_changed_radio_volume, '9', number)
804   self.itemarea_volume_button.clicked.connect(self.
      itemarea_volume_button_callback)
805   self.itemarea_comment_edit_callback = partial(self.
      itemarea_parameter_changed_line, '11', number, )
806   self.itemarea_comment_edit.textChanged.connect(self.
      itemarea_comment_edit_callback)
807   self.itemarea_hline_01 = QFrame()
808   self.itemarea_hline_01.setFrameShape(QFrame.HLine)
809   self.itemarea_hline_01.setFrameShadow(QFrame.Sunken)
810   layout.addWidget(self.itemarea_hline_01, 3, 0, 3, 20)
811  # noinspection PyUnresolvedReferences
812  def create_input_area(self, layout):
813   # Input-Field
814   self.input_label = QLabel("Befehl:")
815   self.input_edit = QLineEdit()
816   self.input_edit.returnPressed.connect(self.input_changed)
817   self.input_edit.setMaximumHeight(28)
818   self.input_label.setBuddy(self.input_edit)
819   layout.addWidget(self.input_label)
820   layout.addWidget(self.input_edit)
821  # noinspection PyUnresolvedReferences
822  def create_output_area(self, layout):
823   self.output_label = QLabel("Konsole:")
824   self.output_label.setBuddy(self.serial_label)
825   layout.addWidget(self.output_label)
826   self.serial_label = QTextEdit()
827   self.serial_label.setObjectName("serial_label")
828   layout.addWidget(self.serial_label)
829  def refresh_clicked(self):
830   # refresh the list of availiable devices in /dev - if none there,
      show Loewe01-device
831   global settings_area
832   global main_area
833   self.device_edit.clear()
```

```
834     self.interrupt_device_edit.clear()
835     filenames = list(filter(lambda x: x.startswith('ttyACM'), os.
          listdir('/dev/')))
836     if len(filenames) == 0 and not backend.device == 'Loewe01':
837       self.device_edit.addItem("keine Pumpe gefunden, bitte
          aktualisieren")
838       self.device_edit.addItem("Loewe01")
839       backend.device = False
840       self.interrupt_device_edit.addItem("none")
841       backend.interrupt_device = False
842       self.connect_button.setVisible(False)
843       self.identify_button.setVisible(False)
844       self.pump_label.setVisible(False)
845       self.pump_version.setVisible(False)
846       self.disconnect_button.setVisible(False)
847       settings_area.setVisible(False)
848       main_area.setVisible(False)
849     else:
850       index=0
851       for name in filenames:
852         self.device_edit.addItem(filenames[index])
853         if index > 0:
854           self.interrupt_device_edit.addItem(filenames[index])
855         index += 1
856       backend.device = '/dev/' + filenames[0]
857       backend.interrupt_device = 'none'
858       self.connect_button.setVisible(True)
859       self.identify_button.setVisible(True)
860     self.get_pump_version()
861   def pump_device_changed(self):
862    # get information from pump if user changed device
863    if len(self.device_edit.currentText()) > 1 and not self.device_edit
        .currentText() == 'keine Pumpe gefunden, bitte aktualisieren':
864      if os.path.exists('/dev/' + self.device_edit.currentText()) ==
        True:
865        backend.device = '/dev/' + self.device_edit.currentText()
866        if backend.device == backend.interrupt_device:
867          backend.interrupt_device = 'none'
868      elif self.device_edit.currentText() == 'Loewe01':
869        backend.device = 'Loewe01'
870        backend.interrupt_device = 'none'
871        self.connect_button.setVisible(True)
872      else:
873        self.refresh_clicked()
874    else:
875      backend.device = False
876      backend.interrupt_device = False
877    self.get_pump_version()
878   def interrupt_device_changed(self):
```

```
879     # get information from pump if user changed device
880     global main_area
881     if len(self.interrupt_device_edit.currentText()) > 1 and not self.
          interrupt_device_edit.currentText() == 'none':
882       if os.path.exists('/dev/' + self.interrupt_device_edit.currentText
          ()) == True:
883         backend.interrupt_device = '/dev/' + self.interrupt_device_edit.
          currentText()
884         if backend.interrupt_device == device:
885           backend.interrupt_device = 'none'
886       else:
887         backend.interrupt_device = 'none'
888     else:
889       backend.interrupt_device = 'none'
890     if main_area:
891       for number in range(backend.papCounter):
892         if main_area.findChild(QCheckBox, "pap_"+str(number)+"_signal"):
893           main_area.findChild(QCheckBox, "pap_"+str(number)+"_signal").
          setText("nach Abschluss ein Signal an '"+backend.interrupt_device
          +"' ausgeben")
894   def identify_pump(self):
895     # use dimming of pump display to make choosen pump visible
896     if backend.device != False and backend.device != 'Loewe01' and os.
          path.exists(backend.device) == True:
897       ser_identify = serial.Serial(port=backend.device,baudrate=115200,
          parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_TWO,bytesize=
          serial.EIGHTBITS)
898       ser_identify.isOpen()
899       n = 10
900       for i in range(1,n):
901         ser_identify.write('dim 0\r\n'.encode('UTF-8'))
902         time.sleep(0.15)
903         ser_identify.write('dim 100\r\n'.encode('UTF-8'))
904         time.sleep(0.15)
905       ser_identify.close()
906     elif backend.device == 'Loewe01':
907       print ('on-off-on-off')
908     else:
909       self.refresh_clicked()
910   def connect_pump(self):
911     # check if serial connection availiable, then show settings_area
          and main_area
912     global settings_area
913     global main_area
914     global output_area
915     if backend.device != False and backend.device != 'Loewe01' and os.
          path.exists(backend.device) == True:
916       # set pump to echo on
917       ser = serial.Serial(port=backend.device,baudrate=115200,parity=
```

```
        serial.PARITY_NONE, stopbits=serial.STOPBITS_TWO, bytesize=serial.
        EIGHTBITS)
918     ser.isOpen()
919     ser.write(('echo on\r\n').encode('UTF-8'))
920     out = ''
921     output = ''
922     # wait one second and give device time to answer
923     time.sleep(1)
924     while ser.inWaiting() > 0:
925       out += ser.read(1).decode('UTF-8')
926     if out != '':
927       output = out + '\n' + self.serial_label.toPlainText()
928     self.serial_label.clear()
929     self.serial_label.setText(output)
930     ser.close()
931     self.connect_button.setVisible(False)
932     self.disconnect_button.setVisible(True)
933     settings_area.setVisible(True)
934     main_area.setVisible(False)
935   elif backend.device == 'Loewe01':
936     time.sleep(0.1)
937     output = 'echo on\n' + self.serial_label.toPlainText()
938     self.serial_label.clear()
939     self.serial_label.setText(output)
940     self.connect_button.setVisible(False)
941     self.disconnect_button.setVisible(True)
942     settings_area.setVisible(True)
943     main_area.setVisible(False)
944   else:
945     self.refresh_clicked()
946 def disconnect_pump(self):
947   # check if serial connection still open, then close and hide
        settings_area and main_area
948   global settings_area
949   global main_area
950   global output_area
951   self.disconnect_button.setVisible(False)
952   self.connect_button.setVisible(True)
953   self.serial_label.clear()
954   settings_area.setVisible(False)
955   main_area.setVisible(False)
956 def input_changed(self):
957   # send command to syringe-pump, show output in console
958   if backend.device != False and backend.device != 'Loewe01' and os.
        path.exists(backend.device) == True:
959     ser = serial.Serial(port=backend.device, baudrate=115200, parity=
        serial.PARITY_NONE, stopbits=serial.STOPBITS_TWO, bytesize=serial.
        EIGHTBITS)
960     ser.isOpen()
```

