Hubert Tockner, BSc

Entwicklung eines automatisierten Perfusionssystems für Live Cell Imaging Anwendungen

Masterarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Biomedical Engineering

eingereicht an der

Technischen Universität Graz

Betreuer

Univ.-Prof. Dipl.-Ing. Dr.techn. Christian Baumgartner

Institut für Health Care Engineering mit Europaprüfstelle für Medizinprodukte

Graz, April 2018

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

Datum

Unterschrift

Entwicklung eines automatisierten Perfusionssystems für Live Cell Imaging Anwendungen

Zur Untersuchung dynamischer Vorgänge innerhalb lebender Zellen werden unterschiedliche Methoden, wie beispielsweise Phasenkontrastmikroskopie oder Fluoreszenzmikroskopie eingesetzt, wobei zentrale Punkte das Aufrechterhalten der Umgebungsparameter und das Applizieren von Substanzen mittels eines Perfusionssystems sind. Ziel dieser Arbeit ist ein Konzept eines bedienerfreundlichen, automatisierten Perfusionssystems für Live Cell Imaging Anwendungen zu erstellen. Das Ergebnis ist ein prototypischer Aufbau, wobei anhand beispielhafter Anwendungen mit Fluoreszenzmikroskopischen Untersuchungen an lebenden Zellen das Gesamtsystem evaluiert wird.

Schlüsselwörter: Live Cell Imaging, Fluoreszenzmikroskopie, Perfusionssystem, Konzeptentwicklung

Development of an automated Perfusion System for Live Cell Imaging Applications

Various methods, such as phase-contrast Microscopy or fluorescence Microscopy, are used to study dynamic processes that take place within living cells, where the maintenance of enviromental parameters and the application of substances are critical tasks, that can be completed with perfusion systems. The aim of this work is the creation of a concept for a user-friendly, automated perfusion system for live cell Imaging applications. The result is a prototype, where the whole system is evaluated on the basis of exemplary applications, analyzing living cells using fluorescence Microscopy technices.

Key words: live cell imaging, fluorescence microscopy, perfusion system, concept design

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Die Motivation für diese Arbeit ist die Erstellung eines einheitlichen Konzeptes für eine einfache und genaue Perfusion von Zellen in einem breit gefächerten Anwendungsspektrum. Derzeit verfügbare Systeme können großteils nicht automatisiert angesteuert werden, außerdem besteht keine Möglichkeit Schaltvorgänge zu protokollieren, und für wiederholte Durchführungen von Messvorgängen Vorlagen zu erstellen und zu exportieren.

Durch die Entwicklung eines automatisierten Perfusionssystems sollte es möglich sein, diese Lücke zu schließen, um somit vor allem eine einfache Nachvollziehbarkeit und Wiederholbarkeit der durchgeführten Versuche gewährleisten zu können. Hauptsächlich ergeben sich dadurch Vorteile im Anwendungsbereich von Live Cell Imaging Anwendungen, bei denen der Einsatz von Systemen, welche Perfusion ermöglichen, eine wichtige Rolle spielt.

Das System sollte mit Hinblick auf eine mögliche zukünftige Integration in ein Medizinprodukt oder als Zubehör für ein System zur In-Vitro Diagnostik entwickelt werden, somit sollte sowohl die Hardware- und Software Entwicklung, als auch die Untersuchung der Gebrauchstauglichkeit, in Anlehnung an wichtige Punkte der entsprechenden Normen erfolgen.

1.1. Live Cell Imaging

Neben dem strukturellen Aufbau von Zellen, welche beispielsweise mithilfe konventioneller Mikroskopie-Techniken untersucht werden, können weitere wichtige Erkenntnisse durch Beobachten dynamischer Prozesse innerhalb lebender Zelle erhalten werden. Diese dynamischen Prozesse können beispielsweise Zellmigration, morphologische und biochemische Veränderungen von Zellen oder Änderungen der intrazellulären Ionenzusammensetzung sein. Unter Live-Cell Imaging versteht man im Allgemeinen alle Techniken, welche zur Untersuchung solcher dynamischer Vorgänge in lebenden Präparaten eingesetzt werden können [1].

Mithilfe dieser Techniken können Zellfunktionen quasi in Echtzeit unter Normal- und Experimentalbedigungen beobachtet werden. Ein weiterer Vorteil der sich durch die Beobachtung lebender Zellen ergibt, ist das Vorliegen der Zellstrukturen in natürlicher Umgebung. Außerdem ist es möglich, Interaktionen zwischen Zellen und Zellverbänden sichtbar zu machen.

Während des gesamten Untersuchungszeitraumes müssen die untersuchten Zellen am Leben gehalten werden, wobei dieser Punkt eine große Herausforderung bei Live Cell Imaging Anwendungen darstellt. Es ist erforderlich dass die Umgebung der Zellen innerhalb des physiologischen Bereichs liegt, vor allem in Hinblick auf Temperatur, pH-Wert und Osmolarität [2]. Um Zellreaktionen zu untersuchen, werden unterschiedliche Substanzen zeitabhängig den Zellen zugeführt.

Es existieren unterschiedliche Möglichkeiten um Live Cell Imaging Untersuchungen durchführen zu können, zum Beispiel Phasenkontrastmikroskopie oder Fluoreszenzmikroskopie. In dieser Arbeit werden großteils die Möglichkeiten von Live

1. Einleitung

Cell Imaging Untersuchungen in Zusammenhang mit der Fluoreszenzmikroskopie in Kombination mit Perfusionssystemen beschrieben. Untersuchungen lebender Zellen mithilfe der Fluoreszenzmikroskopie ermöglichen es, mit fluoreszierenden Proteinen Veränderungen der Zelle qualitativ und quantitativ darzustellen. Ein zu beachtender Punkt beim Einsatz fluoreszierenden Proteine ist das Ausbleichen der zu untersuchenden Proben, wobei dieser Effekt auch als Photobleaching bezeichnet wird, und in eine zeitabhängige Abnahme des Fluoreszenzsignals resultiert [3].

Bei der Untersuchung lebender Zellen mithilfe der Fluoreszenzmikroskopie werden Fluorophore eingesetzt, welche optimalerweise in der Nähe ihres maximalen Anregungsspektrums mit einer Lichtquelle passender Wellenlänge angeregt werden. Anschließend wird das vom Fluorophor emittierte Licht, welches eine höhere Wellenlänge als das Anregungslicht aufweist, mithilfe lichtempfindlicher Detektoren aufgezeichnet. Die aufgenommene Intensität des emittierten Lichts ermöglicht es, unterschiedliche Vorgänge und Veränderungen in der Zelle zu erkennen [4].

Die Komponenten, welche im Verlauf einer Live Cell Imaging Untersuchung den größten Einfluss auf den Verlauf der Messung haben, werden untenstehend kurz beschrieben. In Abbildung 1.1 ist der optische Strahlengang eines Fluoreszenzmikroskopes, welches zur Vermessung von Cyan und Gelb fluoreszierenden Sensoren (CFP-YFP), eingesetzt werden kann, dargestellt. Die simultane Detektion zweier Wellenlängen mithilfe zweier Kameras ermöglicht es Förster-Resonanzenergietransfer (FRET) basierende Sensoren zu vermessen. Die zu untersuchenden Zellen werden auf einem Träger aufgebracht, wobei sich das Objektiv direkt unter diesem Träger befindet. Zellen, welche den fluoreszierenden Sensor beinhalten, werden durch das Objektiv mittels einer Lichtquelle geeigneter Wellenlänge angeregt, wobei durch einen definierten Dichroic-Filter, in Abbildung 1.1 mit D1 bezeichnet, die Trennung von Anregungslicht und Emissionslicht erfolgt. Die Intensität des Emissionslicht wird

anschließend in definierten Zeitabständen detektiert und ausgewertet. Die anschließende Auswertung des aufgenommenen Signals erfolgt über eine spezielle Software, wobei unterschiedliche Algorithmen zur Bleaching-Korrektur oder zur Erhöhung des Kontrasts zum Einsatz kommen.

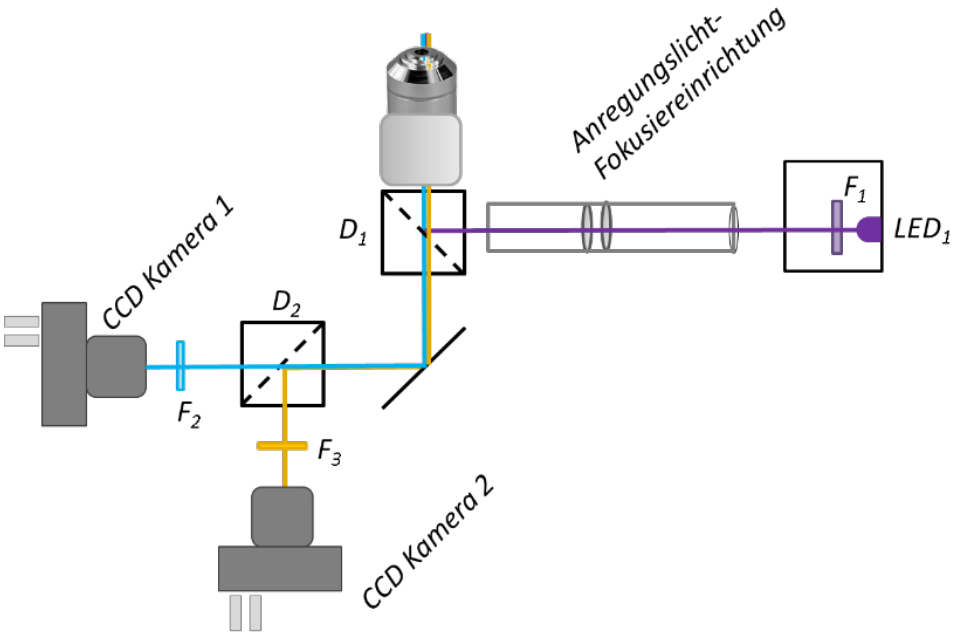


Abbildung 1.1.: Aufbau eines Fluoreszenzmikroskopes zur Verwendung mit CFP-YFP Sensoren

1.2. Perfusion

Der Begriff Perfusion beschreibt dass Passagieren von Flüssigkeiten durch oder über Zellen und Zellverbänden [1]. Systeme, welche Perfusion ermöglichen, werden im Zuge von Live Cell Imaging Untersuchungen für das kontrollierte Passagieren von Flüssigkeiten eingesetzt. Außerdem werden mithilfe von Perfusionssystemen Substanzen appliziert, welche das Überleben der Zellen während des Untersuchungszeitraumes ermöglichen.

1.2.1. Möglichkeiten der Perfusion

Es existieren unterschiedliche Möglichkeiten, um einen Fluß einer Substanz in einem Medium zu gewährleisten, wobei jede Variante Vorteile und Nachteile aufweist. In Hinblick auf Live Cell Imaging Anwendungen ist vor allem ein kontinuierliches Strömungsprofil und ein schnelles Umschalten zwischen den Kanälen, welche unterschiedliche Substanzen beinhalten, notwendig. Untenstehend folgt eine kurze Auflistung der gängigsten Methoden zur Erzeugung und Regulierung eines Strömungsprofils und der Gegenüberstellung der jeweiligen Vor- und Nachteile [5]. Grob lassen sich diese Möglichkeiten in energetisch betriebene und nicht energetisch betriebene Arten der Perfusion einteilen.

Nicht energetisch betriebene Perfusion - Schwerkraftperfusion

Zur nicht energetisch betriebenen Perfusion zählt die Schwerkraftsperfusion. Treibende Kraft bei der Schwerkraftperfusion ist der hydrostatische Druck, welcher sich durch den Einfluss der Gravitation auf das Fluid einstellt. Die Flussgeschwindigkeit wird bei dieser Methode über die Höhendifferenz zwischen dem Flüssigkeitspegeln in den Aufbewahrungsbehältnissen und der Höhe des Objektträgers am Mikroskops bestimmt.

Vorteile:

- kostengünstig
- einfach zu bedienen
- laminarer, gleichmäßiger Fluss
- Einsatz unterschiedlicher Flüssigkeiten möglich
- rasches Umschalten der gewünschten Substanzen möglich

Nachteile:

- Einschränkungen in der Festlegung der Flussrate
- Einsatzdauer begrenzt durch Volumen der Behältnisse
- stark schwankende Genauigkeit der Flussrate

Energetisch betriebene Perfusion

Spritzenpumpen

Bei Spritzenpumpen wird der fixierte Spritzenkolben durch einen Schrittmotor angetrieben, wobei beim Einsatz eines linearen Präzisionsantrieb eine sehr genaue Einstellung der Flussrate erfolgen kann. Einige Modelle weisen außerdem bereits Schnittstellen, beispielsweise zur synchronen Ansteuerung oder zur Überwachung der Pumpe, auf.

Vorteile

- sehr hohe Dosiergenauigkeit
- kontinuierliche Förderung

Nachteile

- teuer im Einsatz
- definierte Eigenschaften der eingesetzten Spritzen erforderlich
- einfaches Umschalten der gewünschten Flüssigkeiten während einer Messung ist nur sehr aufwendig realisierbar

Rollenpumpen

Rollenpumpen zeichnen sich dadurch aus, dass rotierende Rollen, welche in einem definierten Abstand zueinander fixiert sind, nacheinander auf einen Abschnitt des flexiblen Schlauchs drücken. Durch die abschnittsweise Verformung des Schlauchs wird bei jeder Umdrehung des Pumpenkopfs ein definiertes Volumen vorangetrieben.

Vorteile

- hohe Dosiergenauigkeit
- keine Ventile oder andere schaltenden Bauteile erforderlich
- Anpassung der Flussrate während der Messung möglich

Nachteile

- möglicher Abrieb des Schlauchs durch die mechanische Belastung
- kein durchgehend kontinuierlicher Fluss
- kein einfaches Umschalten der gewünschten Flüssigkeiten während einer Messung möglich

Piezopumpen

Einen weiteren Pumpentyp stellt die Piezopumpe dar. Die Wirkungsweise einer Piezopumpe basiert auf den inversen piezoelektrischen Effekt, wobei sich definierte Materialien beim Anlegen einer elektrischen Spannung elastisch verformen. Durch diese Verformung kann über unterschiedliche Druckverhältnisse ein Flüssigkeitstransport erfolgen.