```
961     userinput = self.input_edit.text()
962     self.input_edit.clear()
963     ser.write((userinput+'\r\n').encode('UTF-8'))
964     out = ''
965     output = ''
966     # wait one second and give device time to answer
967     time.sleep(1)
968     while ser.inWaiting() > 0:
969      out += ser.read(1).decode('UTF-8')
970     if out != '':
971      output = out + '\n' + self.serial_label.toPlainText()
972     self.serial_label.clear()
973     self.serial_label.setText(output)
974    elif backend.device == 'Loewe01':
975     userinput = self.input_edit.text()
976     self.input_edit.clear()
977     time.sleep(1)
978     output = userinput + '\n' + self.serial_label.toPlainText()
979     self.serial_label.clear()
980     self.serial_label.setText(output)
981
982  def get_pump_version(self):
983   # receive/read version and serial from syringe-pump, set pump-
       address to zero if necessary
984   if backend.device != False and backend.device != 'Loewe01' and os.
       path.exists(backend.device) == True:
985    ser_getver = serial.Serial(port=backend.device,baudrate=115200,
       parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_TWO,bytesize=
       serial.EIGHTBITS)
986    ser_getver.isOpen()
987    ser_getver.write('ver\r\n'.encode('UTF-8'))
988    getver = ''
989    time.sleep(0.1)
990    while ser_getver.inWaiting() > 0:
991     getver += ser_getver.read(1).decode('UTF-8')
992    pump_ver = getver.splitlines()
993    getver = ''
994    ser_getver.write('version\r\n'.encode('UTF-8'))
995    getver = ''
996    time.sleep(0.1)
997    while ser_getver.inWaiting() > 0:
998     getver += ser_getver.read(1).decode('UTF-8')
999    pump_serial_full = getver.splitlines()
1000   pump_serial = pump_serial_full[3].split()
1001   if "PHD" in pump_ver[1]:# special case for old models of syringe
       pumps
1002    if int(pump_serial[2]) != 0:# set address to zero, if not already
        zero
1003     pump_ver[1] = "11 Elite"+pump_ver[1][6:]
```

```
1004        ser_getver.write('address 0\r\n'.encode('UTF-8'))
1005        time.sleep(0.1)
1006        while ser_getver.inWaiting() > 0:
1007         getver += ser_getver.read(1).decode('UTF-8')
1008        getver = ''
1009        pump_serial[2] = "(altes Modell)"
1010       else:
1011        pump_ver[1] = "11 Elite"+pump_ver[1][3:]
1012        pump_serial[2] = "(altes Modell)"
1013      if self.pump_version:
1014       self.pump_version.clear()
1015       self.pump_version.setText(pump_ver[1] + ' - ' + pump_serial[2])
1016       getver = ''
1017       self.pump_label.setVisible(True)
1018       self.pump_version.setVisible(True)
1019      ser_getver.close()
1020     elif backend.device == 'Loewe01':
1021      self.pump_version.clear()
1022      self.pump_version.setText(backend.Loewe01[0])
1023      self.pump_label.setVisible(True)
1024      self.pump_version.setVisible(True)
1025    def get_pump_settings(self):
1026     # receive/read diameter and volume of syringe from syringe-pump
1027     global main_area
1028     if backend.device != False and backend.device != 'Loewe01' and os.
         path.exists(backend.device) == True:
1029      ser_getset = serial.Serial(port=backend.device,baudrate=115200,
         parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_TWO,bytesize=
         serial.EIGHTBITS)
1030      ser_getset.isOpen()
1031      ser_getset.write('diameter\r\n'.encode('UTF-8'))
1032      time.sleep(0.1)
1033      getset = ''
1034      output = ''
1035      while ser_getset.inWaiting() > 0:
1036       getset += ser_getset.read(1).decode('UTF-8')
1037      if getset != '':
1038       output = getset + '\n' + self.serial_label.toPlainText()
1039      self.serial_label.clear()
1040      self.serial_label.setText(output)
1041      pump_syringe_diameter = getset.splitlines()
1042      getset = ''
1043      ser_getset.write('svolume\r\n'.encode('UTF-8'))
1044      time.sleep(0.1)
1045      getset = ''
1046      output = ''
1047      while ser_getset.inWaiting() > 0:
1048       getset += ser_getset.read(1).decode('UTF-8')
1049      if getset != '':
```

```
1050        output = getset + '\n' + self.serial_label.toPlainText()
1051        self.serial_label.clear()
1052        self.serial_label.setText(output)
1053        pump_syringe_volume = getset.splitlines()
1054        pump_diameter = pump_syringe_diameter[1].split()
1055        pump_volume = pump_syringe_volume[1].split()
1056        diameter_unit = 0 if pump_volume[1]=="mm" else 0
1057        volume_unit = 0 if pump_volume[1]=="ml" else 1 if pump_volume[1]==
            "ul" else 2 if pump_volume[1]=="nl" else 3 if pump_volume[1]=="pl
            " else 4 if pump_volume[1]=="fl" else 0
1058        self.settings_diameter_edit.setText(str(round(float(pump_diameter
            [0]), 2)))
1059        self.settings_volume_edit.setText(str(round(float(pump_volume[0]),
             2)))
1060        self.settings_diameter_box.setCurrentIndex(diameter_unit)
1061        self.settings_volume_box.setCurrentIndex(volume_unit)
1062        ser_getset.close()
1063        main_area.setVisible(True)
1064    elif backend.device == 'Loewe01':
1065        self.settings_diameter_edit.setText(backend.Loewe01[1])
1066        self.settings_diameter_box.setCurrentIndex(backend.Loewe01[2])
1067        self.settings_volume_edit.setText(backend.Loewe01[3])
1068        self.settings_volume_box.setCurrentIndex(backend.Loewe01[4])
1069        output = str('svolume '+self.settings_volume_edit.text()+' '+self.
            settings_volume_box.currentText()) + '\n' + str('diameter '+self.
            settings_diameter_edit.text()+' '+self.settings_diameter_box.
            currentText()) + '\n' + self.serial_label.toPlainText()
1070        self.serial_label.clear()
1071        self.serial_label.setText(output)
1072        main_area.setVisible(True)
1073    def set_pump_settings(self):
1074     # transmit/send diameter and volume of syringe to syringe-pump
1075     global main_area
1076     if backend.device != False and backend.device != 'Loewe01' and os.
            path.exists(backend.device) == True:
1077        ser_setset = serial.Serial(port=backend.device,baudrate=115200,
            parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_TWO,bytesize=
            serial.EIGHTBITS)
1078        ser_setset.isOpen()
1079        diameter_value = self.settings_diameter_edit.text().replace(',','.
            ')
1080        volume_value = self.settings_volume_edit.text().replace(',','.')
1081        if diameter_value != self.settings_diameter_edit.text():
1082         self.settings_diameter_edit.setText(diameter_value)
1083        elif volume_value != self.settings_volume_edit.text():
1084         self.settings_volume_edit.setText(volume_value)
1085        if float(diameter_value) != 0 and float(volume_value) != 0:
1086         ser_setset.write(str('diameter '+diameter_value+' '+self.
            settings_diameter_box.currentText()+'\r\n').encode('UTF-8'))
```

```
1087        time.sleep(0.1)
1088        out = ''
1089        output = ''
1090        while ser_setset.inWaiting() > 0:
1091         out += ser_setset.read(1).decode('UTF-8')
1092        if out != '':
1093         output = out + '\n' + self.serial_label.toPlainText()
1094        self.serial_label.clear()
1095        self.serial_label.setText(output)
1096        ser_setset.write(str('svolume '+volume_value+' '+self.
            settings_volume_box.currentText()+'\r\n').encode('UTF-8'))
1097        time.sleep(0.1)
1098        out = ''
1099        output = ''
1100        while ser_setset.inWaiting() > 0:
1101         out += ser_setset.read(1).decode('UTF-8')
1102        if out != '':
1103         output = out + '\n' + self.serial_label.toPlainText()
1104        self.serial_label.clear()
1105        self.serial_label.setText(output)
1106       ser_setset.close()
1107       main_area.setVisible(True)
1108      elif backend.device == 'Loewe01':
1109       diameter_value = self.settings_diameter_edit.text().replace(',','.
           ')
1110       volume_value = self.settings_volume_edit.text().replace(',','.')
1111       backend.Loewe01[1] = diameter_value
1112       backend.Loewe01[2] = self.settings_diameter_box.currentIndex()
1113       backend.Loewe01[3] = volume_value
1114       backend.Loewe01[4] = self.settings_volume_box.currentIndex()
1115       if diameter_value != self.settings_diameter_edit.text():
1116        self.settings_diameter_edit.setText(diameter_value)
1117       elif volume_value != self.settings_volume_edit.text():
1118        self.settings_volume_edit.setText(volume_value)
1119       output = str('svolume '+volume_value+' '+self.settings_volume_box.
           currentText()) + '\n' + str('diameter '+diameter_value+' '+self.
           settings_diameter_box.currentText()) + '\n' + self.serial_label.
           toPlainText()
1120       self.serial_label.clear()
1121       self.serial_label.setText(output)
1122       main_area.setVisible(True)
1123     def save_pap(self):
1124      # quicksave, if pap-array already saved to file or opened from a
           file
1125      global fname_pap
1126      if fname_pap:
1127       np.savetxt(fname_pap, backend.pap, delimiter="\t", fmt="%s")
1128     def saveto_pap(self):
1129      # save pap-array to new file, store filename for quicksave
```

```
1130     global fname_pap
1131     fname_pap_new, _ = QFileDialog.getSaveFileName(self, 'Datei
           speichern unter','.', "PAP Files (*.pap)")
1132     if fname_pap_new:
1133      np.savetxt(fname_pap_new, backend.pap, delimiter="\t", fmt="%s")
1134      fname_pap = fname_pap_new
1135      self.save_button.setEnabled(True)
1136   def new_pap(self):
1137    # clear pap-array and refresh all cells (does not remove unused pap
         -areas)
1138    global main_area
1139    for i in range(len(backend.pap)):
1140     for j in range(12):
1141      if j in (0,1,2,6,8,10):
1142       backend.pap[i][j] = 0
1143      elif j in (5,7,9):
1144       backend.pap[i][j] = 0.0
1145      else:
1146       backend.pap[i][j] = ""
1147     if main_area.findChild(QLineEdit, "pap_"+str(i)+"
        _flowrate_value_edit"):
1148      main_area.findChild(QCheckBox, "pap_"+str(i)+"_active").
        setChecked(False)
1149      main_area.findChild(QCheckBox, "pap_"+str(i)+"_signal").
        setChecked(False)
1150      main_area.findChild(QLineEdit, "pap_"+str(i)+"
        _flowrate_value_edit").setText(str(backend.pap[i][5]))
1151      main_area.findChild(QLabel, "pap_"+str(i)+"_flowrate_value_view")
        .setText(str(backend.pap[i][5]))
1152      main_area.findChild(QComboBox, "pap_"+str(i)+"_flowrate_unit_edit
        ").setCurrentIndex(int(backend.pap[i][6]))
1153      main_area.findChild(QLabel, "pap_"+str(i)+"_flowrate_unit_view").
        setText(str(backend.unit[0][int(backend.pap[i][6])]))
1154      main_area.findChild(QLineEdit, "pap_"+str(i)+"
        _duration_value_edit").setText(str(backend.pap[i][7]))
1155      main_area.findChild(QLabel, "pap_"+str(i)+"_duration_value_view")
        .setText(str(backend.pap[i][7]))
1156      main_area.findChild(QComboBox, "pap_"+str(i)+"_duration_unit_edit
        ").setCurrentIndex(int(backend.pap[i][8]))
1157      main_area.findChild(QLabel, "pap_"+str(i)+"_duration_unit_view").
        setText(str(backend.unit[0][int(backend.pap[i][8])]))
1158      main_area.findChild(QLineEdit, "pap_"+str(i)+"_volume_value_edit"
        ).setText(str(backend.pap[i][9]))
1159      main_area.findChild(QLabel, "pap_"+str(i)+"_volume_value_view").
        setText(str(backend.pap[i][9]))
1160      main_area.findChild(QComboBox, "pap_"+str(i)+"_volume_unit_edit")
        .setCurrentIndex(int(backend.pap[i][10]))
1161      main_area.findChild(QLabel, "pap_"+str(i)+"_volume_unit_view").
        setText(str(backend.unit[0][int(backend.pap[i][10])]))
```

```
1162        main_area.findChild(QTextEdit, "pap_"+str(i)+"_comment").setText(
            str(backend.pap[i][11].replace('<br />', '\n')))
1163        fname_pap = False
1164        self.save_button.setEnabled(False)
1165      self.refresh_total_duration()
1166      self.refresh_total_volume()
1167    def open_pap(self):
1168     # open saved pap from file and store in pap-array and refresh all
            cells, store filename for quicksave
1169      global fname_pap
1170      global main_area
1171      fname_pap_open, _ = QFileDialog.getOpenFileName(self, 'Datei öffnen
            ','.', "PAP Files (*.pap)")
1172      if fname_pap_open:
1173       pop = np.genfromtxt(fname_pap_open, delimiter="\t", dtype="str",
           usemask=True)
1174       fname_pap = fname_pap_open
1175       self.save_button.setEnabled(True)
1176      for i in range(len(backend.pap)):
1177       pap_in_use = 0
1178       for j in range(12):
1179        if j in (0,1,2,6,8,10):
1180         backend.pap[i][j] = int(float(pop[i][j]))
1181        elif j in (5,7,9):
1182         backend.pap[i][j] = float(pop[i][j])
1183         pap_in_use += backend.pap[i][j]# count all values to see if PAP
          empty or not
1184        else:
1185         if len(str(pop[i][j])) < 5 and str(pop[i][j]) != '0':
1186          backend.pap[i][j] = ''
1187         else:
1188          backend.pap[i][j] = str(pop[i][j])
1189       if main_area.findChild(QLineEdit, "pap_"+str(i)+"
          _flowrate_value_edit"):
1190        if backend.pap[i][0] == 1:# set the status of every PAP
1191         main_area.findChild(QCheckBox, "pap_"+str(i)+"_active").
          setChecked(True)
1192        else:
1193         main_area.findChild(QCheckBox, "pap_"+str(i)+"_active").
          setChecked(False)
1194        if backend.pap[i][1] == 1:# set the interrupt-status of every PAP
1195         main_area.findChild(QCheckBox, "pap_"+str(i)+"_signal").
          setChecked(True)
1196        else:
1197         main_area.findChild(QCheckBox, "pap_"+str(i)+"_signal").
          setChecked(False)
1198        main_area.findChild(QLineEdit, "pap_"+str(i)+"
          _flowrate_value_edit").setText(str(backend.pap[i][5]))
1199        main_area.findChild(QLabel, "pap_"+str(i)+"_flowrate_value_view")
```

```
        .setText(str(backend.pap[i][5]))
1200    main_area.findChild(QComboBox, "pap_"+str(i)+"_flowrate_unit_edit
        ").setCurrentIndex(int(backend.pap[i][6]))
1201    main_area.findChild(QLabel, "pap_"+str(i)+"_flowrate_unit_view").
        setText(str(backend.unit[0][int(backend.pap[i][6])]))
1202    main_area.findChild(QLineEdit, "pap_"+str(i)+"
        _duration_value_edit").setText(str(backend.pap[i][7]))
1203    main_area.findChild(QLabel, "pap_"+str(i)+"_duration_value_view")
        .setText(str(backend.pap[i][7]))
1204    main_area.findChild(QComboBox, "pap_"+str(i)+"_duration_unit_edit
        ").setCurrentIndex(int(backend.pap[i][8]))
1205    main_area.findChild(QLabel, "pap_"+str(i)+"_duration_unit_view").
        setText(str(backend.unit[0][int(backend.pap[i][8])]))
1206    main_area.findChild(QLineEdit, "pap_"+str(i)+"_volume_value_edit"
        ).setText(str(backend.pap[i][9]))
1207    main_area.findChild(QLabel, "pap_"+str(i)+"_volume_value_view").
        setText(str(backend.pap[i][9]))
1208    main_area.findChild(QComboBox, "pap_"+str(i)+"_volume_unit_edit")
        .setCurrentIndex(int(backend.pap[i][10]))
1209    main_area.findChild(QLabel, "pap_"+str(i)+"_volume_unit_view").
        setText(str(backend.unit[0][int(backend.pap[i][10])]))
1210    main_area.findChild(QTextEdit, "pap_"+str(i)+"_comment").setText(
        str(backend.pap[i][11].replace('<br />', '\n')))
1211   if int(pap_in_use) > 0:# show all PAPs already filled with values
1212    while backend.papCounter <= i:
1213     self.build_item_area(str(backend.papCounter))
1214     backend.papCounter = backend.papCounter+1
1215  self.refresh_total_duration()
1216  self.refresh_total_volume()
1217 def refresh_total_volume(self):
1218  # calculate total volume from values in pap-array
1219  global settings_area
1220  global main_area_header
1221  total_volume = 0.0
1222  for i in range(len(backend.pap)):
1223   factor = 1 if backend.pap[i][10] == 0 else 1/1000 if backend.pap[i
      ][10] == 1 else 1/1000000 if backend.pap[i][10] == 2 else
      1/1000000000 if backend.pap[i][10] == 3 else 0
1224   if backend.pap[i][0] == 1 and backend.pap[i][5] > 0:
1225    total_volume += (backend.pap[i][9]*factor)
1226  max_volume = float(settings_area.findChild(QLineEdit, "
      settings_volume_edit").text())
1227  max_volume_unit = 1/(10**(3*int(settings_area.findChild(QComboBox,
      "settings_volume_unit").currentIndex())))
1228  if (max_volume*max_volume_unit) < total_volume:# red output, when
      volume higher than syringe volume
1229   main_area_header.findChild(QLabel, "total_volume_view").setText("<
      font color=red>"+str(round(total_volume, 2))+" mL</font>")
1230  else:
```