Vorteile

- selbstansaugende Ausführung möglich
- einfache Einstellung der Flussrate über eine definierte elektrische Spannung möglich
- Pumpen mit eingebauter Regelfunktion erhältlich
- kompakte Abmessungen

Nachteile

- nur geringe Flussraten möglich
- Material in dauernden Kontakt mit der eingesetzten Substanz

1.3. Gebrauchstauglichkeit

Im Zuge dieser Arbeit sollte eine Untersuchung der Gebrauchstauglichkeit durchgeführt werden, und die Entwicklung anhand eines Gebrauchstauglichkeitsorientierten Prozesses erfolgen. Der Begriff Gebrauchstauglichkeit beschreibt die Eignung eines Produktes, seinen bestimmungsgemäßen Verwendungszweck zu erfüllen, wobei die Eigenschaften der Gebrauchstauglichkeit in objektiv und nicht objektiv ermittelbaren Größen eingeteilt werden können. Objektiv feststellbare Merkmale beschreiben hierbei die technisch-funktionelle Eigenschaften des Produktes und werden unter dem Begriff Funktionalität zusammengefasst. Unter den nicht Objektiv feststellbaren Größen der Gebrauchstauglichkeit werden alle Eigenschaften zusammengefasst, welche nicht nur vom Produkt, sondern auch vom Anwender und der Art der Anwendung abhängig sind [6].

1.3.1. Definitionen und Aspekte der Gebrauchstauglichkeit

Eine formale Definition der Gebrauchstauglichkeit erfolgt in der Norm EN 62366 [7]

> "Eigenschaft des USER INTERFACE, die den Gebrauch unterstützt und damit EFFEKTIVITÄT, EFFIZIENZ sowie Zufriedenheit des USERS in der festgelegten NUTZUNGSUMGEBUNG erzielt" [7]

Durch eine Ergänzung der Definition wird ein Zusammenhang der Gebrauchstauglichkeit und der Beeinflussung der Sicherheit hergestellt:

> "Alle Aspekte der USABILITY wie EFFEKTIVITÄT, EFFIZIENZ und ZUFRIEDENHEIT, können SICHERHEIT erhöhen oder verringern." [7]

Die Begriffe Effektivität, Effizienz und Zufriedenheit werden in diesem Zusammenhang wie folgt definiert:

Effektivität

Die Effektivität eines Systems kann auch als Wirksamkeit beschrieben werden, welche im Zusammenhang der Gebrauchstauglichkeit den Umfang beschreibt, in welchem ein Benutzer die Funktionalität eines Systems ausnutzt, um eine definierte Aufgabe zu erfüllen. Effektivität ist in der Norm EN-62366 wie folgt definiert:

> "Genauigkeit und Vollständigkeit, mit denen USER festgelegte Ziele erreichen." [7]

Effizienz

Effizienz kann grundsätzlich als Kenngröße der Wirtschaftlichkeit eines Systems verstanden werden. Sie beschreibt den Ausmaß des Aufwandes, welcher für den Einsatz des Gerätes erforderlich ist. Im Vergleich zur Effektivität ist die Effizienz abhängig vom eingesetzten Aufwand, wobei bei der Effektivität beschrieben wird, in welchem Umfang das resultierende Ergebnis vom erwarteten Ergebnis abweicht.

> "aufgewendete Ressourcen in Relation zur EFFEKTIVITÄT." [7]

Zufriedenheit

Durch den Aspekt der Benutzerzufriedenheit kann festgestellt werden, wie gut das Produkt an die Benutzermerkmale und Anforderungen des Benutzers angepasst ist [6].

2. Aufgabenstellung

Ziel dieser Arbeit ist die Entwicklung eines semiautomatischen Perfusionssystems, wobei die einzelnen Komponenten für Live Cell Imaging Anwendungen im Zuge von Messungen mittels Fluoreszenzmikroskopie optimiert werden sollten.

Es erfolgt eine Gliederung der Gesamtentwicklung in eine Entwicklung der Hardware Komponenten und in eine Entwicklung der Software Komponenten. Im Zuge der Entwicklung der Hardware Komponenten sollten die mechanischen Komponenten modelliert und prototypisch gefertigt, und eine elektronische Schaltung erstellt und aufgebaut werden. Anschließend sollte bei der Entwicklung der Software Komponenten ein Konzept zur Ansteuerung der elektronischen Komponenten erstellt, und eine Anwendungssoftware programmiert werden, welche eine benutzerfreundliche Ansteuerung des Gesamtsystems ermöglicht. Während der gesamten Entwicklungsdauer sollte ein gebrauchstauglichkeitsorientierter Prozess verfolgt werden, wobei mithilfe laufender Interviews mit den zukünftigen Anwendern des Systems bereits im Zuge der Entwicklungsphasen Optimierungen durchgeführt werden sollten.

Folgende Anforderungen an das System wurden identifiziert:

- Entwicklung eines Gehäuses für die Aufbewahrung 9 unterschiedlicher Substanzen
- Möglichkeit zur indirekten Einstellung der Fließgeschwindigkeit durch Höhenverstellbarkeit der Pufferbehälter
- Dimensionierung einer elektronischen Schaltung, welche es ermöglicht durch elektromechanische Komponenten den Fluss unterschiedlicher Flüssigkeiten zu steuern
- Definieren der Schnittstelle zwischen Hardware und Software Komponenten, und beschreiben der zum Einsatz kommenden Protokolle
- Entwicklung einer Anwendungssoftware mit zugehöriger grafischer Benutzeroberfläche zur benutzerfreundlichen Ansteuerung des Gesamtsystems
- Ermöglichen einer einfachen Automatisierung von Schaltvorgängen zur Versuchsplanung
- Datenbankbasierte Dokumentation der verwendeten Substanzen und Protokolle
- Fertigung eines prototypischen Aufbaus des entwickelten Systems
- Durchführen eines gebrauchstauglichkeitsorientierten Entwicklungsprozess
- Evaluierung des Gesamtsystems anhand konkreter Anwendungsbeispiele

3. Entwicklung und Evaluierung der Hardware Komponenten

Der erste Punkt zur Realisierung des Perfusionssystems stellt die Entwicklung der benötigten Hardware Komponenten dar.
Basierend auf den Vorgaben, ein höhenverstellbares Gehäuse für die verwendeten Behältnisse und Elektronikkomponenten zu erstellen, wurde ein Gehäuse konzipiert, welches möglichst kompakte Abmessungen aufweist. Um den gezielten Fluss der einzelnen Pufferlösungen zu ermöglichen wurden passende Schlauchklemmventile ausgewählt, und deren Charakteristika und Eigenschaften dargestellt. Darauf basierend erfolgte die Entwicklung einer elektronischen Schaltung, welche im Zusammenspiel mit einem Mikrocontroller über eine grafische Benutzeroberfläche die Ansteuerung des Perfusionssystems ermöglicht.

Abbildung 3.1 zeigt den prinzipiellen Aufbau der Hardware Komponenten und der eingesetzten Schnittstellen. Außerdem sind Überschneidungen zwischen der Entwicklung der elektronischen Komponenten und der mechanischen Komponenten sichtbar, wobei die einzelnen mechanischen Komponenten mit einem blauen Rechteck und die einzelnen elektronischen Komponenten mit einem grünen Rechteck hinterlegt sind. Basierend auf der in der Einleitung beschriebener Gegenüberstellung möglicher

Arten der Perfusion wurde entschieden ein gravitationsbasiertes Perfusionssystem zu entwickeln.

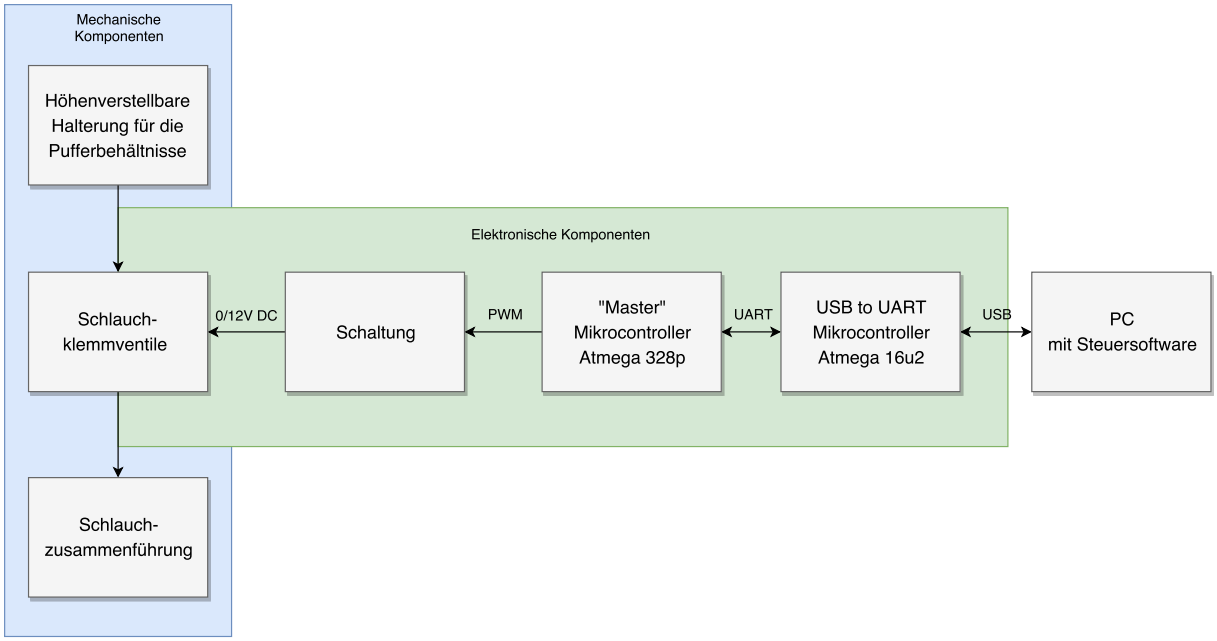


Abbildung 3.1.: Übersicht über die verwendeten Hardware-Komponenten und Schnittstellen

3.1. Konstruktion der mechanischen Komponenten

Der mechanische Aufbau beinhaltet eine höhenverstellbare Halterung der Spritzen, in denen die unterschiedlichen Pufferlösungen eingefüllt werden können, ein Gehäuse für die elektronischen Komponenten, welches ebenfalls höhenverstellbar ausgeführt ist, und eine Schlauchzusammenführung, in welcher die Schläuche der neuen Behältnisse zu einem resultierenden Schlauch zusammengeführt werden. Die Modellierung der mechanischen Komponenten wurde mithilfe der CAD Software Inventor 2017 (Autodesk Corp.,USA) und der CAD Software Catia v5R21 (Dassault Systèmes SE, Frankreich) durchgeführt.

3.1.1. Spritzenhalterung

Der erste Punkt des mechanischen Aufbaus stellt die Halterung der Behältnisse für die einzelnen Pufferlösungen dar. In einer ersten Iteration des Gehäuses wurden drei Reihen mit jeweils neun Spritzen verwendet, welche über kommunizierende Gefäße zusammengeführt wurden. Es ergab sich dadurch ein sehr hohes Gewicht des Aufbaus, und eine Vergrößerung der Außenabmessungen. Aufgrund der Anforderung ein kompaktes System zu erstellen, wurde ein Behältnis pro Kanal gewählt und somit eine Reduktion auf insgesamt neun Behältnisse durchgeführt. Zur Aufbewahrung der Substanzen dienen handelsübliche Laborspritzen mit einem Füllvermögen von 50ml.

In Abbildung 3.2 ist ein Modell der Spritzenhalterung abgebildet. Die obere Platte mit den neun Öffnungen dient dabei der Aufbewahrung der Behältnisse. Die Platte auf der Unterseite wird zum Verbinden der Halterung mit dem Gehäuse für die elektronischen Komponenten verwendet. Außerdem ist an der Vorderseite der Halterung ein Haken

für die Aufnahme weiterer Komponenten, welche zukünftig das System erweitern sollten, vorgesehen.

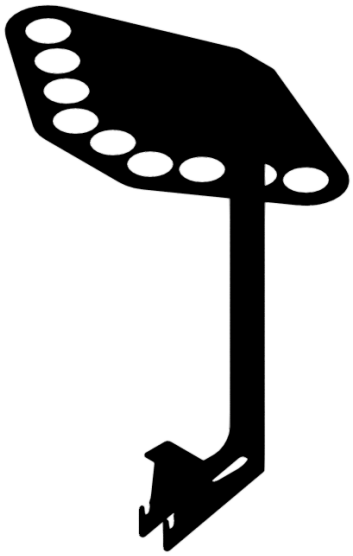


Abbildung 3.2.: Halterung für die Aufnahme von 9 Perfusionsbehältnisse

Die Spritzenhalterung kann auf der Rückseite mit einem Aluminium Extrusionsprofil, welches eine Seitenlänge von 30mm aufweist, verbunden werden. Mittels eines Hebels ist es möglich, die Spritzenhalterung in der Höhe zu verstellen und somit die Fließgeschwindigkeit der Substanz zu verändern. Abbildung 3.3 zeigt die mit dem Standfuß verbundene Spritzenhalterung und den montierten Hebel, welcher zur Höhenverstellbarkeit des Systems genutzt werden kann. Um eine hohe Stabilität gewährleisten zu können, wurden zwei Profile mittels Winkelelemente verbunden. Durch die kompakten Abmessungen des Standfußes ist es möglich, das System auf einen Labortisch neben einem Mikroskop zu platzieren. Außerdem wird durch den stabilen Stand des Systems das Risiko des möglichen Umkippens und des damit verbundenen Flüssigkeitsaustritts minimiert.

Abbildung 3.3.: Spritzenhalterung verbunden mit Extrusionsprofilen zur Gewährleistung der Höhenverstellbarkeit

3.1.2. Gehäuse für die elektronischen Komponenten

Für die Aufbewahrung der Klemmventile und der weiteren elektronischen Komponenten erfolgte die Konstruktion eines Gehäuses, welches für einen leichten Transport mit geringen Aufwand vom System getrennt und wieder hinzugefügt werden kann. Nachdem bei den ersten Iterationen das Gehäuses aus Acrylglas gefertigt wurde und sich dadurch Herausforderungen im Zusammenbau und in der Genauigkeit des Aufbaus ergaben, wurde bei der finalen Version des Gehäuses eine additive Fertigungsmöglichkeit mittels eines 3D-Druckers gewählt. Ein großer Vorteil dieses Fertigungsverfahren stellt

die Tatsache dar, dass Änderungen am Gehäuse rascher realisiert werden können und eine mögliche Fertigung von Kleinserien direkt mittels additiven Fertigungsverfahren erfolgen kann [8]. Ein computergeneriertes Modell, in welchem die Vorderseite des Gehäuses und Öffnungen für die Klemmventile und Leuchtdioden eingezeichnet sind, zeigt Abbildung 3.4.

Da das Gesamtsystem für einen Einsatz in einer Laborumgebung vorgesehen ist, fiel die Entscheidung auf ein geschlossenes und abgedichtetes Gehäuse. Diese Entscheidung wurde vor allem wegen dem erhöhten Berührungs-, Fremdkörper- und Wasserschutz im Vergleich zu einem offenen Gehäuse getroffen. Es dürfen hier allerdings nur geringe Verlustleistungsdichten auftreten, da im Inneren des Gehäuses lediglich die natürliche Konvektion zur Wärmeübertragung beziehungsweise Wärmeabführung genutzt wird [9].

Abbildung 3.4.: CAD-Modell des Gehäuses für die Aufnahme der elektronischen Komponenten

An der Vorderseite des Gehäuses können neun Klemmventile befestigt werden, welche zur Steuerung des Pufferflusses eingesetzt werden, wobei sich unter jedem Klemmventil eine Öffnung für die Montage der Leuchtdiode zur Statusanzeige des jeweiligen

Kanals befindet. Die Montage der Klemmventile erfolgt über zwei Schraubverbindungen, die Leuchtdioden zur Anzeige des aktuellen Zustands werden über eine für das Leuchtdiodengehäuse passende Mutter befestigt. Um eine einfache Montage der Klemmventile und Verkabelung der Komponenten gewährleisten zu können, wurde die Befestigung der Platine am hinteren Deckel des Gehäuses durchgeführt. Eine Zugentlastung und die Gewährleistung einer geordneten Verkabelung erfolgt durch die Befestigung der Verbindungen an vorgegebenen Stellen am Boden des Gehäuses. An der Unterseite der Gehäuseaußenseite ist ein Steg vorgesehen, an welchem das Gehäuse mit dem restlichen Komponenten verbunden werden kann.

Abbildung 3.5 zeigt den modellierten Deckel des Gehäuses. Um eine bessere Stabilität des Gehäuses zu erhalten, wurden Verstrebungen an der Innenseite des Deckels angebracht. Zwei Öffnungen am Deckeln dienen der Montage der Buchsen für die USB-Verbindung und die Spannungsversorgung. Die gefertigte Platine kann über vier Schraubverbindungen am Deckel befestigt werden.

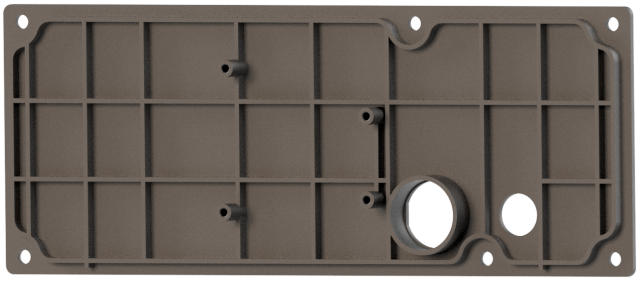


Abbildung 3.5.: Modell des Gehäusedeckels

Die prototypische Fertigung des Gehäuses erfolgte mithilfe eines 3D-Druckers (Flashforge Dreamer). Als Material für das Gehäuse und den dazugehörigen Deckel wurde

Filament eingesetzt, welches der Sorte der Polyactide (PLA) zugeordnet ist, den Vorteil der einfachen Verfügbarkeit hat und gute Stabilitätseigenschaften aufweist.

3.1.3. Schlauchzusammenführung

Um einen resultierenden Kanal zu erhalten, muss der Fluss der neun Kanäle zu einer gemeinsamen Öffnung zusammengefasst werden, an welchem ein zum Mikroskoptisch führender Schlauch angebracht wird. Dies erfolgt durch einen konstruierten Flüssigkeitsaustauscher, welcher neun Einlässe an der Oberseite und einen resultierenden Auslass an der Unterseite aufweist. Innerhalb des Austauschers erfolgt der Pufferfluss durch neun konische Bohrungen, welche in den gemeinsamen Kanal münden. Das gemeinsame Volumen sollte dabei so gering wie möglich gehalten werden, um ein Vermischen der Flüssigkeiten beim Umschalten des aktuellen Kanals entgegenzuwirken. Eine Herausforderung bei der Modellierung der Schlauchzusammenführung stellte die Konstruktion der Anschlussstellen der Schläuche und der korrekten Abdichtung dieser Stellen dar. Ein Modell der Zusammenführung ist in Abbildung 3.6 dargestellt. Zusätzlich zu den konisch verlaufenden Anschlussstellen wurden handelsübliche Dichtungsringe an den Anschlüssen der Schläuche angebracht.

3.2. Entwicklung der elektronischen Komponenten

Nach der Modellierung der mechanischen Komponenten erfolgte die Entwicklung und Evaluierung der elektronischen Komponenten. Zunächst wurden die verwendeten Ventile in Hinblick auf die wichtigsten Eigenschaften charakterisiert. Basierend auf diesen Ergebnissen erfolgte die Entwicklung einer elektronischen Schaltung. Im

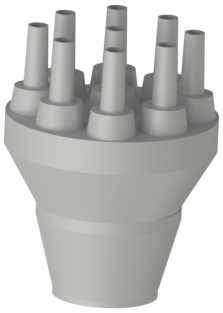


Abbildung 3.6.: Modell der Schlauchzusammenführung

Anschluss darauf konnte eine Platine designet und gefertigt, und die Funktionalität der elektronischen Komponenten getestet werden.

3.2.1. Evaluierung der eingesetzten Ventile

Für den kontrollierten Fluss der einzelnen Substanzen kommen elektrisch ansteuerbare Schlauchklemmventile zum Einsatz. Der große Vorteil dieser Ventile ist eine Trennung der Ventilkomponenten vom transportierten Medium, lediglich die Innenfläche des Schlauchs tritt in Kontakt mit der zu fördernden Substanz. Somit ist keine Reinigung des Ventils erforderlich und es kann keine Vermischung unterschiedlicher Substanzen im Abschnitt vor der Schlauchzusammenführung erfolgen. Die verwendeten Schlauchklemmventilen wurden vor dem Einsatz hinsichtlich ihrer elektronischen und mechanischen Eigenschaften charakterisiert. Folgende Daten wurden vom Hersteller zur Verfügung gestellt [10].

Basierend auf diesen Daten wurden unterschiedliche Ansteuerungsmöglichkeiten erprobt und verglichen. Wichtiger Aspekt ist hierbei die auftretende Erwärmung des

Tabelle 3.1.: Hersteller Spezifikationen der verwendeten Klemmventile

Typ	2/2 Wege, normal geschlossen
Schlauchinnendurchmesser	0.8 - 2mm
Schlauchaußendurchmesser	2.4 - 4mm
Betriebsspannung	12V DC
Leistungsaufnahme Betriebsart	3.0W 100% Einsatzdauer
Umgebungstemperatur	0-40 Grad Celsius

Ventils und der damit verbundenen notwendigen Abfuhr der Verlustleistung.

Funktionsweise der Ventile und mögliche Ansteuerungsarten

Die eingesetzten Ventile besitzen zwei unterschiedliche Kanäle, wobei ein Kanal stromlos geschlossen und der zweite Kanal stromlos geöffnet ist. Innerhalb des Ventils ist eine Spule angebracht, welche beim Anlegen der Betriebsspannung ein magnetisches Feld induziert. Durch dieses magnetische Feld wird ein Kolben bewegt, welcher an einem Plastikstück befestigt ist und durch Gegendruck den Schlauch, welcher sich im Stromlos geöffneten Kanal befindet, abklemmt und somit den Kanal schließt, und den Schlauch im stromlos geschlossenen Kanal freigibt, somit wird der Durchfluss in diesem Kanal ermöglicht. Nach Abschalten der Betriebspannung bewegt sich durch eine Feder im Inneren des Gehäuses der Kolben wieder in die Ausgangsstellung zurück und der Ausgangszustand der beiden Kanäle wird wiederhergestellt.

Der Einschaltvorgang des Ventils erfordert eine hohe Stromstärke, da ein größerer Abstand des Kolbens zum Endzustand herrscht. Die benötigte Kraft, welche vom

Magnetfeld zum Anziehen des Kolbens überwunden werden muss, kann mittels Formel 3.1 näherungsweise bestimmt werden. In dieser Formel beschreibt N die Anzahl der Windungen, μ_0 die magnetische Permeabilität, l die Länge des Spaltes, welcher überwunden werden muss, A die Querschnittsfläche des Kolbens und i den durch die Spule fließenden Strom [11].

$$F = \frac{N^2 \mu_0 A}{2l^2} i^2 \tag{3.1}$$

Sobald sich der Kolben im Endzustand befindet und das Ventil eingeschaltet ist, verringert sich der Abstand l und es wird weniger Kraft zum Halten des Kolbens in der aktuellen Position benötigt. Somit verringert sich auch die benötigte Stromstärke bei einer Gleichspannungsansteuerung. Bei Ansteuerung des Ventils mit einer Wechselspannung erhöht sich die Induktanz der Spule durch das Annähern des Kolbens an die Spule. Durch die erhöhte Impedanz der Spule verringert sich der Spulenstrom und somit auch die resultierende Leistung, welche zum Aufrechterhalten des Magnetfeldes benötigt wird.

Bei einer Ansteuerung mit Gleichspannung steigt zunächst der Strom durch die Spule asymptotisch an, bis dieser gleich der Betriebsspannung geteilt durch den ohmschen Spulenwiderstand ist. Somit resultieren bei dieser Ansteuerungsart höhere Verluste als bei der Ansteuerung mit Wechselspannung. Da jedoch bei der Wechselspannungsansteuerung üblicherweise höhere Versorgungsspannungen wie bei der Gleichspannungsansteuerung verwendet werden und somit weitere Herausforderungen bezüglich der sicheren und einfachen Ansteuerung über programmierbare Steuerungen entstehen, wird meist eine Ansteuerung mittels Gleichspannung gewählt.

Um die bei der Gleichspannungsansteuerung auftretenden Verluste und in weiterer

Folge auch auftretende Erwärmung der Spule und des Ventilgehäuses minimieren zu können, existierten unterschiedliche Methoden, welche in weiterer Folge kurz beschrieben und verglichen werden.

Hit-and-Hold-Schaltung
Mithilfe sogenannter Hit-and-Hold Schaltungen wird beim initialen Einschalten des Ventils die gesamte Versorgungsspannung an die Kontakte angelegt. Nach einer definierten Zeitkonstante, bei der gewährleistet ist das der Kolben des Ventils angezogen ist, wird schaltungstechnisch die Versorgungsspannung auf einen Bruchteil der initialen Versorgungsspannung reduziert. Durch diese Reduktion verringert sich laut dem Ohmschen Gesetz der durch die Spule fließende Strom, welcher durch den ohmschen Widerstand der Spule bei Gleichspannungsansteuerung bestimmt wird. Ein Vorteil dieser Ansteuerungsmöglichkeit ist, dass es möglich ist, eine Hit-and-Hold-Schaltung zu dimensionieren, bei der keine hochfrequenzen Signalformen auftreten.

Pulsweitenmodulation Bei einer pulsweitenmodulierten Ansteuerung ergibt sich der Vorteil, dass die elektronische Schaltung nicht beziehungsweise nur geringfügig im Vergleich zur einfachen Gleichspannungsansteuerung verändert werden muss. Somit ist eine Ansteuerung, beispielsweise mittels eines Mikrocontrollers, möglich.

Ein pulsweitenmoduliertes Signal zeichnet sich dadurch aus, dass es nicht einen konstanten Spannungspegel aufweist, sondern für die Dauer einer Periode einen High-Pegel und für die restliche Dauer der Periode einen Low-Pegel aufweist. Der Betrag des Tastgrades (Duty Cycle) beschreibt hierbei die Zeit der Periode, in der ein High-Pegel ausgegeben wird, in Prozent [12].

Aufgrund induktiver Eigenschaften der Spule im Ventil verschwindet der Spulenstrom nach Abschalten der Versorgungsspannung nicht sofort und steigt auch nicht sofort

nach Anlegen einer Versorungsspannung auf einen konstanten Wert. Somit weist der Spulenstrom eine gewisse Trägheit auf. Bei der pulsweitenmodulierten Ansteuerung des Ventils wird diese Trägheit ausgenutzt und es ist möglich bei genügend hohen Frequenzen durch Variieren des Tastgrades einen annähernd konstanten Spulenstrom festzulegen. Ein Vorteil dieser Methode ist die einfache Realisierung, beispielsweise durch PWM Module oder digitale Timer. Außerdem besteht keine Notwendigkeit ein diskretes Signal in einen analogen Wert umzuwandeln, somit können mit einem geringen Schaltungsaufwand durch zeitabhängiges Schalten digitaler Signale Spannungen ausgegeben werden, welche im Mittel einem analogen Spannungswert aufweisen.

Integrierte Schaltkreise Eine weitere Möglichkeit zur effizienten Ansteuerung von Ventilen ist das Verwenden integrierter Schaltkreise. Es existieren Lösungen unterschiedlicher Hersteller, wobei ein integrierter Schaltkreis, welcher für die Ansteuerung von Ventilspulen optimiert wurde, der Chip DRV110 vom Hersteller Texas Instruments ist. Bei diesem und ähnlichen Bauelementen kann der initiale Aktivierungsstrom (I_{peak}), der Haltestrom (I_{hold}) und die Einschaltzeit über diskrete Bauteile wie Widerstände und Kondensatoren eingestellt und somit für das verwendete Ventil optimiert werden. Allerdings ergibt sich durch die Verwendung von integrierten Schaltkreisen ein erhöhter Schaltungsaufwand.

Vergleich der unterschiedlichen Ansteuerungsmöglichkeiten

Um eine optimierte Ansteuerung der Ventile zu erreichen, wurden drei unterschiedliche Ansteuerungsarten verglichen. Ziel dieser Messungen war es, eine Ansteuerungsart zu definieren, bei der die Verluste durch Wärme und die Temperatur der Spule im Inneren des Ventils innerhalb eines tolerierbaren Rahmens liegen.
In Abbildung 3.7 sind die Ergebnisse dieser Messungen dargestellt, wobei die blaue Kurve die Außentemperatur des Gehäuse bei einer Ansteuerung des Ventils mit konstanter 12V Gleichspannung abbildet und der orange Graph den Verlauf der Gehäusetemperatur bei initialer Ansteuerung mit 12V Gleichspannung und Reduktion nach einer Dauer von 500ms auf 5V Gleichspannung darstellt. Der Verlauf der Gehäusetemperatur bei einer initialen pulsweitenmodulierten Versorgungsspannung von 12V Gleichspannung mit einem Tastverhältnis von 1 innerhalb der ersten 500ms nach Einschalten des Ventils und in einem darauffolgenden Tastverhältnis von 0.4 wird mittels der grünen Kurve in Abbildung 3.7 dargestellt. Die Tabelle mit den exakten Messwerten der unterschiedlichen Ansteuerungsarten ist in Abschnitt A.2 im Anhang hinterlegt.