```
1231      main_area_header.findChild(QLabel, "total_volume_view").setText(
            str(round(total_volume, 2))+" mL")
1232    def refresh_total_duration(self):
1233     # calculate total duration from values in pap-array
1234     global main_area_header
1235     total_duration = 0.0
1236     for i in range(len(backend.pap)):
1237      factor = 1 if backend.pap[i][8] == 0 else 1/60 if backend.pap[i
           ][8] == 1 else 60 if backend.pap[i][8] == 2 else 0
1238      if backend.pap[i][0] == 1 and backend.pap[i][5] > 0:
1239       total_duration += (backend.pap[i][7]*factor)
1240     main_area_header.findChild(QLabel, "total_duration_view").setText(
           str(round(total_duration, 2))+" min")
1241    def itemarea_parameter_changed_line(self, name='name', number=999,
         item=999):
1242     # one of the values changed, recalculate values or set activation/
         comment
1243     global main_area
1244     number = int(number)
1245     name = int(name)
1246     if name == 0: #PAP activation
1247      backend.pap[number][name] = 1 if main_area.findChild(QCheckBox, "
           pap_"+str(number)+"_active").isChecked() else 0
1248      self.refresh_total_duration()
1249      self.refresh_total_volume()
1250     elif name == 1: #PAP interrupt activation
1251      backend.pap[number][name] = 1 if main_area.findChild(QCheckBox, "
           pap_"+str(number)+"_signal").isChecked() else 0
1252     elif name == 11: #PAP comment
1253      backend.pap[number][name] = main_area.findChild(QTextEdit, "pap_"+
           str(number)+"_comment").toPlainText()
1254      backend.pap[number][name] = backend.pap[number][name].replace('\n'
           , '<br />')
1255     else:
1256      item = item.replace(',','.')
1257      if len(item) > 0:
1258       backend.pap[number][name] = float(item)
1259       self.calc_parameter(number,backend.pap[number][5],backend.pap[
           number][6],backend.pap[number][7],backend.pap[number][8],backend.
           pap[number][9],backend.pap[number][10])
1260       self.set_parameter(number)
1261       self.refresh_total_duration()
1262       self.refresh_total_volume()
1263    def itemarea_parameter_changed_box(self, index):
1264     # one of the units changed, recalculate values
1265     source = self.sender().itemData(index)
1266     number = int(source[1:])
1267     name = int(source[0])
1268     backend.pap[number][name] = int(index)
```

```
1269    self.calc_parameter(number,backend.pap[number][5],backend.pap[
          number][6],backend.pap[number][7],backend.pap[number][8],backend.
          pap[number][9],backend.pap[number][10])
1270    self.set_parameter(number)
1271    self.refresh_total_duration()
1272    self.refresh_total_volume()
1273  def itemarea_parameter_changed_radio_flowrate(self, name='name',
          number=999):
1274    # flowrate set to be readonly
1275    global main_area
1276    number = int(number)
1277    name = int(name)
1278    self.set_parameter(number)
1279    main_area.findChild(QLineEdit, "pap_"+str(number)+"
          _flowrate_value_edit").setVisible(False)
1280    main_area.findChild(QComboBox, "pap_"+str(number)+"
          _flowrate_unit_edit").setVisible(False)
1281    main_area.findChild(QLineEdit, "pap_"+str(number)+"
          _duration_value_edit").setVisible(True)
1282    main_area.findChild(QComboBox, "pap_"+str(number)+"
          _duration_unit_edit").setVisible(True)
1283    main_area.findChild(QLineEdit, "pap_"+str(number)+"
          _volume_value_edit").setVisible(True)
1284    main_area.findChild(QComboBox, "pap_"+str(number)+"
          _volume_unit_edit").setVisible(True)
1285    main_area.findChild(QLabel, "pap_"+str(number)+"
          _flowrate_value_view").setVisible(True)
1286    main_area.findChild(QLabel, "pap_"+str(number)+"_flowrate_unit_view
          ").setVisible(True)
1287    main_area.findChild(QLabel, "pap_"+str(number)+"
          _duration_value_view").setVisible(False)
1288    main_area.findChild(QLabel, "pap_"+str(number)+"_duration_unit_view
          ").setVisible(False)
1289    main_area.findChild(QLabel, "pap_"+str(number)+"_volume_value_view"
          ).setVisible(False)
1290    main_area.findChild(QLabel, "pap_"+str(number)+"_volume_unit_view")
          .setVisible(False)
1291  def itemarea_parameter_changed_radio_duration(self, name='name',
          number=999):
1292    # duration set to be readonly
1293    global main_area
1294    number = int(number)
1295    name = int(name)
1296    self.set_parameter(number)
1297    main_area.findChild(QLineEdit, "pap_"+str(number)+"
          _flowrate_value_edit").setVisible(True)
1298    main_area.findChild(QComboBox, "pap_"+str(number)+"
          _flowrate_unit_edit").setVisible(True)
1299    main_area.findChild(QLineEdit, "pap_"+str(number)+"
```

```
    _duration_value_edit").setVisible(False)
1300  main_area.findChild(QComboBox, "pap_"+str(number)+"
    _duration_unit_edit").setVisible(False)
1301  main_area.findChild(QLineEdit, "pap_"+str(number)+"
    _volume_value_edit").setVisible(True)
1302  main_area.findChild(QComboBox, "pap_"+str(number)+"
    _volume_unit_edit").setVisible(True)
1303  main_area.findChild(QLabel, "pap_"+str(number)+"
    _flowrate_value_view").setVisible(False)
1304  main_area.findChild(QLabel, "pap_"+str(number)+"_flowrate_unit_view
    ").setVisible(False)
1305  main_area.findChild(QLabel, "pap_"+str(number)+"
    _duration_value_view").setVisible(True)
1306  main_area.findChild(QLabel, "pap_"+str(number)+"_duration_unit_view
    ").setVisible(True)
1307  main_area.findChild(QLabel, "pap_"+str(number)+"_volume_value_view"
    ).setVisible(False)
1308  main_area.findChild(QLabel, "pap_"+str(number)+"_volume_unit_view")
    .setVisible(False)
1309 def itemarea_parameter_changed_radio_volume(self, name='name',
    number=999):
1310  # volume set to be readonly
1311  global main_area
1312  number = int(number)
1313  name = int(name)
1314  self.set_parameter(number)
1315  main_area.findChild(QLineEdit, "pap_"+str(number)+"
    _flowrate_value_edit").setVisible(True)
1316  main_area.findChild(QComboBox, "pap_"+str(number)+"
    _flowrate_unit_edit").setVisible(True)
1317  main_area.findChild(QLineEdit, "pap_"+str(number)+"
    _duration_value_edit").setVisible(True)
1318  main_area.findChild(QComboBox, "pap_"+str(number)+"
    _duration_unit_edit").setVisible(True)
1319  main_area.findChild(QLineEdit, "pap_"+str(number)+"
    _volume_value_edit").setVisible(False)
1320  main_area.findChild(QComboBox, "pap_"+str(number)+"
    _volume_unit_edit").setVisible(False)
1321  main_area.findChild(QLabel, "pap_"+str(number)+"
    _flowrate_value_view").setVisible(False)
1322  main_area.findChild(QLabel, "pap_"+str(number)+"_flowrate_unit_view
    ").setVisible(False)
1323  main_area.findChild(QLabel, "pap_"+str(number)+"
    _duration_value_view").setVisible(False)
1324  main_area.findChild(QLabel, "pap_"+str(number)+"_duration_unit_view
    ").setVisible(False)
1325  main_area.findChild(QLabel, "pap_"+str(number)+"_volume_value_view"
    ).setVisible(True)
1326  main_area.findChild(QLabel, "pap_"+str(number)+"_volume_unit_view")
```

```
    .setVisible(True)
1327 def set_parameter(self, number):
1328  # refresh flowrate-duration-volume (e.g. after calculation)
1329  global main_area
1330  if main_area.findChild(QRadioButton, "pap_"+str(number)+"
       _flowrate_ro").isChecked(): #auto flowrate
1331   objectname = "pap_"+str(number)+"_flowrate"
1332   main_area.findChild(QLineEdit, objectname+"_value_edit").setText(
       str(backend.pap[number][5]))
1333   main_area.findChild(QLabel, objectname+"_value_view").setText(str(
       backend.pap[number][5]))
1334   main_area.findChild(QComboBox, objectname+"_unit_edit").
       setCurrentIndex(int(backend.pap[number][6]))
1335   main_area.findChild(QLabel, objectname+"_unit_view").setText(str(
       backend.unit[0][int(backend.pap[number][6])]))
1336  elif main_area.findChild(QRadioButton, "pap_"+str(number)+"
       _duration_ro").isChecked(): #auto duration
1337   objectname = "pap_"+str(number)+"_duration"
1338   main_area.findChild(QLineEdit, objectname+"_value_edit").setText(
       str(backend.pap[number][7]))
1339   main_area.findChild(QLabel, objectname+"_value_view").setText(str(
       backend.pap[number][7]))
1340   main_area.findChild(QComboBox, objectname+"_unit_edit").
       setCurrentIndex(int(backend.pap[number][8]))
1341   main_area.findChild(QLabel, objectname+"_unit_view").setText(str(
       backend.unit[1][int(backend.pap[number][8])]))
1342  elif main_area.findChild(QRadioButton, "pap_"+str(number)+"
       _volume_ro").isChecked(): #auto volume
1343   objectname = "pap_"+str(number)+"_volume"
1344   main_area.findChild(QLineEdit, objectname+"_value_edit").setText(
       str(backend.pap[number][9]))
1345   main_area.findChild(QLabel, objectname+"_value_view").setText(str(
       backend.pap[number][9]))
1346   main_area.findChild(QComboBox, objectname+"_unit_edit").
       setCurrentIndex(int(backend.pap[number][10]))
1347   main_area.findChild(QLabel, objectname+"_unit_view").setText(str(
       backend.unit[2][int(backend.pap[number][10])]))
1348 def calc_parameter(self, number, flowrate=0, flowrate_unit=0,
     duration=0, duration_unit=0, volume=0, volume_unit=0):
1349  # calculate flowrate-duration-volume in pap-area
1350  global main_area
1351  if main_area.findChild(QRadioButton, "pap_"+str(number)+"
       _flowrate_ro").isChecked(): #calculate flowrate
1352   flowrate = 0
1353   if int(duration) != 0:
1354    flowrate = volume/duration
1355    if volume_unit == 0:
1356     flowrate_unit=0 if duration_unit==0 else 1 if duration_unit==1
     else 2 if duration_unit==2 else 0
```

```
1357        elif volume_unit == 1:
1358         flowrate_unit=4 if duration_unit==0 else 3 if duration_unit==1
            else 5 if duration_unit==2 else 0
1359        elif volume_unit == 2:
1360         flowrate_unit=7 if duration_unit==0 else 6 if duration_unit==1
            else 8 if duration_unit==2 else 0
1361        elif volume_unit == 3:
1362         flowrate_unit=10 if duration_unit==0 else 9 if duration_unit==1
            else 11 if duration_unit==2 else 0
1363        backend.pap[number][6] = flowrate_unit
1364       backend.pap[number][5] = round(flowrate, 2)
1365      elif main_area.findChild(QRadioButton, "pap_"+str(number)+"
           _volume_ro").isChecked(): #calculate volume
1366       volume = 0
1367       if flowrate != 0 and duration != 0:
1368        # flowrate in xL/min, duration in min
1369        if flowrate_unit == 0 or flowrate_unit == 4 or flowrate_unit == 7
           or flowrate_unit == 10:
1370         if duration_unit == 0:
1371          volume = flowrate*duration
1372         elif duration_unit == 1:
1373          volume = flowrate*duration/60
1374         elif duration_unit == 2:
1375          volume = flowrate*duration*60
1376         volume_unit=0 if flowrate_unit==0 else 1 if flowrate_unit==4
            else 2 if flowrate_unit==7 else 3 if flowrate_unit==10 else 0
1377        # flowrate in xL/sec, duration in sec
1378        elif flowrate_unit == 1 or flowrate_unit == 3 or flowrate_unit ==
           6 or flowrate_unit == 9:
1379         if duration_unit == 0:
1380          volume = flowrate*duration*60
1381         elif duration_unit == 1:
1382          volume = flowrate*duration
1383         elif duration_unit == 2:
1384          volume = flowrate*duration*3600
1385         volume_unit=0 if flowrate_unit==1 else 1 if flowrate_unit==3
            else 2 if flowrate_unit==6 else 3 if flowrate_unit==9 else 0
1386        # flowrate in xL/hour, duration in hour
1387        elif flowrate_unit == 2 or flowrate_unit == 5 or flowrate_unit ==
           8 or flowrate_unit == 11:
1388         if duration_unit == 0:
1389          volume = flowrate*duration/60
1390         elif duration_unit == 1:
1391          volume = flowrate*duration/3600
1392         elif duration_unit == 2:
1393          volume = flowrate*duration
1394         volume_unit=0 if flowrate_unit==2 else 1 if flowrate_unit==5
            else 2 if flowrate_unit==8 else 3 if flowrate_unit==11 else 0
1395        backend.pap[number][10] = volume_unit
```

```
1396        backend.pap[number][9] = round(volume, 2)
1397    elif main_area.findChild(QRadioButton, "pap_"+str(number)+"
            _duration_ro").isChecked(): #calculate duration
1398        duration = 0
1399        if flowrate != 0 and volume != 0:
1400            # flowrate in mL/*, volume in mL
1401            if flowrate_unit == 0 or flowrate_unit == 1 or flowrate_unit ==
            2:
1402                if volume_unit == 1:
1403                    volume = volume/1000
1404                elif volume_unit == 2:
1405                    volume = volume/1000000
1406                elif volume_unit == 3:
1407                    volume = volume/1000000000
1408                duration = volume/flowrate
1409                duration_unit=0 if flowrate_unit==0 else 1 if flowrate_unit==1
            else 2 if flowrate_unit==2 else 0
1410            # flowrate in uL/*, volume in uL
1411            elif flowrate_unit == 3 or flowrate_unit == 4 or flowrate_unit ==
            5:
1412                if volume_unit == 0:
1413                    volume = volume*1000
1414                elif volume_unit == 2:
1415                    volume = volume/1000
1416                elif volume_unit == 3:
1417                    volume = volume/1000000
1418                duration = volume/flowrate
1419                duration_unit=0 if flowrate_unit==0 else 1 if flowrate_unit==1
            else 2 if flowrate_unit==2 else 0
1420            # flowrate in nL/*, volume in nL
1421            elif flowrate_unit == 6 or flowrate_unit == 7 or flowrate_unit ==
            8:
1422                if volume_unit == 0:
1423                    volume = volume*1000000
1424                elif volume_unit == 1:
1425                    volume = volume*1000
1426                elif volume_unit == 3:
1427                    volume = volume/1000
1428                duration = volume/flowrate
1429                duration_unit=0 if flowrate_unit==0 else 1 if flowrate_unit==1
            else 2 if flowrate_unit==2 else 0
1430            # flowrate in xL/min, volume in pL
1431            elif flowrate_unit == 9 or flowrate_unit == 10 or flowrate_unit
            == 11:
1432                if volume_unit == 0:
1433                    volume = volume*1000000000
1434                elif volume_unit == 1:
1435                    volume = volume*1000000
1436                elif volume_unit == 2:
```

```
1437            volume = volume*1000
1438           duration = volume/flowrate
1439           duration_unit=0 if flowrate_unit==0 else 1 if flowrate_unit==1
       else 2 if flowrate_unit==2 else 0
1440           backend.pap[number][8] = duration_unit
1441          backend.pap[number][7] = round(duration, 2)
1442      def add_pap(self):
1443       # add another pap-area (up to 20)
1444       if backend.papCounter < 20:
1445        self.build_item_area(str(backend.papCounter))
1446       backend.papCounter = backend.papCounter+1
1447      def pap_set_colors(self):
1448       global pump_01_area
1449       global main_area
1450       if backend.start_pump == True:# program running
1451        if pump_01_area.findChild(QFrame, "serial_connection_area"):
1452         pump_01_area.findChild(QFrame, "serial_connection_area").
       setEnabled(False)
1453        if pump_01_area.findChild(QFrame, "settings_area"):
1454         pump_01_area.findChild(QFrame, "settings_area").setEnabled(False)
1455        for number in range(backend.papCounter):
1456        # set the background-color of every processed pap to light green
1457         if backend.pap[int(number)][0] == 1 and  backend.pap[int(number)
       ][2] == 1:
1458          if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1459           main_area.findChild(QFrame, "item_"+str(number)+"_area").
       setStyleSheet("QFrame { background-color: rgb(212, 255, 199); }")
1460        # set the background-color of the active running pap to light red
1461         elif backend.pap[int(number)][0] == 1 and  backend.pap[int(number
       )][2] == 2:
1462          if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1463           main_area.findChild(QFrame, "item_"+str(number)+"_area").
       setStyleSheet("QFrame { background-color: rgb(252, 243, 156); }")
1464        # set the background-color of the active running pap to light blue
1465         elif backend.pap[int(number)][0] == 1 and  backend.pap[int(number
       )][2] == 0:
1466          test = "test"
1467         else:
1468          if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1469           main_area.findChild(QFrame, "item_"+str(number)+"_area").
       setVisible(False)
1470         main_area.findChild(QFrame, "item_"+str(number)+"_area").
       setEnabled(False)
1471       elif backend.pause_pump == True:# program paused
1472        if pump_01_area.findChild(QFrame, "serial_connection_area"):
1473         pump_01_area.findChild(QFrame, "serial_connection_area").
       setEnabled(False)
1474        if pump_01_area.findChild(QFrame, "settings_area"):
1475         pump_01_area.findChild(QFrame, "settings_area").setEnabled(False)
```