Durch diese Messungen konnte bestätigt werden, dass die durchgehende Ansteuerung der Klemmventile bei konstanter Spannungsversorgung mit 12V Gleichspannung zu unvertretbar hohen Erwärmungen der Ventilspule und somit auch des Ventilgehäuses führt. Dies würde im Dauerbetrieb zu einer erhöhten Abnützung der Spule und somit zu einer verringerten Lebensdauer des Ventils führen. Es müssten daher weitere Konzepte zur Kühlung des Ventils und zum Abtransport der Verlustwärme entworfen werden. Um die Punkte der Norm EN-61010-1 - Sicherheitsbestimmungen für elektrische Mess-, Steuer-, Regel- und Laborgeräte einzuhalten, muss eine Auswahl der Ansteuerungsmöglichkeit getroffen werden, welche im Normalfall und im ersten Fehlerfall zu einer resultierenden Erwärmung des Gehäuses führt, die innerhalb der

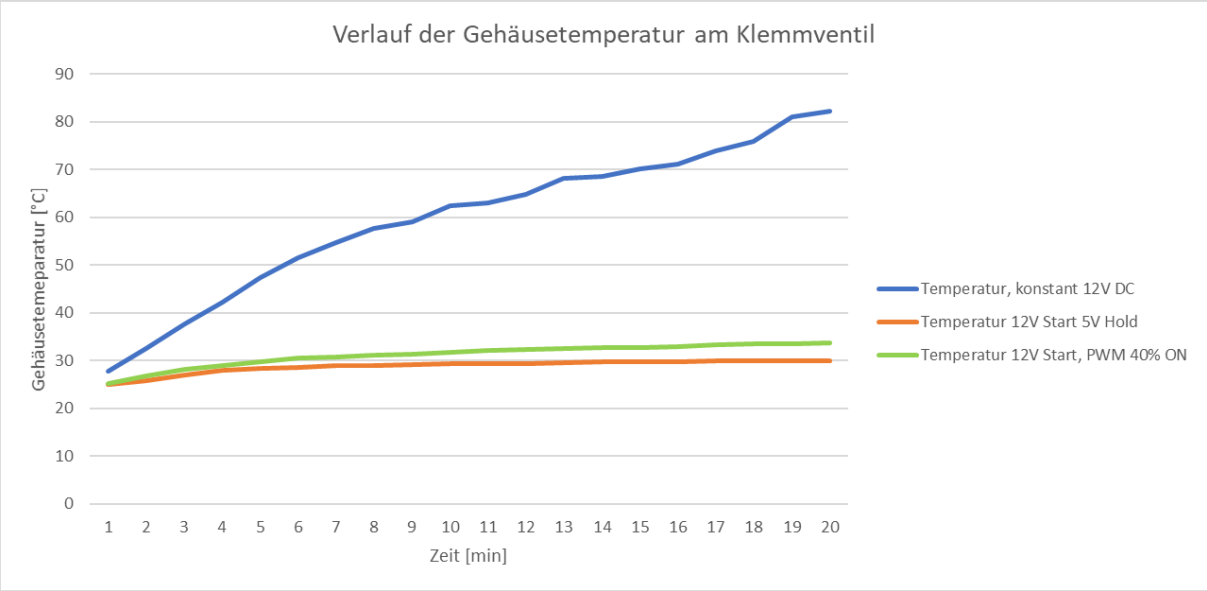


Abbildung 3.7.: Messung der Temperatur am Ventilgehäuse bei unterschiedlichen Ansteuerungsarten

tolerierbaren Grenzen liegt. Diese Bereiche und Grenzen sind in der Norm EN-61010-1, im Kapitel 10, Gerätetemperaturgrenzen und Wärmebeständigkeit, dargestellt [13].

Bei der pulsweitenmodulierten Ansteuerung und der Ansteuerung mit einer verringerten Haltespannung von 5V Gleichspannung zeigte sich, dass sich die Außentemperatur des Ventilgehäuses nur geringfügig über den Betrachtungszeitraum von 20 Minuten erhöht. Da diese beiden Ansteuerungsmöglichkeiten ähnliche Effizienz im Hinblick auf die resultierenden Verluste aufweisen, fiel die Entscheidung auf eine Pulsweitenmodulierte Ansteuerung der Ventile, vor allem aufgrund der Tatsache, dass hier nur ein geringer Schaltungsaufwand für die Ansteuerung notwendig ist, eine Einstellung des Tastverhältnisses einfach über die Digital-Ausgänge des verwendeten Mikrocontrollers möglich ist und die resultierende Erwärmung im tolerierbaren Bereich der Norm EN-61010-1 liegt.

3.2.2. Entwicklung der elektronischen Schaltung

Basierend auf den Ergebnissen der Temperaturmessung im vorigen Kapitel wurde eine elektronische Schaltung zur effizienten Ansteuerung der Ventile dimensioniert. Um die pulsweitenmodulierte Ansteuerung der Ventile zu ermöglichen, wurde ein Mikrocontroller der Marke Atmega verwendet. Dieser Mikrocontroller bildet die zentrale Steuereinheit des Perfusionssystems und wird für das Verarbeiten der Befehle, welche über die serielle Verbindung gesendet werden, eingesetzt und kann über seine GPIO (General Purpose Input Output) Pins ein Pulsweitenmoduliertes Signal ausgeben, welches von einer weiteren Einheit der Schaltung zur Ansteuerung der Ventile genutzt werden kann.

Um eine einfache Konstruktion zu ermöglichen, wurde der Mikrocontroller Atmega 328p von der Firma Atmel für die erste Variante des Systems ausgewählt. Dieser Mikrocontroller hat den großen Vorteil, das unter der Plattformbezeichnung "Arduino" bereits mehrere Prototyping-Plattformen existieren, welche die wichtigsten Bauelemente kombiniert auf einer Platine beinhalten. Das hier verwendete Prototyping-Board mit der Bezeichnung "Arduino UNO R3" beinhaltet einen Atmega328p Mikrocontroller, einen Atmega16u2 Mikrocontroller für die USB-Kommunikation, einen Spannungsregler-Chip und GPIO-Pins, die als Eingang oder Ausgang genutzt werden können. Der verwendete Atmega328 Mikrocontroller besitzt als Programmspeicher einen 32kB Flash-Speicher, 2kB RAM, und 1kB EEPROM Speicher. Außerdem sind hardwaremäßig eine serielle Schnittstelle, diverse Timer und Interrupts in den Baustein integriert. Die Spannungsversorgung des Prototyping Boards kann dabei entweder über die DC Niederspannungsbuchse (9-12V DC) oder direkt über die USB-Verbindung erfolgen [14]. Ein weiteres Prototyping-Board mit der Bezeichnung "Arduino Leonardo" wurde für eine weitere Iteration des Aufbaus verwendet. Dieses Board hat den Vorteil, dass in diesem Fall ein Mikrocontroller Einsatz findet, welcher bereits einen integrier-

ten USB Controller aufweist. Die Schaltpläne der beiden benutzten Prototyping-Boards sind im Anhang im Abschnitt A.4 abgebildet.

Die Verbindung zu den einzelnen Pins des Mikrocontrollers kann mithilfe der auf dem Prototyping-Board montierten Buchsenleisten hergestellt werden. Somit ist es einfach möglich, das Prototyping-Board und den verbauten Mikrocontroller um eine angepasste Schaltung zu erweitern. Das Prinzip für die Ansteuerung eines Ventils mittels eines Mikrocontroller-Ausgangs ist in Abbildung 3.8 dargestellt. Das Ventil ist mittels Induktivität und Serienwiderstand dargestellt, weitere parasitäre Elemente werden als vernachlässigbar angesehen. Als schaltendes Element wird ein N-Kanal MOSFET eingesetzt, in Abbildung 3.8 mit der Bezeichnung M1 dargestellt. Wenn am Gate-Anschluss des MOSFETs eine Spannung größer der Gate-Source Schwellwertspannung anliegt, wird der Drain-Source Kanal niederohmig, somit wird ein Stromfluss ermöglicht, das Magnetfeld der Spule im Ventil wird aufgebaut und der Kanal des Ventils öffnet. Zur Begrenzung des Gatestromes während des Umschaltens des Transistors, wurde ein Widerstand zwischen GPIO Ausgang und Gate des Transistors eingebaut (Widerstand R2 in Abbildung 3.8). Um bei der initialen Spannungsversorgung des Prototyping-Boards einen definierten Zustand des Ventils zu erreichen, ist ein Pull-Down Widerstand zwischen Gate und Masse jedes Transistors vorgesehen (Widerstand mit der Bezeichnung R3 in Abbildung 3.8) [15]. Beim Ausschalten induktiver Lasten kann die Energie des Magnetfeldes nicht schlagartig abgebaut werden und der Strom fließt über den MOSFET weiter, welcher hochohmig ist. Die in Abbildung 3.8 eingezeichnete Diode D1 hat die Funktion einer Freilaufdiode und verhindert dass der induzierte Strom eine hohe Spannung erzeugt und den Transistor zerstört.

In Abbildung 3.9 ist die gesamte elektronische Schaltung zur Ansteuerung der Klemmventile dargestellt. Die Spannungsversorgung für die einzelnen Ventile wird über ein externes Netzteil mit einer Versorgungsspannung von 12V DC gewährleistet. Die

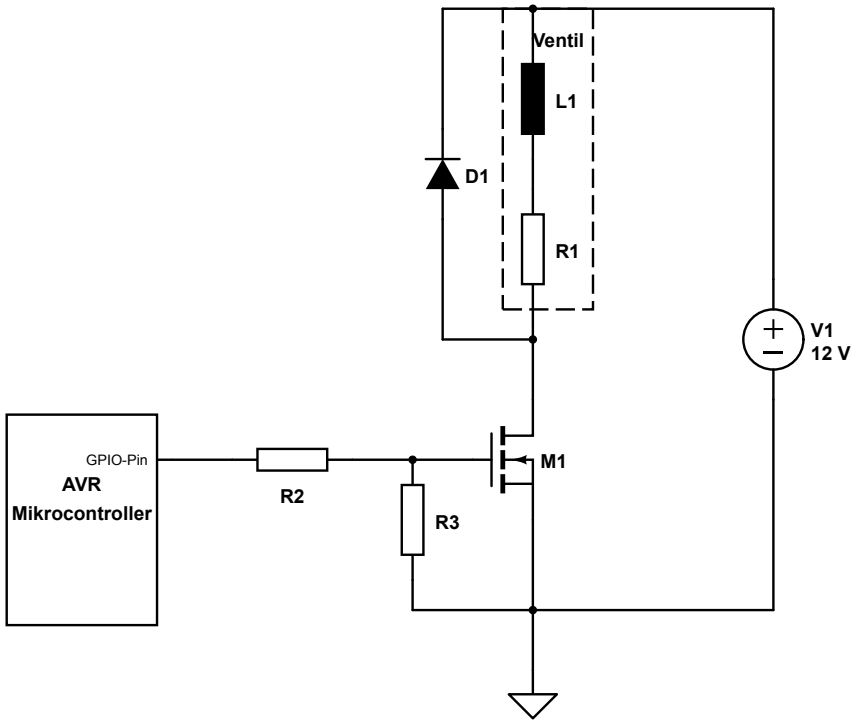


Abbildung 3.8.: Prinzipielle Schaltung zur Ansteuerung eines Ventils. Das Ventil wird durch Induktivität und Serienwiderstand modelliert, zusätzliche parasitäre Elemente wurden vernachlässigt

Versorgung des Mikrocontrollers erfolgt über die 5V-Verbindungsleitung der USB Schnittstelle. Somit ist es nicht notwendig, die Versorgungsspannung von 12V DC auf den Eingangsspannungsbereichs des Mikrocontrollers zu transformieren. Da die geschalteten Ventile induktive Lasten darstellen, ergibt sich durch die Trennung der Spannungsversorgung für den Mikrocontroller und für die Klemmventile eine Verringerung auftretender Störimpulse.

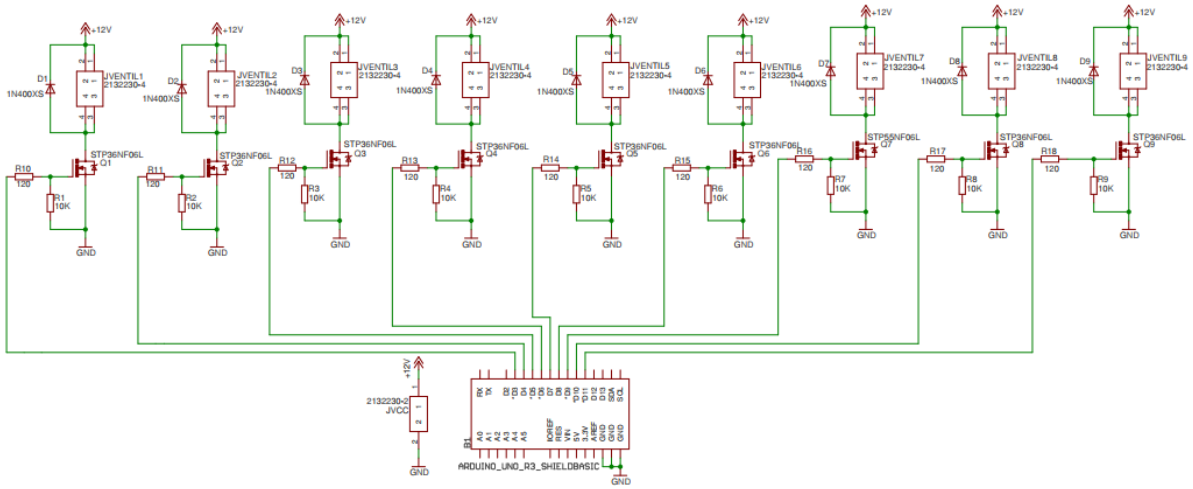


Abbildung 3.9.: Schaltung zur Ansteuerung der Ventile mithilfe des eingesetzten Mikrocontrollers

3.2.3. Leiterplattendesign

Für das Design der Leiterplatte wurde die EDA Software Eagle (Autodesk Inc., USA) verwendet. Ein wichtiger Punkt bei der Konstruktion der Platine war die Notwendigkeit die Außenabmessungen in einem ähnlichen Rahmen wie die Abmessungen des Prototyping-Boards auszuführen. Das pro Kanal verwendete Klemmventil und die dazugehörige Leuchtdiode wurden parallel dazu geschalten, somit ergeben sich 4 Leitungen pro Kanal ausgehend von der Platine zur Leuchtdiode und zum Ventil. In Abbildung 3.10 ist das in der Software Eagle erstellte Modell der Platine abgebildet.

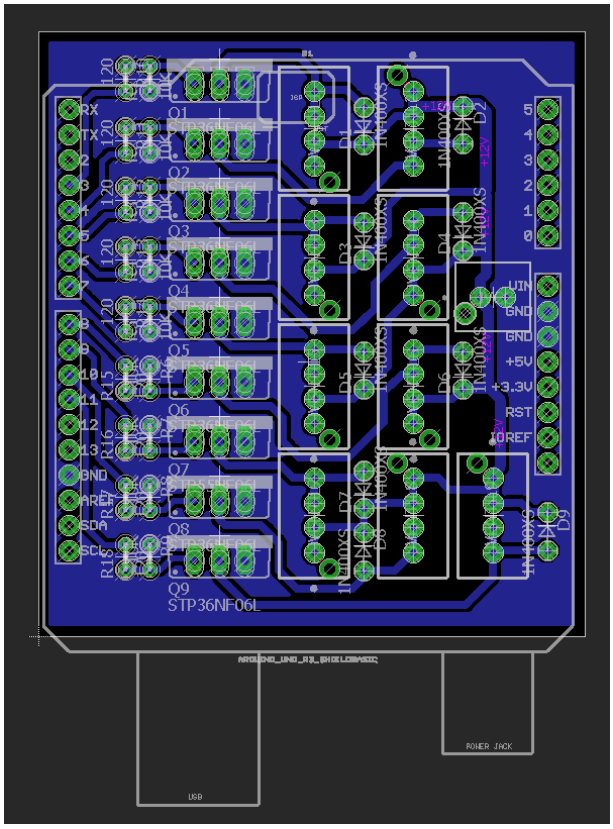


Abbildung 3.10.: Platine für das Prototyping-Board

Zur einfachen Einpassung in das Gehäuse wurden weitere 3D-CAD Modelle des Prototyping Boards und der entwickelten Platine erstellt. Abbildung 3.11 zeigt ein CAD Modell des verwendeten Prototyping-Boards. In Abbildung 3.12 ist die eigens erstellte Platine dargestellt, welche über Steckerleisten mit dem Prototyping-Board verbunden ist. Um eine einfache Fertigung zu ermöglichen, wurde nur ein Layer für das Routing der elektronischen Signale eingesetzt.

Das CAD Modell der erstellten Platine wurde anschließend in das Modell des Gesamtsystems integriert, um eine einfache Einpassung zu ermöglichen.

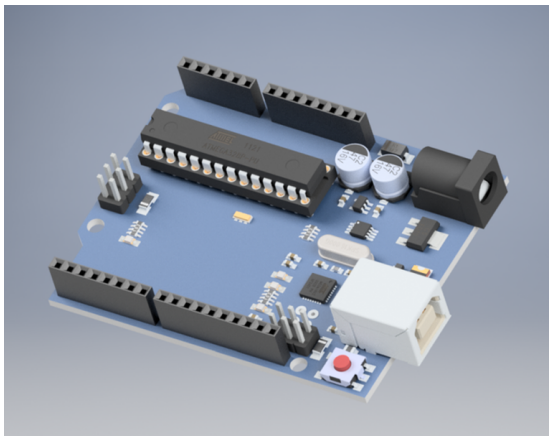


Abbildung 3.11.: Prototyping-Plattform Arduino UnoR3

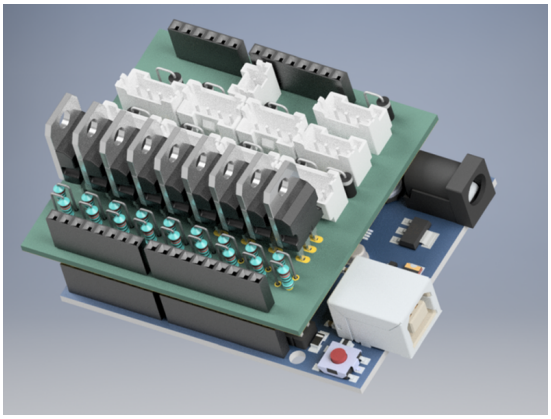


Abbildung 3.12.: Prototyping-Plattform mit aufgesetzter erstellter Platine

4. Entwicklung und Evaluierung der Software Komponenten

Der zweite Teil dieser Arbeit stellt die Entwicklung der Software zur Ansteuerung und Bedienung des Perfusionssystems dar. Dabei wurde zum einen eine Programmierung des verwendeten Mikrocontrollers durchgeführt, zum anderen wurde eine Software in der Programmiersprache C++ entwickelt, welche eine möglichst einfache Bedienung und automatische Ansteuerung des Perfusionssystems ermöglichen sollte. Um eine einfache Erweiterung des Systems für andere Einsatzzwecke gewährleisten zu können, wurde auf wichtige Aspekte der Norm EN-62304, welche Software Lebenszyklus Prozesse beschreibt, Rücksicht genommen, jedoch sollte beachtet werden, dass das entwickelte System kein Medizinprodukt darstellt, sondern im definierten Anwendungsfall die Funktion eines Laborgerätes hat. Somit ist die strikte Einhaltung der Norm EN-62304 nicht zwingend erforderlich, aufgrund der vorgesehenen zukünftigen Integration und Erweiterung werden in dieser Arbeit wichtige Aspekte der Norm EN-62304 bereits berücksichtigt.

4.1. Aspekte der Norm EN-62304

Ein zentraler Punkt bei der Softwareentwicklung ist die Tatsache, dass Software ein Produkt ist, dessen Konformität im fertigen Zustand nicht mehr mit einem vertretbaren Aufwand überprüft werden kann. Somit wird großteils der Entwicklungsprozess der Software überprüft und angenommen, dass ein zuverlässiger Prozess der Entwicklung auch ein zuverlässiges Ergebnis liefert. Die Norm EN-62304 setzt außerdem voraus, dass der Hersteller ein Qualitätsmanagementsystem nach EN ISO 13485 einsetzt, und für die Software den Risikomanagementprozess nach EN ISO 14971 anwendet [16].

Die Norm EN-62304 Medizingeräte-Software - Software Lebenszyklus-Prozesse definiert den Ablauf des Software-Entwicklungs-Prozesses wie folgt, wobei die Ausprägung der einzelnen Prozessschritte abhängig von der jeweiligen Software Sicherheitsklasse sind [17]:

1. Planung der Software-Entwicklung
2. Analyse der Software Anforderungen
3. Design der Software-Architektur
4. Detailliertes Software-Design
5. Implementierung der Software-Einheiten
6. Integration der Software
7. Software-Prüfung
8. Software-Freigabe

4.1.1. Klassifizierung der Software

Durch die Norm EN-62304 werden drei Sicherheitsklassen bestimmt, mit der Motivation den Aufwand der Software-Dokumentation an den Grad möglicher Schäden

anpassen zu können. Außerdem ist die Wahrscheinlichkeit, dass ein Software-Fehler eintritt, mit 100% vorgegeben. Aufgrund des resultierenden Risikopotentials wird die Software in eine der drei folgenden Sicherheitsklassen eingeteilt [17]:
Klasse A: keine oder vernachlässigbar geringe Verletzung oder Gesundheitsschädigung möglich;
Klasse B: keine schwere Verletzung möglich;
Klasse C: schwere Verletzung oder Tod möglich

Beim Betrieb der im Zuge dieser Arbeit entwickelten Software kann auch beim Eintreten von Software-Fehlern keine Gesundheitsschädigung resultieren, somit wird die Software in diesem Fall in die Software-Sicherheitsklasse A eingeteilt.

Aufgrund der Einstufung in die Software-Sicherheitsklasse A ergibt sich eine Reduktion der durchzuführenden Schritte laut Norm EN-62304 auf folgende Prozessschritte:

1. Planung der Software-Entwicklung
2. Analyse der Software Anforderungen
3. Implementierung der Software-Einheiten
4. Software-Prüfung
5. Software-Freigabe

4.1.2. Vorgehensmodelle der Software-Entwicklung

Durch die Norm EN-62304 ergeben sich Anforderungen an den Softwareentwicklungsprozess, jedoch keine unverbindlichen Vorschriften zur Umsetzung dieser. Somit ergibt sich die Herausforderung, die Anforderungen aus den Normen in einem Entwicklungsprozess umzusetzen [18]. Im (nicht normativen) Anhang der Norm EN-62304 sind

drei exemplarische Vorgehensmodelle beschrieben, welche eingesetzt werden können, um die Anforderungen der Norm zu erfüllen, welche zunächst kurz beschrieben und verglichen werden.

Wasserfall-Modell

Das einfachste Modell, welches exemplarisch in der Norm EN-62304 dargestellt wird, ist das Wasserfallmodell. Bei dieser Vorgehensweise wird festgelegt, dass bevor zur nächsten Phase übergegangen werden darf, die vorgehende zunächst komplett abgeschlossen werden sollte. Das schematische Vorgehen im Wasserfallmodell ist in Abbildung 4.1 dargestellt. Aufgrund der stufenweise Vorgehensweise wird dieses Modell auch als Wasserfallmodell bezeichnet

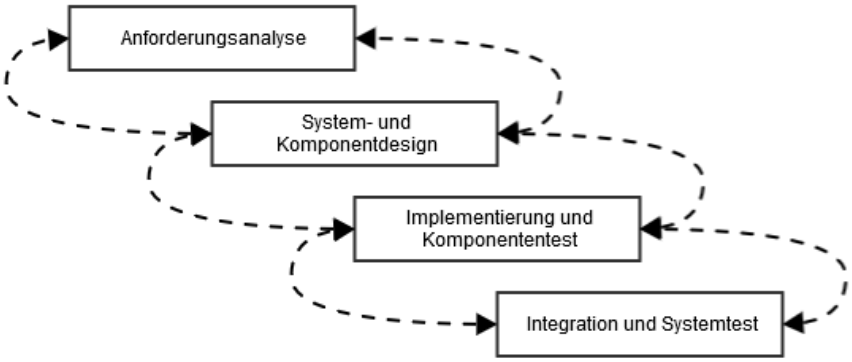


Abbildung 4.1.: Schematisches Vorgehen nach dem Wasserfallmodell

Die resultierende Einfachheit des Modells stellt einen großen Vorteil dar, wobei das Grundprinzip für jeden Projektmitarbeiter sichtbar ist, außerdem kann die Vorgehensweise ohne größere Änderungen beispielsweise auf Entwicklung von Hardwarekomponenten oder elektronischer Schaltkreise übertragen werden. Ein möglicher Nachteil des Modells ist das Risiko, dass Fehler erst in einer späteren Phase entdeckt werden,

beispielsweise wird in der Phase "Integration und Systemtest" ein Fehler aufgedeckt, welcher allerdings den Ursprung in der Architektur hat und diese Phase bereits formell abgeschlossen ist. Aufgrund der Tatsache, dass beim Auftreten von möglichen Fehlern es oft notwendig ist entgegen der Richtung des Wasserfalls zurückzugehen, wird dieses Modell nur selten in der Praxis angewandt und stellt meist nur eine theoretische Vorgehensweise dar [18].

Iterativ-inkrementelles Modell

Eine weitere Möglichkeit für die Vorgehensweise bei der Software-Entwicklung ist das Iterativ-inkrementelle Modell. Dieses sollte eine zeitnahe Fehlererkennung ermöglichen, wobei alle Aktivitäten des Entwicklungszyklus mehrfach durchlaufen werden. Aufgrund der Annahme, dass in jeder Iteration ein Fortschritt in der Entwicklung der Software vorliegt (siehe Abbildung 4.2), wird das Modell Iterativ-inkrementelles Modell genannt [18].

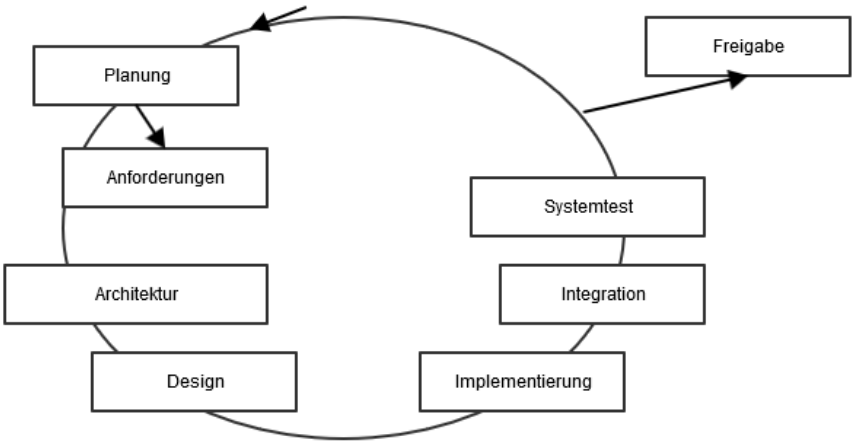


Abbildung 4.2.: Schematisches Vorgehen nach einem iterativ-inkrementellen Modell

Somit ergibt sich der Vorteil, dass beispielsweise bei einer Software, bei der 10 Funktionen entwickelt werden sollten, diese Funktionen nicht alle in einem einzigen Entwicklungszyklus erstellt werden. Der Ablauf wird dabei beispielsweise in fünf Iterationen unterteilt wobei in jeder Iteration zwei Funktionen entwickelt werden. Die Korrektur etwaiger Fehler am Ende einer Iteration im Testlauf kann einfach im nächsten Durchlauf eingeplant werden, somit resultiert nur ein minimaler zusätzlicher Zeitaufwand.

V-Modell

Eine Weiterentwicklung des Wasserfall Modells stellt das V-Modell dar. Durch den Einsatz dieses Modelles sollte das schwerwiegendste Problem beim Wasserfallmodell beseitigt werden, nämlich dass Fehler, welche in einer frühen Phase gemacht wurden, erst in einer späten Phase erkannt werden [18]. Beim V-Modell steht dabei jeder konstruktiven Phase eine entsprechende Testphase gegenüber, wie in Abbildung 4.3 dargestellt wird.

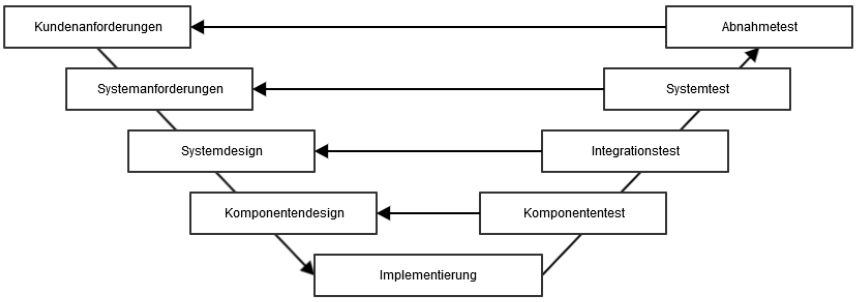


Abbildung 4.3.: Schematisches Vorgehen nach dem V-Modell

Das V-Modell wird sehr oft im Zuge der Software-Entwicklung eingesetzt. Der grundsätzliche Vorgang kann wie folgt beschrieben werden. Zunächst wird basierend

auf die Vorgaben des Lastenheftes eine Risikoanalyse durchgeführt, ein Sicherheitskonzept entwickelt und das Pflichtenheft erstellt. Darauf folgend wird die Software-Architektur entwickelt, schrittweise in Subsysteme gegliedert und weiter in kleinere Einheiten zerlegt. Dies wird solange durchgeführt, bis überschaubare und direkt überprüfbare Module vorliegen, welche verifiziert und über Komponenten und Subsysteme zum Gesamtsystem zusammengestellt werden.