```
1476        for number in range(backend.papCounter):
1477        # set the background-color of every processed pap to light green
1478            if backend.pap[int(number)][0] == 1 and  backend.pap[int(number)][2] == 1:
1479                if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1480                    main_area.findChild(QFrame, "item_"+str(number)+"_area").setStyleSheet("QFrame { background-color: rgb(212, 255, 199); }")
1481        # set the background-color of the active running pap to light red
1482            elif backend.pap[int(number)][0] == 1 and  backend.pap[int(number)][2] == 2:
1483                if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1484                    main_area.findChild(QFrame, "item_"+str(number)+"_area").setStyleSheet("QFrame { background-color: rgb(255, 194, 156); }")
1485        # set the background-color of the active running pap to light blue
1486            elif backend.pap[int(number)][0] == 1 and  backend.pap[int(number)][2] == 0:
1487                test = "test"
1488            else:
1489                if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1490                    main_area.findChild(QFrame, "item_"+str(number)+"_area").setVisible(False)
1491                    main_area.findChild(QFrame, "item_"+str(number)+"_area").setEnabled(False)
1492        else:
1493            if pump_01_area.findChild(QFrame, "serial_connection_area"):
1494                pump_01_area.findChild(QFrame, "serial_connection_area").setEnabled(True)
1495            if pump_01_area.findChild(QFrame, "settings_area"):
1496                pump_01_area.findChild(QFrame, "settings_area").setEnabled(True)
1497            for number in range(backend.papCounter):
1498                main_area.findChild(QFrame, "item_"+str(number)+"_area").setStyleSheet("QFrame { background-color: rgb(246, 244, 242); }"
                    +
1499                                    "QTextEdit { background-color: rgb(255, 255, 255); }")
1500                main_area.findChild(QFrame, "item_"+str(number)+"_area").setVisible(True)
1501                main_area.findChild(QFrame, "item_"+str(number)+"_area").setEnabled(True)
1502    def pap_start(self):
1503        if backend.status == 0:
1504            backend.start_pump = True
1505            backend.pause_pump = False
1506            backend.stop_pump = False
1507        elif backend.status == 2:
1508            backend.start_pump = True
1509            backend.pause_pump = False
1510            backend.stop_pump = False
1511    def pap_pause(self):
```

```
1512    if backend.status == 1:
1513     backend.pause_pump = True
1514     backend.start_pump = False
1515     backend.stop_pump = False
1516   def pap_stop(self):
1517    backend.start_pump = False
1518    backend.pause_pump = False
1519    backend.stop_pump = True
1520   @Slot(str)
1521   def receiveEvent(event):
1522    global output_area
1523    global main_area
1524    global pap_sc_area
1525    global pump_01_area
1526    if int(event[0:2]) == 10:#print output to console
1527     output = ''
1528     output = backend.output + '\n' + output_area.findChild(QTextEdit,
        "serial_label").toPlainText()
1529     output_area.findChild(QTextEdit, "serial_label").clear()
1530     output_area.findChild(QTextEdit, "serial_label").setText(output)
1531    if int(event[0:2]) == 11:#update progressbar
1532     main_area.findChild(QProgressBar, "pap_"+str(int(event[3:5]))+"
        _progressbar").setValue(int(event[6:]))
1533    elif int(event[0:2]) == 12:#modify ui when started
1534     main_area.findChild(QToolButton, "new_button").setEnabled(False)
1535     main_area.findChild(QToolButton, "open_button").setEnabled(False)
1536     main_area.findChild(QToolButton, "save_button").setEnabled(False)
1537     main_area.findChild(QToolButton, "saveto_button").setEnabled(False
        )
1538     main_area.findChild(QToolButton, "start_button").setEnabled(False)
1539     main_area.findChild(QToolButton, "pause_button").setEnabled(True)
1540     main_area.findChild(QToolButton, "stop_button").setEnabled(True)
1541     main_area.findChild(QPushButton, "pap_addbutton").setEnabled(False
        )
1542     if pump_01_area.findChild(QFrame, "serial_connection_area"):
1543      pump_01_area.findChild(QFrame, "serial_connection_area").
        setEnabled(False)
1544     if pump_01_area.findChild(QFrame, "settings_area"):
1545      pump_01_area.findChild(QFrame, "settings_area").setEnabled(False)
1546     for number in range(backend.papCounter):
1547     # set the background-color of every processed pap to light green
1548      if backend.pap[int(number)][0] == 1 and  backend.pap[int(number)
        ][2] == 1:
1549       if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1550        main_area.findChild(QFrame, "item_"+str(number)+"_area").
       setStyleSheet("QFrame { background-color: rgb(212, 255, 199); }")
1551     # set the background-color of the active running pap to light red
1552      elif backend.pap[int(number)][0] == 1 and  backend.pap[int(number
        )][2] == 2:
```

```
1553            if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1554                main_area.findChild(QFrame, "item_"+str(number)+"_area").
                setStyleSheet("QFrame { background-color: rgb(252, 243, 156); }")
1555            # set the background-color of the active running pap to light blue
1556            elif backend.pap[int(number)][0] == 1 and  backend.pap[int(number
                )][2] == 0:
1557                test = "test"
1558            else:
1559                if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1560                    main_area.findChild(QFrame, "item_"+str(number)+"_area").
                    setVisible(False)
1561                main_area.findChild(QFrame, "item_"+str(number)+"_area").
                setEnabled(False)
1562                main_area.findChild(QProgressBar, "pap_"+str(number)+"
                _progressbar").setVisible(True)
1563        elif int(event[0:2]) == 13:#modify ui when paused
1564            main_area.findChild(QToolButton, "new_button").setEnabled(False)
1565            main_area.findChild(QToolButton, "open_button").setEnabled(False)
1566            main_area.findChild(QToolButton, "save_button").setEnabled(False)
1567            main_area.findChild(QToolButton, "saveto_button").setEnabled(False
                )
1568            main_area.findChild(QToolButton, "start_button").setEnabled(True)
1569            main_area.findChild(QToolButton, "pause_button").setEnabled(False)
1570            main_area.findChild(QToolButton, "stop_button").setEnabled(True)
1571            main_area.findChild(QPushButton, "pap_addbutton").setEnabled(False
                )
1572            if pump_01_area.findChild(QFrame, "serial_connection_area"):
1573                pump_01_area.findChild(QFrame, "serial_connection_area").
                setEnabled(False)
1574            if pump_01_area.findChild(QFrame, "settings_area"):
1575                pump_01_area.findChild(QFrame, "settings_area").setEnabled(False)
1576            for number in range(backend.papCounter):
1577            # set the background-color of every processed pap to light green
1578                if backend.pap[int(number)][0] == 1 and  backend.pap[int(number)
                ][2] == 1:
1579                    if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1580                        main_area.findChild(QFrame, "item_"+str(number)+"_area").
                    setStyleSheet("QFrame { background-color: rgb(212, 255, 199); }")
1581            # set the background-color of the active running pap to light red
1582                elif backend.pap[int(number)][0] == 1 and  backend.pap[int(number
                )][2] == 2:
1583                    if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1584                        main_area.findChild(QFrame, "item_"+str(number)+"_area").
                    setStyleSheet("QFrame { background-color: rgb(255, 194, 156); }")
1585            # set the background-color of the active running pap to light blue
1586                elif backend.pap[int(number)][0] == 1 and  backend.pap[int(number
                )][2] == 0:
1587                    test = "test"
1588                else:
```