Aufgrund der einfachen Unterteilung der Software in Teilkomponenten, wurde im Zuge der Software-Entwicklung zur Ansteuerung des Perfusionssytems das V-Modell als Vorgehensmodell ausgewählt.

4.1.3. Software-Entwicklungs-Prozess

Planung der Software Entwicklung

Im ersten Prozessschritt erfolgte die grundsätzliche Planung der Software Entwicklung. Am Ende dieses Schrittes sollten die Prozeduren zur Entwicklung festgelegt sein und in einem Plan resultieren, welcher die eingesetzten Methoden, Standards und Tools festlegt. In der Norm EN-62304 ist allerdings nur festgelegt, dass ein Software-Plan zwingend erstellt werden muss, allerdings ist nicht das Aussehen oder die Struktur dieses Plans definiert. Eine beispielhafte Vorlage für ein umfassendes Planungsdokument ist unter [19] angegeben, die untenstehende Struktur der Entwicklungsplanung basiert auf Teilelementen dieses Dokuments.

Standards, Methoden, Tools und Techniken

Als grundsätzliche Programmiersprache zur Erstellung der Anwendungssoftware wird die Sprache C++ in der Version 17 verwendet. Bei der Programmierung des Mikrocontrollers erfolgt der Einsatz der Programmiersprache C in der Version 11,

wobei Teilmodule mittels der Entwicklungsumgebung Arduino erstellt werden. Für die Erstellung der grafischen Benutzeroberfläche wird das Framework QT eingesetzt, zur Beschreibung der einzelnen grafischen Elemente die deklarative Sprache QML. Um eine Datenpersistenz gewährleisten zu können, erfolgt das Abspeichern von Daten in eine SQLite Datenbank. Für die Versionsverwaltung der programmierten Software wird das Verwaltungssystem "Git" eingesetzt.

Software Dokumentation

Das Toolkit Doxygen wird zur Dokumentation der Software herangezogen. Mittels Doxygen kann Source Code unter Zuhilfenahme erstellter Kommentare analysiert und automatisiert eine Dokumentation generiert werden [20]. Ein Vorteil dieser Dokumentationsmethode gegenüber einer externen Entwicklerdokumentation ist, dass eine Redundanz so gut wie möglich vermieden wird und die Dokumentation bei einer Änderung der Strukturen oder Funktionen direkt im Quelltext aktualisiert werden kann. Die Dokumentation der gesamten Anwendungssoftware ist im Anhang dieser Arbeit im Abschnitt A.5 hinterlegt.

Analyse der Software Anforderungen

Folgende Aspekte sollte im Zuge der Software-Entwicklung behandelt werden [18]:

Benutzerschnittstellen:

- Bedienung über eine grafische Benutzeroberfläche, mit der Möglichkeit den Fluss einzelner Kanäle über Aktivieren oder Deaktivieren dazugehöriger Schaltflächen ein- oder auszuschalten.

- Das System sollte grafisch nachgebildet werden, so dass auf einem Blick ersichtlich ist, welcher Kanal derzeit geöffnet ist und welche Substanz sich in welchem Behältnis befindet.
- In einem zweiten Modus sollte es möglich sein, Schaltvorgänge definieren zu können, welche sequentiell abgearbeitet werden. Diese Schaltvorgänge sollten ähnlich einem Diagramm übersichtlich aneinander angereiht dargestellt werden.
- Über Drop-Down Menüs sollte den einzelnen Kanälen definierte Bezeichnungen der beinhalteten Substanz zugewiesen werden können. Über ein Pop-Up können neue Substanzen permanent definiert werden.
- Schaltvorgänge im automatischen Modus sollten mit weiteren Informationen wie Datum, Zeitpunkt, Dauer, Puffernamen persistent gespeichert und somit dokumentiert werden.

Grafische Benutzeroberfläche:

- Die Entwicklung der grafischen Benutzeroberfläche sollte nach den Richtlinien zur Erstellung von Apps für die Universelle Windows-Plattform von Microsoft erfolgen.
- Die Software sollte einfach bedienbar gestaltet werden. In einem definierten Feld sollte die derzeitige Seite angezeigt sein, außerdem erfolgt eine Anzeige des aktuellen Systemstatus und des aktuellen Datums und der Uhrzeit.

Verhalten des Systems auf Benutzeraktionen:

- Bei Auswählen des gewünschten Behältnisses sollte je nach aktuellen Status des Kanals der ausgewählte Kanal entweder geöffnet oder geschlossen werden.
- Wenn automatische Schaltvorgänge definiert werden, darf es nicht möglich sein nach einem Schaltvorgang darauffolgend den gleichen Kanal einzuschalten.

- Es sollte möglich sein, automatische Durchläufe lokal in eine Datenbank und in eine externe Datei im CSV Format zu exportieren.
- Die gespeicherten Durchläufe sollten als Template ausgewählt werden können und somit für spätere Versuche einfach genutzt werden können.
- Der Abbruch eines automatischen Durchlaufes erfolgt mittels einer zusätzlichen Bestätigung in einem Pop-Up Fenster.
- Während des automatischen Modus darf es nicht möglich sein, zwischen den unterschiedlichen Funktionen des Programmes zu wechseln.
- Bei Verkleinern des Hauptfensters muss es weiterhin möglich sein, manuelle Schaltvorgänge durchzuführen. Die grafische Abbildung des Systems wird durch Schalter ersetzt, wenn die Fensterabmessungen nicht für die Darstellung der Systemabbildung ausreichen.

Installationsumgebung:

- PC mit Windows Betriebssystem
- mindestens 1GB RAM Arbeitsspeicher
- mindestens 100MB freier Festplattenspeicher
- vorhandener USB-Anschluss

Technische Schnittstellen:

- Die Verbindung zu den elektronischen Komponenten erfolgt über die serielle Verbindung.
- Beim Starten der Anwendungssoftware erfolgt der Verbindungsaufbau zum Mikrocontroller, wobei der benutzte Anschluss automatisch festgelegt wird..
- Der Export und Import von Daten und Schaltvorgängen sollte über das XML Dateiformat ermöglicht werden.

- Puffernamen und Schaltvorgänge sollten in eine SQLite Datenbank abgespeichert werden können.
- Zum zeitgesteuerten Schalten der Kanäle werden die Timer der Anwendungssoftware verwendet.
- Zwischen dem eingestellten Schaltzeitpunkt und dem tatsächlichen Schaltvorgang des Ventils dürfen höchstens 500ms Toleranzzeit liegen.

Implementierung der Software-Einheiten

Zunächst wurde die benötigte Software für den eingesetzten Mikrocontroller entwickelt und evaluiert. Die Entwicklung der Anwender-Software ist grundsätzlich in die Entwicklung von Frontend und Backend Komponenten aufgeteilt. Im Zuge der Programmierung des Backends wurde ein C++ Programm erstellt, wobei das Paradigma der objektorientierten Programmierung genutzt wurde. Mittels des Backend-Programmes ist es möglich, eine Verbindung zu den elektronischen Komponenten des Perfusionssystems aufzubauen und Befehle für Schaltvorgänge an das System zu senden. Das Frontend stellt die Schnittstelle zum Benutzer dar, und dient der Anzeige und Verarbeitung einer grafischen Benutzeroberfläche.

Entwicklung der Mikrocontroller-Software

Die Firmware des Mikrocontrollers wurde mithilfe der Plattform "Arduino" entwickelt, welche ein schnelles Prototyping elektronischer Systeme ermöglicht. Es ist keine zusätzliche Hardware zum Programmieren des Mikrocontrollers erforderlich, da der Mikrocontroller bereits vorab einen Bootloader enthält, um eine direkte Programmierung über die USB Verbindung zu gewährleisten. Die Mikrocontroller-Programmierung ist in drei Bereiche eingeteilt:

- Definieren eines Kommunikationskonzeptes zum Aufbauen der Verbindung
- Verarbeiten der Befehle
- Durchführen der Schaltvorgänge mit zeitgesteuerter PWM-Aktivierung

Kommunikationskonzept

Die grundsätzliche Kommunikation zwischen PC mit laufender Anwendungssoftware und dem Mikrocontroller mit der erstellten Platine zum Ansteuern der Ventile erfolgt über eine serielle Verbindung und eigens definierter Befehle. Der größte Vorteil dieser Ansteuerungsart ist die Möglichkeit einer seriellen Verbindung, welche bereits über eine vorhandene USB-Schnittstelle am Prototyping-Board gewährleistet ist. Die für die Verwendung der seriellen Schnittstelle erforderlichen Einstellungen sind in Tabelle 4.1 zusammengefasst.

Tabelle 4.1.: Einstellungen für die Kommunikation über die serielle Schnittstelle

Parameter	Wert
Baud-Rate	115200 Bd
Daten-Bits	8
Parität	Keine
Stop-Bits	1
Flusskontrolle	keine Flusskontrolle

Struktur der Befehle

Für die Ansteuerung der Ventile wurden eigens definierte Befehle zu einem Kommunikationsprotokoll zusammengefasst. Diese werden vom Mikrocontroller verarbeitet, wobei je nach eingegebenen Befehlen entweder das Ventil geöffnet oder geschlossen wird. In Tabelle 4.2 sind die möglichen Befehle dargestellt. Das erste Zeichen im Befehl (”!”) signalisiert dem Mikrocontroller, das ein Befehl empfangen wurde, welcher weiterverarbeitet werden muss. Anschließend wird darauf durch das zweite Zeichen

(”o”,”c” oder ”t”) definiert, in welchem Zustand das Ventil geschaltet werden sollte. Mithilfe des dritten Zeichens wird der zu schaltende Kanal definiert.

Tabelle 4.2.: Definierte Befehle zur Ansteuerung der Ventile

Befehl	Vorgang	Beispiel
!o#	Öffne Ventil mit der Nummer #	!o1 - Öffne Ventil 1
!c#	Schließe Ventil mit der Nummer #	!c1 - Schließe Ventil 1
!t#	Toggle Ventil mit der Nummer #	!t1 - Toggle Ventil 1

Schaltvorgänge mit PWM-Aktivierung

Nachdem ein Befehl zur Aktivierung eines Kanals verarbeitet wurde, wird der zugehörige Pin des Mikrocontrollers auf High-Pegel gesetzt. Nach der definierten Zeit von 500ms wird auf dem geschalteten Pin eine softwarebasierte Pulsweitenmodulierung mit festgelegtem Duty-Cycle gestartet. Da der verwendete Mikrocontroller nur sechs Pins mit Hardware PWM-Funktion aufweist, wird für die Pulsweitenmodulierung eine softwareseitige PWM-Funktion, welche die integrierten Timer zur Ausgabe der High-Pegel und Low-Pegel nutzt, eingesetzt. Der Quellcode der Mikrocontrollersoftware ist im Anhang im Abschnitt A.3 hinterlegt.

Entwicklung der Anwendungssoftware

Die Anwendungssoftware wurde nach dem Model View Controller Entwurfsmuster entwickelt. Durch diese Methode werden die Verantwortlichkeiten der einzelnen Software-Komponenten in drei Rollen eingeteilt [21]:

- Rolle der Darstellung (View)
- Rolle der Datenverwaltung (Model)
- Rolle der Kontrolle von Eingaben und Datenänderungen (Controller)

Durch die Trennung der Verantwortlichkeiten ist beispielsweise die Möglichkeit gegeben, das Benutzerinterface getrennt von der übrigen Logik anzupassen und zu konfigurieren.
In Abbildung 4.4 ist die Umsetzung des MVC-Entwurfsmusters im Zuge der Entwicklung der Anwendungssoftware dargestellt. In der Rolle des Controllers fallen die Klassen "Perfusionssystem, Channel, FileIO und Database", welche für das Verarbeiten von Befehlen benötigt werden und den Import und Export von Daten steuern. Die Verwaltung der darzustellenden Daten wird von den Klassen durchgeführt, welche in der Rolle "Model" aufgeführt sind. Die unterschiedlichen Klassen werden für die Verwaltung der Puffernamen, der gespeicherten Vorlagen und der gespeicherten Schaltvorgänge eingesetzt. In der View Rolle sind die für die Interaktion mit dem Benutzer erforderlichen grafischen Module aufgelistet.

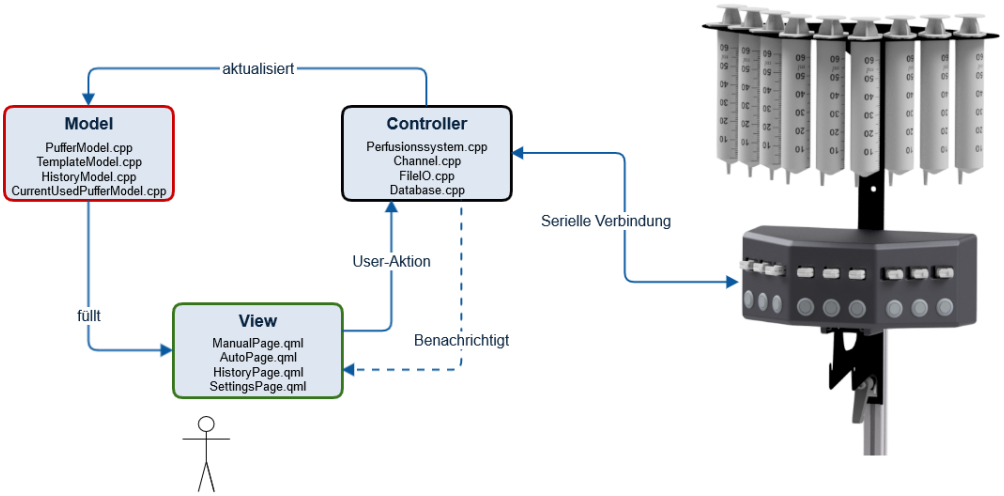


Abbildung 4.4.: Module der Anwendungssoftware nach dem MVC-Entwurfsmuster

Backend Entwicklung

Die Entwicklung der Backend-Software umfasste die Konzipierung eines Programmes

in der Programmiersprache C++, welches für den grundsätzlichen Verbindungsaufbau zwischen Mikrocontroller und PC eingesetzt werden kann. Das Backend besteht aus zwei Hauptklassen, welche angelehnt an das Paradigma der objektorientierten Entwicklung definiert wurden. Somit wurde das gesamte Objekt "Perfusionssystem" mithilfe der dazugehörigen Klasse modelliert. Beim erstmaligen Starten der Anwendersoftware wird ein Objekt der Klasse Perfusionssystem erstellt, welches ein Array von Objekten der Klasse Channel als privates Attribut enthält. Die Klasse Channel bildet dabei einen einzelnen Kanal, welcher über die Methoden des dazugehörigen Objektes ein- und ausgeschaltet werden kann, nach. Die Struktur der zwei Hauptklassen Perfusionssystem und Channel ist in Abbildung 4.5 dargestellt.