```
1589            if main_area.findChild(QCheckBox, "pap_"+str(number)+"_active"):
1590                main_area.findChild(QFrame, "item_"+str(number)+"_area").
                setVisible(False)
1591                main_area.findChild(QFrame, "item_"+str(number)+"_area").
                setEnabled(False)
1592                main_area.findChild(QProgressBar, "pap_"+str(number)+"
                _progressbar").setVisible(True)
1593        elif int(event[0:2]) == 14:#modify ui when stopped
1594            main_area.findChild(QToolButton, "new_button").setEnabled(True)
1595            main_area.findChild(QToolButton, "open_button").setEnabled(True)
1596            main_area.findChild(QToolButton, "save_button").setEnabled(True)
1597            main_area.findChild(QToolButton, "saveto_button").setEnabled(True)
1598            main_area.findChild(QToolButton, "start_button").setEnabled(True)
1599            main_area.findChild(QToolButton, "pause_button").setEnabled(False)
1600            main_area.findChild(QToolButton, "stop_button").setEnabled(False)
1601            main_area.findChild(QPushButton, "pap_addbutton").setEnabled(True)
1602            if pump_01_area.findChild(QFrame, "serial_connection_area"):
1603                pump_01_area.findChild(QFrame, "serial_connection_area").
                setEnabled(True)
1604            if pump_01_area.findChild(QFrame, "settings_area"):
1605                pump_01_area.findChild(QFrame, "settings_area").setEnabled(True)
1606            for number in range(backend.papCounter):
1607                main_area.findChild(QFrame, "item_"+str(number)+"_area").
                setStyleSheet("QFrame { background-color: rgb(246, 244, 242); }"
                +
1608                              "QTextEdit { background-color: rgb(255, 255,
                255); }")
1609                main_area.findChild(QFrame, "item_"+str(number)+"_area").
                setVisible(True)
1610                main_area.findChild(QFrame, "item_"+str(number)+"_area").
                setEnabled(True)
1611                main_area.findChild(QProgressBar, "pap_"+str(number)+"
                _progressbar").reset()
1612                main_area.findChild(QProgressBar, "pap_"+str(number)+"
                _progressbar").setVisible(False)
1613        elif int(event[0:2]) == 15:#focus on current pap
1614            pap_sc_area.ensureWidgetVisible(main_area.findChild(QCheckBox, "
                pap_"+str(int(event[3:5]))+"_active"))
1615        elif int(event[0:2]) == 16:#display finished dialog
1616            finished_msg = QMessageBox()
1617            finished_msg.setText("Alle Pumpvorgänge wurden abgeschlossen!")
1618            finished_msg.setIconPixmap(os.path.join('icons', 'syringe_small.
                png'))
1619            finished_msg.exec_()
1620        elif int(event[0:2]) == 17:#display stalled dialog
1621            stalled_msg = QMessageBox()
1622            stalled_msg.setText("Maximale Kraft erreicht, Pumpe blockiert!")
1623            stalled_msg.setIcon(QMessageBox.Critical)
1624            stalled_msg.exec_()
```