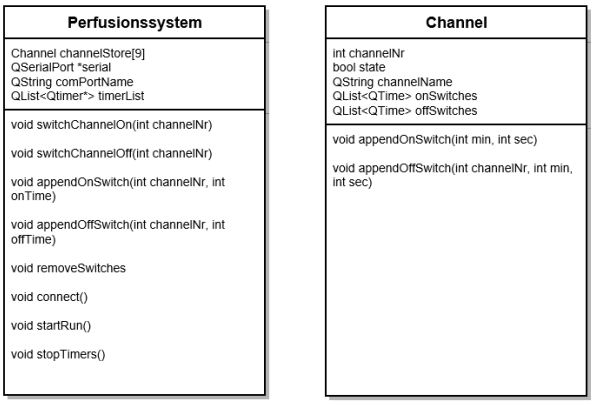


Abbildung 4.5.: Struktur der Klassen "Perfusionssystem" und "Channel"

Um die Struktur der Software so übersichtlich wie möglich zu gestalten, wurden weitere Klassen für den Import und Export von Vorlagen und Puffernamen erstellt. Die Klasse "FileIO" wird in diesem Zusammenhang für das Einlesen und Schreiben von CSV Dateien benötigt und kann benutzt werden, um mittels definierter Methoden Vorlagen und Protokolle zu importieren und exportieren.
Der strukturelle Aufbau der CSV Datei für den Import und Export von Vorlagen und Protokollen ist in Abbildung 4.6 dargestellt. Zwingend muss in der CSV Datei in jeder

verwendeten Zeile die Nummer des zu schaltenden Kanals in Spalte 2 definiert und die Dauer des Schaltvorgangs in Spalte 3 beschrieben sein. Falls keine Startzeit in Spalte 4 definiert ist, reiht die Anwendungssoftware den jeweiligen Schaltvorgang sequentiell hinter einem möglichen vorhergehenden Schaltvorgang. Die übrigen Felder sind für eine einfache Dokumentation des Protokolls vorgesehen, müssen allerdings nicht zwingend definiert sein.

Channel Names	Channel-Nr	Duration [mm:ss]	Start-Time [mm:ss]	Description:	Description-Text
Optional			Optional	Optional	Optional

Abbildung 4.6.: Aufbau der CSV Datei für Import und Export von Schaltvorgängen

Die Klasse "Database" beinhaltet Methoden zum Definieren und Abspeichern neuer Puffernamen in eine SQLite Datenbank. Das Paket SQLite ist ein relationales Datenbanksystem, welches im Gegensatz zu Client-Server Datenbanksystemen direkt in das Endprogramm integriert werden kann. Ein weiterer Vorteil dieses Datenbanksystems stellt die Tatsache dar, dass nur eine einzige Bibliothek für den Datenbankzugriff notwendig ist. Der größte Vorteil von SQLite ist somit die geringe Größe der Bibliothek und der verringerte Administrationsaufwand im Vergleich zu den meisten anderen Datenbanksystemen [22].
Die Datenbank ist in die Tabellen "Puffer" und "Template" aufgeteilt, wobei die Tabelle Puffer die Bezeichnung aller verwendeten Puffer beinhaltet und die Tabelle Template zur Speicherung der Vorlagen, welche für automatische Durchläufe genutzt werden, dient. In der Tabelle Template werden für jeden Kanal die Einschaltzeitpunkte und die Dauer der Schaltvorgänge durch ein Trennzeichen getrennt abgespeichert.

Frontend Entwicklung

Für die Entwicklung des Frontends und der Umsetzung der grafischen Benutzero-

berfläche wurde das Framework QT in Form der deklarativen Programmiersprache QML eingesetzt. Die Syntax der Sprache QML ist angelehnt an JSON (JavaScript Object Notation) und es werden JavaScript Ausdrücke innerhalb von QML Dateien oder in seperaten Dateien unterstützt [23].

Als Startpunkt für die Entwicklung der grafischen Benutzeroberfläche wurde ein MockUp erstellt, welches in Abbildung 4.7 dargestellt ist, wobei dieses grob die Struktur der Oberfläche definieren sollte. Um die Oberfläche des Bildschirms optimal ausnutzen zu können, sollte es möglich sein, die blau hinterlegten Bereiche der Benutzeroberfläche zu minimieren.
Im Kopfbereich des Programms wird der Name des derzeitigen Fensters angezeigt. Den größten Bereich der Oberfläche stellt das Hauptfenster dar, in dem die zur Verfügung stehenden Bedienfunktionen dargestellt werden. Der linke Bereich der grafischen Benutzeroberfläche dient der Navigation innerhalb der Software. Somit sind hier Buttons zum Wechseln zwischen manuellen und automatischen Modus angebracht. Unterhalb der Navigationsleiste ist es möglich, den einzelnen Kanälen Namen unterschiedlicher Substanzen zuzuweisen. Am unteren Ende der gesamten Benutzeroberfläche wird die aktuelle Uhrzeit und der aktuelle Status der Verbindung angezeigt. Darüber befindet sich das lokale Protokoll, in welchem alle manuellen Schaltvorgänge mit der Kanal-Nummer, Bezeichnung des jeweiligen Kanals, und Uhrzeit dargestellt werden.

Beim erstmaligen Starten des Programmes wird zunächst das Fenster zur manuellen Ansteuerung des Perfusionssystems angezeigt, welches in Abbildung 4.8 abgebildet ist. In diesem Fenster ist der Aufbau des Perfusionssystems schematisch nachgebildet. Um nun einen Kanal zu öffnen und somit einen Fluss der Pufferlösung zu ermöglichen, genügt entweder ein Linksklick auf das jeweilige abgebildete Pufferbehältnis oder ein Aktivieren der zum Kanal gehörigen Schaltfläche. Die hier vorgenommenen Schaltvorgänge werden automatisch am unteren Bereich des Fensters mitprotokolliert. Außer-

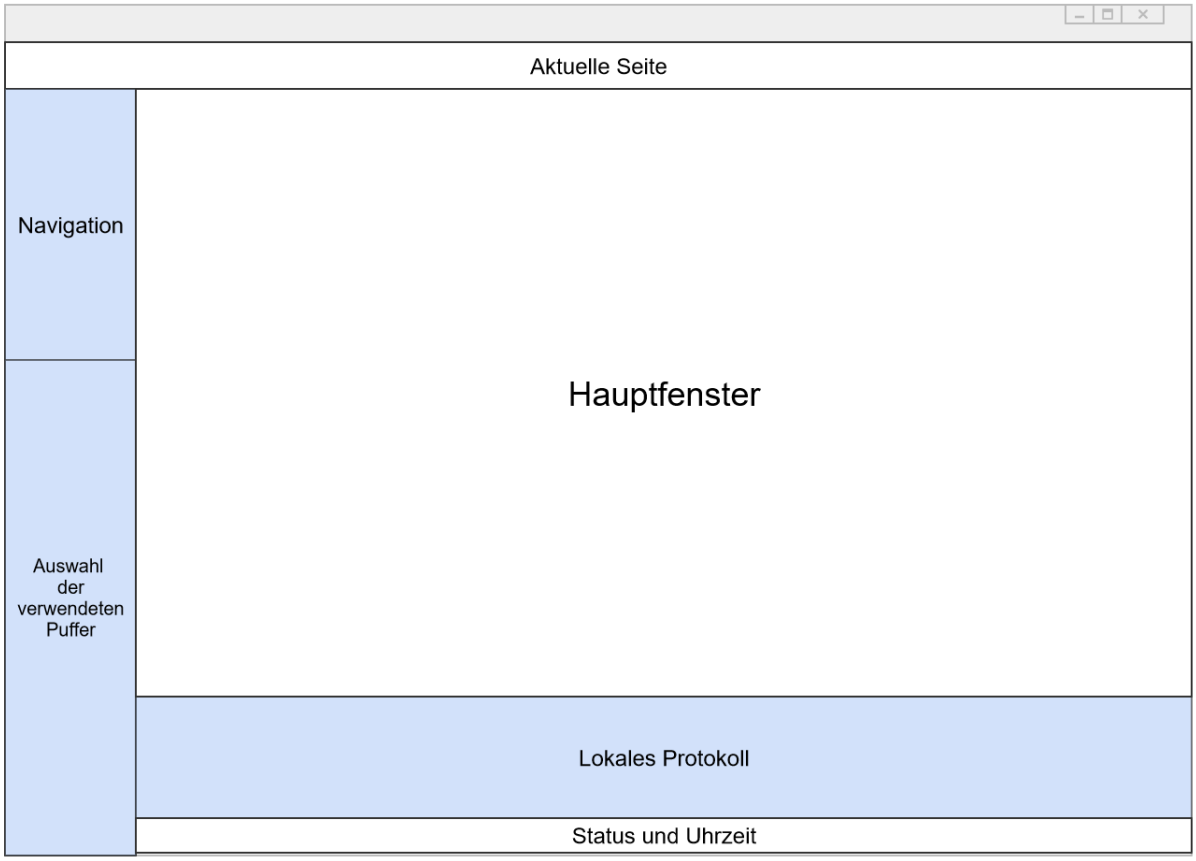


Abbildung 4.7.: Mockup der Benutzeroberfläche

dem ist es möglich, über eine Log Funktion die Schaltvorgänge mit den zugehörigen Zeitpunkten mitzuloggen und somit für spätere automatische Durchläufe als Vorlage einzusetzen.

Als alternativer Modus für die Ansteuerung des Perfusionssystems ist der automatische Modus vorgesehen, welcher über einen Klick auf die Schaltfläche "Automatic Mode", die sich auf der linken Seite des Fensters im Navigationsbereich befindet, aufgerufen werden kann. Die Struktur dieser Oberfläche ist in Abbildung 4.9 dargestellt, wobei in diesem Fenster die Navigationsleiste verkleinert dargestellt ist und der Bereich des lokalen Verlaufs deaktiviert wurde.

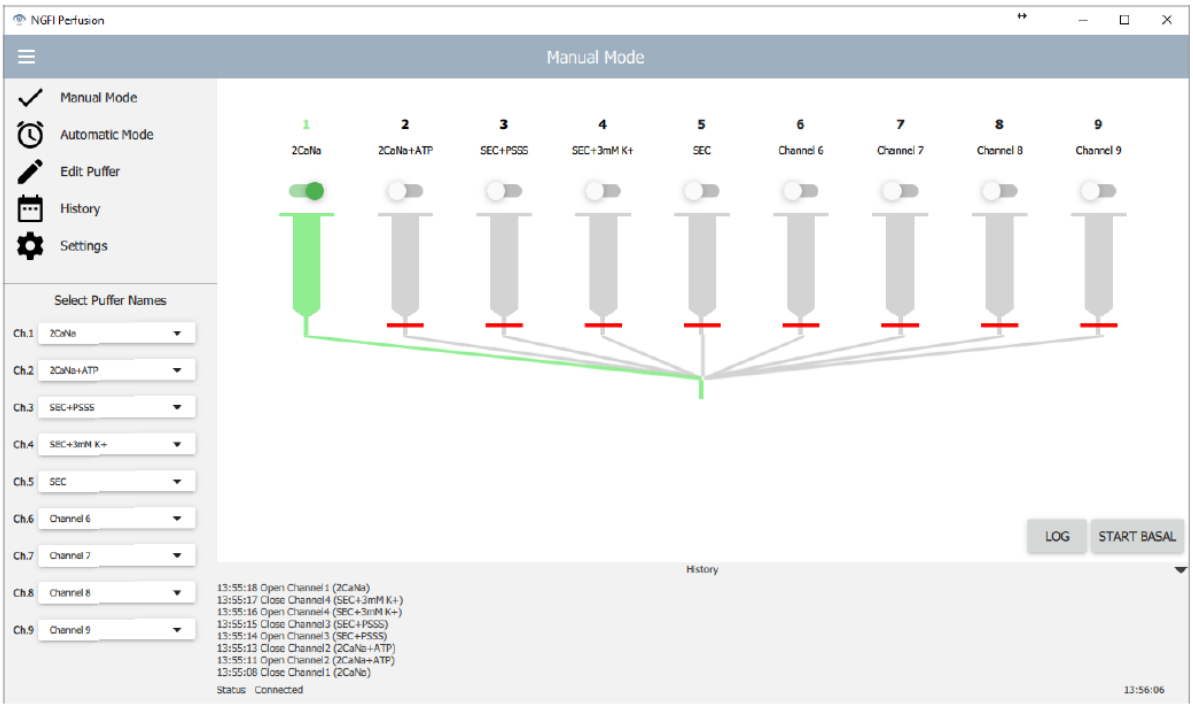


Abbildung 4.8.: Grafische Benutzeroberfläche zur manuellen Ansteuerung des Perfusionsystems

Im automatischen Modus kann der Durchlauf eines Versuches vor dem Start mit den gewünschten Schaltzeitpunkten definiert werden. Mithilfe der Schaltfläche mit der Aufschrift ”+” kann ein einzelner Schaltvorgang zur gesamten Liste hinzugefügt werden. Dieser kann mit Kanalname und Dauer definiert werden und wird anschließend an der Unterseite des Hauptfensters mithilfe grafischer Blöcke dargestellt. Die Länge des jeweiligen Blockes wird mithilfe der relativen Zeit des jeweiligen Schaltvorganges im Verhältnis zur gesamten Zeit des Durchlaufes berechnet. Die Schaltfläche ”Start” dient bei Betätigung dem Start des definierten Ablaufs, wobei die einzelnen Schaltvorgänge sequentiell abgearbeitet werden. Im Hintergrund erfolgt die Übertragung der eingegebenen Daten in die Funktionen des Backends und der Start der angelegten Timer für jeden Schaltvorgang.
Mithilfe der Funktion ”Clear” können alle festgelegten Schaltvorgänge gelöscht wer-

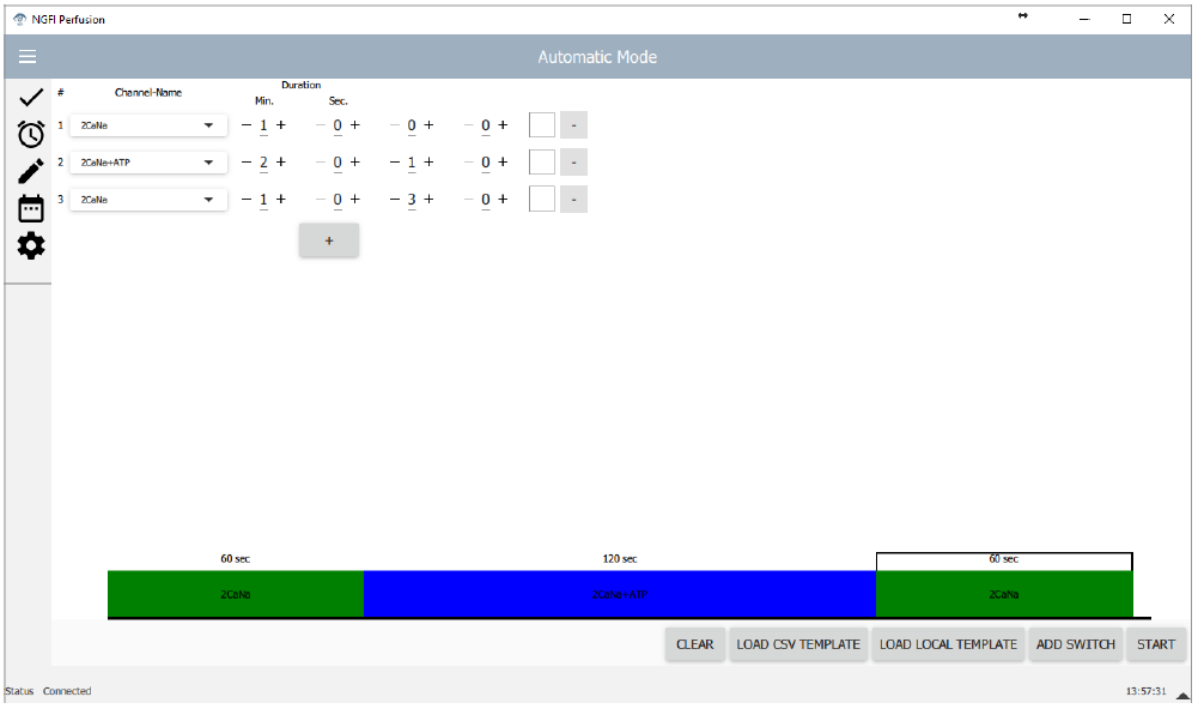


Abbildung 4.9.: Grafische Benutzeroberfläche zur automatischen Ansteuerung des Perfusionsystems

den, außerdem ist es möglich über die Schalfläche "Load CSV Template" Vorlagen, welche über CSV Dateien definiert wurden, in das Programm einzulesen. Mithilfe der Funktion "Load local Template" ist der Abruf in der Datenbank gespeicherter Durchläufe möglich.