```
1625  def closeEvent(self, event):
1626   # stop all threads
1627   backend.end_threads = True
1628   time.sleep(0.2)
1629   if backend.can_exit:
1630    event.accept() # let the window close
1631   else:
1632    event.ignore()
```

Anhang B

Datenblätter

Stand 01.08.2018

- Datenblatt Arduino Uno
  `https://www.farnell.com/datasheets/1682209.pdf`
- Datenblatt Dallas DS18S20
  `https://datasheets.maximintegrated.com/en/ds/DS18S20.pdf`
- Datenblatt Everlight ITR8307
  `http://www.everlight.com/file/ProductFile/ITR8307.pdf`
- Datenblatt Everlight ITR9904
  `http://www.everlight.com/file/ProductFile/ITR9904.pdf`
- Datenblatt Microchip MCP3008
  `http://ww1.microchip.com/downloads/en/DeviceDoc/21295C.pdf`
- Datenblatt Microchip MCP3208
  `http://ww1.microchip.com/downloads/en/DeviceDoc/21298c.pdf`
- Datenblatt Meder Electronic DIP05-1C90-51L
  `http://www.farnell.com/datasheets/91165.pdf`
- Datenblatt Microchip MCP4261
  `http://ww1.microchip.com/downloads/en/DeviceDoc/22059b.pdf`
- Datenblatt Maxim MAX5250AEPP
  `https://datasheets.maximintegrated.com/en/ds/MAX5250.pdf`
- Datenblatt Kel-F®
  `http://www.laminatedplastics.com/kel-f.pdf`

- Datenblatt FC-40®
  http://multimedia.3m.com/mws/media/64888O/fluorinert-electronic-liquid-fc-40.pdf
- Datenblatt Gluetec EPOT5.S25
  https://gluetec-industrieklebstoffe.de/sites/default/files/field/file/tdb/epoxy-transparent-5-min-tdb.pdf
- Datenblatt Melexis MLX90621
  https://www.melexis.com/-/media/files/documents/datasheets/mlx90621-datasheet-melexis.pdf
- Datenblatt Adafruit Schrittmotor (ID 324)
  https://cdn-shop.adafruit.com/product-files/324/C140-A+datasheet.jpg
  https://www.adafruit.com/product/324
- Datenblatt Bivar UV5TZ-390-30
  https://www.bivar.com/portals/0/products/UV5TZ-XXX-XX.pdf
- Datenblatt Osram SFH 309
  https://www.osram.com/media/resource/hires/osram-dam-2495962/SFH 309.pdf
- Datenblatt Adafruit MotorShield (ID 81)
  https://www.adafruit.com/product/81
- Datenblatt Texas Instruments L293D
  http://www.ti.com/lit/ds/symlink/l293.pdf
- Datenblatt FRAC100
  https://www.gelifesciences.co.jp/tech_support/manual/pdf/56104677al.pdf
- Datenblatt Harvard Apparatus Pump 11 Elite
  https://www.harvardapparatus.com/media/manuals/Product Manuals/11 Elite Manual_5420-002REV1.0.pdf
- Datenblatt B.Braun Injekt®Solo Einmalspritzen
  https://www.bbraun.de/de/products/b0/injekt-solo.html

Literaturverzeichnis

[1] A. I. Stankiewicz, J. A. Moulijn, Process intensification: Transforming chemical engineering (vol 96, pg 22, 2000), *Chem. Eng. Prog.*, **2000**, *96*, 2.

[2] V. Hessel, H. Löwe, Organische Synthese mit mikrostrukturierten Reaktoren, *Chemie Ingenieur Technik*, **2004**, *76*, 5, 535–554.

[3] V. Hessel, S. Hardt, H. Löwe, *Chemical Micro Process Engineering: Fundamentals, Modelling and Reactions*, Wiley, **2006**. ISBN: 978-3-52760-537-8.

[4] N.-T. Nguyen, Z. Wu, Micromixers—a review, *Journal of Micromechanics and Microengineering*, **2005**, *15*, 2, R1.

[5] V. Hessel, A. Renken, J. Schouten, J. Yoshida, *Micro process engineering : a comprehensive handbook*, Wiley-VCH Verlag, Germany, **2009**. ISBN: 978-3-52731-550-5.

[6] H. Löwe, V. Hessel, A. Mueller, Microreactors. Prospects already achieved and possible misuse, *Pure and applied chemistry*, **2002**, *74*, 12, 2271–2276.

[7] V. Taly, B. T. Kelly, A. D. Griffiths, Droplets as microreactors for high-throughput biology, *ChemBioChem*, **2007**, *8*, 3, 263–272.

[8] H. Pennemann, V. Hessel, H. Löwe, Chemical microprocess technology—from laboratory-scale to production, *Chemical Engineering Science*, **2004**, *59*, 22, 4789 – 4794. ISCRE18.

[9] A. M. Foudeh, T. F. Didar, T. Veres, M. Tabrizian, Microfluidic designs and techniques using lab-on-a-chip devices for pathogen detection for point-of-care diagnostics, *Lab on a Chip*, **2012**, *12*, 18, 3249–3266.

[10] D. Kirschneck, G. Tekautz, Integration of a microreactor in an existing production plant, *Chemical Engineering & Technology: Industrial Chemistry-Plant Equipment-Process Engineering-Biotechnology*, **2007**, *30*, 3, 305–308.

[11] A. Knauer, J. M. Köhler, *Micro-Segmented Flow*, Springer, **2014**, 149–200.

[12] Y. H. Ghallab, W. Badawy, *Lab-on-a-chip: techniques, circuits, and biomedical applications*, Artech House, **2010**. ISBN: 978-1-59693-418-4.

[13] E. Kumacheva, P. Garstecki, *Microfluidic reactors for polymer particles*, John Wiley & Sons, **2011**. ISBN: 978-0-47005-773-5.

[14] K. Jähnisch, V. Hessel, H. Löwe, M. Baerns, Chemie in Mikrostrukturreaktoren, *Angewandte Chemie*, **2004**, *116*, 4, 410–451.

[15] T. Illg, P. Löb, V. Hessel, Flow chemistry using milli-and microstructured reactors—from conventional to novel process windows, *Bioorganic & medicinal chemistry*, **2010**, *18*, 11, 3707–3719.

[16] T. Razzaq, C. O. Kappe, Continuous flow organic synthesis under high-temperature/pressure conditions, *Chemistry–An Asian Journal*, **2010**, *5*, 6, 1274–1289.

[17] Y. Voloshin, R. Halder, A. Lawal, Kinetics of hydrogen peroxide synthesis by direct combination of H2 and O2 in a microreactor, *Catalysis Today*, **2007**, *125*, 1-2, 40–47.

[18] J.-i. Yoshida, *Flash chemistry: fast organic synthesis in microsystems*, John Wiley & Sons, **2008**. ISBN: 978-0-47003-586-3.

[19] D. Bošković, *Experimentelle Bestimmung und Modellierung des Verweilzeitverhaltens mikrofluidischer Strukturen*, Fraunhofer-Verlag, **2010**. ISBN: 978-3-83960-263-8.

[20] V. Misuk, *Tropfenbasierte Multiphasen – Mikroreaktionstechnologie in organisch-chemischer Synthese*, Dissertation, Johannes Gutenberg-Universität Mainz, **2014**.

[21] V. Misuk, A. Mai, Y. Zhao, J. Heinrich, D. Rauber, K. Giannopoulos, H. Löwe, Active Mixing Inside Double Emulsion Segments in Continuous Flow, *Journal of Flow Chemistry*, **2015**, *5*, 2, 101–109.

[22] V. Misuk, A. Mai, K. Giannopoulos, D. Karl, J. Heinrich, D. Rauber, H. Löwe, Palladium-Catalyzed Carbon–Carbon Cross-Coupling Reactions in Thermomorphous Double Emulsions, *Journal of Flow Chemistry*, **2015**, *5*, 1, 43–47.

[23] V. Misuk, A. Mai, K. Giannopoulos, F. Alobaid, B. Epple, H. Loewe, Micro magnetofluidics: droplet manipulation of double emulsions based on paramagnetic ionic liquids, *Lab on a Chip*, **2013**, *13*, 23, 4542–4548.

[24] T. Nisisako, T. Torii, Formation of Biphasic Janus Droplets in a Microfabricated Channel for the Synthesis of Shape-Controlled Polymer Microparticles, *Advanced materials*, **2007**, *19*, 11, 1489–1493.

[25] E. Bormashenko, Y. Bormashenko, R. Pogreb, O. Gendelman, Janus droplets: liquid marbles coated with dielectric/semiconductor particles, *Langmuir*, **2010**, *27*, 1, 7–10.

[26] T. Nisisako, Recent advances in microfluidic production of Janus droplets and particles, *Current Opinion in Colloid & Interface Science*, **2016**, *25*, 1–12.

[27] J. Jeong, A. Gross, W.-S. Wei, F. Tu, D. Lee, P. J. Collings, A. Yodh, Liquid crystal Janus emulsion droplets: preparation, tumbling, and swimming, *Soft Matter*, **2015**, *11*, 34, 6747–6754.

[28] N. Prasad, J. Perumal, C.-H. Choi, C.-S. Lee, D.-P. Kim, Generation of monodisperse inorganic–organic janus microspheres in a microfluidic device, *Advanced Functional Materials*, **2009**, *19*, 10, 1656–1662.

[29] M. Banzi, Arduino Mikrocontroller, `https://www.arduino.cc`, abgerufen am 01.08.2018.

[30] M. Banzi, *Getting Started with Arduino*, O'Reilly, **2009**. ISBN: 978-0-59615-552-0.

[31] M. Schmidt, *Arduino Ein schneller Einstieg in die Microcontroller-Entwicklung*, dpunkt, Heidelberg, **2011**. ISBN: 978-3-89864-764-9.

[32] U. Sommer, *Arduino: Mikrocontroller-Programmierung mit Arduino, Freeduino*, Franzis, Poing, **2010**. ISBN: 978-3-64565-034-2.

[33] G. Spanner, *Arduino: Schaltungsprojekte für Profis*, Elektor, Aachen, **2012**. ISBN: 978-3-89576-257-4.

[34] H. Timmis, *Arduino in der Praxis*, Franzis, Haar, **2012**. ISBN: 978-3-64565-132-5.

[35] D. Mellis, M. Banzi, D. Cuartielles, T. Igoe, Arduino: An Open Electronics Prototyping Platform. alt. chi section of the CHI 2007conference in San Jose (CA), *Im Erscheinen*, **2007**, *Vol. 2007*.

[36] J. Nguyen, Direct-to-digital temperature sensor, **2001**. US Patent 6,215,635.

[37] F. Pfeifer, F. Schäffer, Termika Fotilo, *Make*, **2017**, *3*, 16 – 25.

[38] R. Axinte, *Wärmeaustausch in Heat Pipe-kontrollierten mikrostrukturierten Reaktoren und Darstellung von magnetischen Flüssigkeiten*, Diplomarbeit, Johannes Gutenberg-Universität Mainz, **2009**.

[39] B. Fry, C. Reas, Processing Entwicklungsumgebung, `https://www.processing.org`, abgerufen am 01.08.2018.

[40] J. Heinrich, *Entwicklung eines aktiven Phasentrenners für mikrofluidische Systeme*, Diplomarbeit, Johannes Gutenberg-Universität Mainz, **2014**.

[41] L. Pohlmann, *Elektrochemische Messmethoden: Mikroelektroden*, Freie Universität Berlin, **2008**.

[42] D. A. Steingart, Princeton Lab for Electrochemical Engineering Systems Research, `https://steingart.princeton.edu/ardustat`, abgerufen am 01.08.2018.

[43] D. A. Steingart, Ardustat Softwarerepository bei Github `https://github.com/dansteingart/Ardustat`, abgerufen am 01.08.2018.

[44] J. R. Burns, C. Ramshaw, Development of a Microreactor for Chemical Production, *Trans. Inst. Chem. Eng.*, **1998**, *77*, 206.

[45] E. Kolehmainen, I. Turunen, Micro-Scale Liquid-Liquid Separation in a Plate-Type Coalescer, *Chemical Engineering and Processing*, **2007**, *46*, 834–839.

[46] J. G. Kralj, H. R. Sahoo, K. F. Jensen, Integrated continuous microfluidic liquid–liquid extraction, *Lab Chip*, **2007**, *7*, 256–263.

[47] A. Adamo, P. L. Heider, N. Weeranoppanant, K. F. Jensen, Membrane-Based, Liquid-Liquid Separator with Integrated Pressure Control, *Ind. Eng. Chem. Res.*, **2013**, *52*, 10802–10808.

[48] L. „Ladyada“ Fried, Adafruit Industries, `https://www.adafruit.com`, abgerufen am 01.08.2018.

[49] L. „Ladyada“ Fried, Adafruit MotorShield for Arduino, `https://learn.adafruit.com/adafruit-motor-shield/overview`, abgerufen am 01.08.2018.

[50] Bedienungsanleitung, *Fraction Collector FRAC-100 and FRAC-200*, Amersham Biosciences AB, **2002**.

[51] D. Karl, *Physikochemische Optimierung von Cycloadditionen und Olefinierungen an bifunktionellen Aldehyden in Mikroreaktoren*, Dissertation, Johannes Gutenberg-Universität Mainz, **2017**.

[52] J. Bager, Die KI-Revolution - Vom Siegeszug der lernenden Software, *c't*, **2016**, *6*, 124 – 129.

[53] A. Trinkwalder, Die Mathematik der neuronalen Netze: einfache Mechanismen, komplexe Konstruktion, *c't*, **2016**, *6*, 130 – 135.

[54] F. S. Foundation, GNU General Public License v. 3.0, `https://www.gnu.org/licenses/gpl-3.0`, abgerufen am 01.08.2018.

[55] C. Commons, Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 3.0 Deutschland, `https://creativecommons.org/licenses/by-nc-sa/3.0/de/`, abgerufen am 01.08.2018.