Software-Prüfung

Die Norm 62304 schreibt eine Prüfung der Software-Anforderungen für Software-Systeme aller Sicherheitsklassen vor. Durch die Ergebnisse der Prüfung sollte nachgewiesen werden, dass alle Software Anforderungen abgedeckt werden [17]. Folgende Pass-Fail-Tests zur Überprüfung der Software wurden durchgeführt, als Testumgebung dient die Software in der Version 1.0 auf einem PC mit einem Windows 10

Betriebssystem, 8GB RAM, und Intel I5 Prozessor.

Manuelles Öffnen und Schließen der einzelnen Kanäle

Beim ersten Pass/Fail Test sollte ein Kanal des Perfusionssystems mithilfe der grafischen Benutzeroberfläche gesteuert werden. Über einen Klick entweder auf die grafische Nachbildung des Perfusionsbehältnisses oder auf die dazugehörige Schaltfläche wurde der Fluss des jeweiligen Kanals gesteuert, wobei sowohl die Aktivierung als auch die Deaktivierung des Kanals korrekt erfolgte und somit der Test als bestanden eingestuft werden kann.

Automatischer Modus mit definierten Schaltzeitpunkten

Der nächste Test dient der Überprüfung des automatischen Modus und dem Hinzufügen automatischer Schaltvorgänge. Dazu wurden drei automatisierte Schaltvorgänge im automatischen Modus durchgeführt und der Durchlauf gestartet. Die Timer wurden dabei mit den korrekten Schaltzeitpunkten und minimaler Abweichung im Millisekunden-Bereich durchgeführt. Aus diesem Grund kann der Test auch als erfolgreich beschrieben werden.

Hinzufügen von Schaltzeitpunkten mit dem gleichen Kanal

Im automatischen Modus wurden dem Versuchsprotokoll Schaltzeitpunkte hinzugefügt, wobei es nicht möglich sein darf, einen darauffolgenden Schaltvorgang den selben Kanal wie den vorherigen Vorgang zuzuweisen. Da im Drop-Down Menü zum Auswählen des zu schaltenden Kanals der vorherige Kanal immer ausgegraut angezeigt wird, kann auch dieser Test als bestanden eingestuft werden.

Exportieren und Importieren von Versuchsprotokollen

Dieser Test dient der Überprüfung des Verhaltens beim Export und Import von Protokollen. Dazu wurden drei Schaltvorgänge sowohl im manuellen als auch im automatischen Modus hinzugefügt und zunächst in eine lokal gespeicherte CSV-Datei gespeichert. Anschließend wurde dieses Protokoll ebenfalls in die lokale SQLite Datenbank exportiert und anschließend wiederum im automatischen Modus importiert und das Protokoll gestartet. Die Schaltvorgänge wurden dabei korrekt exportiert und importiert, der Test verlief somit erfolgreich.

Abbruch eines automatischen Durchlaufs

In diesem Durchgang erfolgte ein Abbruch des automatischen Durchlaufs vor dem letzten geplanten Schaltvorgang. Wichtig war vor allem der Umstand, dass alle Timer korrekt abgebrochen und die dazugehörigen Objekte gelöscht werden. Nach Abbrechen des Protokolldurchlaufs über die Schaltfläche ”Cancel” im automatischen Modus, erfolgte kein definierter Schaltvorgang mehr und die noch nicht abgeschlossenen Timer-Objekte wurden erfolgreich gelöscht, somit kann dieser Test als bestanden eingestuft werden.

Manuelles Schalten beim Verkleinern des Hauptfensters

Dieser Test dient der Überprüfung der grafischen Benutzeroberfläche beim Verkleinern des Fensters. Hier wurde das Hauptfenster so weit verkleinert, dass die vollständige Anzeige der grafischen Nachbildung des Systems nicht mehr vollständig möglich war. In diesem Fall wird die Abbildung nicht mehr dargestellt und nur noch die Knöpfe des Systems angezeigt. Somit ist die Bedienung auch in diesem Fall noch möglich, der Test wird als bestanden angesehen.

Verhalten bei nicht verbundenem Perfusionssystem

Anschließend erfolgte eine Untersuchung des Verhaltens bei nicht verbundenem Perfusionssystem. Dazu wurde zunächst die USB-Verbindung zum System getrennt und Schaltvorgänge durchgeführt. Die Software erkannte die fehlende Verbindung zum System und öffnet anschließend eine Fehlermeldung im Hauptfenster, welches beschreibt, dass keine aktive Verbindung zum System besteht. Somit wird dieser Test als erfolgreich eingestuft.

Verhalten beim Importieren fehlerhafter CSV-Dateien

Ein weiterer Test beinhaltet das Importieren fehlerhafter CSV-Dateien, wobei als Kriterium zum Bestehen des Tests definiert wurde, dass ein Einlesen einer fehlerhaften Datei die Anzeige eines Popups mit einer Fehlermeldung als Resultat hat.

Grundsätzlich existieren mehrere Methoden CSV Dateien aufzubauen oder deren Inhalte zu trennen, wobei der größte Unterschied das verwendete Trennzeichen ist. In diesem Test-Case wurde ein falsches Trennzeichen verwendet und die CSV-Datei als Vorlage in das Programm eingelesen. Die geforderte Antwort der Software ist nicht in den Software Anforderungen weiter definiert, die Software behandelt das Einlesen der fehlerhaften Datei in der Form, dass keine Schaltvorgänge im automatischen Modus importiert werden. Es erscheint jedoch keine Fehlermeldung mit dem Hinweis, dass eine fehlerhafte Datei importiert wurde, somit wird dieser Test als nicht bestanden eingestuft.

Software-Freigabe

Der abschließende Schritt im Softwareentwicklungsprozess stellt die Freigabe der Software dar. In diesem Punkt der Freigabe werden die verbleibenden Fehler dokumentiert und bewertet und somit entschieden, ob der erreichte Stand der Entwicklung freigegeben wird [18]. Wie im Abschnitt "Software-Prüfung" beschrieben, können alle Anforderungen an das Software-System als erfüllt angesehen werden. Beim Import von fehlerhaften CSV-Dateien wurde der durchgeführte Test zwar als nicht bestanden eingestuft, jedoch wird dieses Fehlverhalten als nicht kritisch eingestuft, da in der Dokumentation der Software, der Aufbau und die eingesetzten Trennzeichen der CSV-Datei genau dokumentiert ist. Aus diesem Grund erfolgt die abschließende Freigabe der Software in der Version 1.0. Die mithilfe des Tools Doxygen generierte Dokumentation der Anwendungssoftware ist im Anhang im Abschnitt A.5 ersichtlich.

5. Untersuchung der Gebrauchstauglichkeit

In diesem Teil der Arbeit wird die Gebrauchstauglichkeit des gesamten Sytems untersucht. Zu diesem Zweck wird der gebrauchtauglichkeitsorientierte Entwicklungsprozess, welcher in der Norm EN-62366 dargestellt wird, beschrieben und dokumentiert, in welcher Weise dieser Prozess hier angewandt wurde. Die Untersuchung der Gebrauchstauglichkeit erfolgt wiederum angelehnt an die Norm EN-62366, aufgrund des vorgesehenen Einsatzgebietes als Laborgerät ist die strikte Einhaltung der Norm EN-62366 nicht erforderlich. Aufgrund einfacherer Integrations- und Weiterentwicklungsoptionen wurde jedoch auf Aspekte des in der Norm beschriebenen gebrauchtauglichkeitsorientierten Entwicklungsprozesses Rücksicht genommen.

5.1. Gebrauchstauglichkeitsorientierter Entwicklungsprozess

Mit der Norm EN-62366 existiert ein Standard, welcher die Gebrauchstauglichkeit von Medizinprodukten beschreibt und die Vorgehensweise während der Entwicklung im

Hinblick auf Optimierung der Gebrauchstauglichkeit beschreibt. Somit wird ein Prozess beschrieben, der mit einer Anforderungsanalyse beginnt und mit einem Usability Test mit Anwendern des Produktes abschließt. Dieser abschließende Test dient der Usability Validierung [24].

5.1.1. Spezifikation der Anwendung

Den Beginn des Usability Engineering Prozessses stellt die Thematik der Spezifikation der Anwendung dar. In diesem Punkt wurden zunächst Informationen über die spätere Anwendung des Produktes erfasst, wie beispielsweise vorgesehene Benutzerprofile oder spezielle Gebrauchsbedingungen. Zunächst wurde eine umfangreiche Analyse des Einsatzgebietes durchgeführt und über persönliche Gespräche mit potenziellen Benutzern des Systems Informationen für weitere Schritte erhalten. Es wurden vor allem gewünschte Bedienkonzepte und Funktionen dokumentiert und Systeme, welche bereits im Laborumfeld im Einsatz sind, im Hinblick auf spezifische Vor- und Nachteile analysiert.

5.1.2. Häufig benutzte Funktionen

Der nächste Schritt stellt die Definition der häufig benutzten Funktionen dar. In diesem Schritt, welcher laut Norm EN-62366 im Punkt 5.2 gefordert ist, wurden mithilfe von ersten Entwürfen des Systems und bereits eingesetzten Produkten evaluiert, welche Funktionen typischerweise am öftesten benutzt werden. Das Ergebnis dieser Untersuchung war die Tatsache, dass in den meisten Fällen die Steuerung eines Perfusionssystems bisher über manuelles Aktivieren des gewünschten Kanals und händischen Protokollieren des Schaltzeitpunktes erfolgte. Es sind derzeit zwar Systeme im Einsatz,

welche automatische Ansteuerungen ermöglichen, allerdings ist das Konfigurieren und Erstellen von definierten Schaltzeitpunkten bei diesen derzeit im Einsatz befindlichen Systemen nur sehr komplex möglich und wird somit in der Laborpraxis von den meisten Benutzern nicht angewendet.

Im Zuge der Evaluierung des Systems wurden folgende Funktionen als häufig benutzt definiert:

- Manuelles Aktivieren und Deaktivieren von Kanälen
- Hinzufügen neuer Bezeichnungen verwendeter Substanzen
- Zuweisen der Substanzen zu den einzelnen Kanälen
- Automatischer Durchlauf von vorher erstellten Protokollen

5.1.3. Ermittlung sicherheits-bezogener Merkmale

In diesem Punkt des Entwicklungsprozesses müssten sicherheitsbezogene Merkmale mit Schwerpunkt auf die Gebrauchstauglichkeit ermittelt werden. In Zusammenhang mit der Benutzung des Perfusionssystems ergeben sich Risiken durch Einklemmen von Gliedmaßen bei der Höhenverstellung des Gehäuses und durch Umkippen des gesamten Systems, wobei keine resultierenden Risiken in Zusammenhang mit den häufig benutzten Funktionen existieren.

5.1.4. Hauptbedienfunktionen

In der Norm EN-62366 ist unter dem Punkt 5.4 die Definition der Hauptbedienfunktionen festgelegt. Diese Festlegung der Hauptbedienfunktionen sollte hierbei häufig benutzte Funktionen und sicherheitsbezogene Funktionen umfassen. Unter diesem

Punkt wurden Funktionen definiert, welche nicht als häufig benutzt erachtet werden, jedoch ein gewisses Risikopotential besitzen. Die in diesem Zusammenhang bei der Benutzung des Perfusionsystems resultierenden Risiken lauten wie folgt:

- Einklemmen von Fingern beim Höhenverstellen des Systems
- Umkippen des Systems bei Zug an den Schläuchen

5.1.5. Spezifikation der Gebrauchstauglichkeit

Aufbauend auf die Definition der Risiken und Bedienfunktionen wurden Use-Szenarien erstellt. Diese Szenarien dienten vor allem der erleichterten Entwicklung des Systems, um mögliche Anwendungsfälle so komfortabel wie möglich lösen zu können. Folgende Use-Szenarien wurden dabei definiert:

Szenario: Durchführen manueller Schaltvorgänge im Zuge von Live-Cell Imaging Messungen

1. Vorbereiten der Pufferlösungen
2. Hinzufügen der Bezeichnungen der Pufferlösungen in die Datenbank, falls nicht bereits vorhanden
3. Zuweisen der Pufferbezeichnungen zu den jeweiligen Kanälen in der Software
4. Aufbringen der Zellen auf den Objektträger
5. Starten der Imaging-Software
6. Durchführen von Schaltvorgängen über die grafische Benutzeroberfläche

Szenario: Erstellen und Starten geplanter Schaltvorgänge im automatischen Modus

1. Vorbereiten der Pufferlösungen

2. Hinzufügen der Bezeichnungen der Pufferlösungen in die Datenbank, falls nicht bereits vorhanden
3. Zuweisen der Pufferbezeichnungen zu den jeweiligen Kanälen in der Software
4. Sequentielles Hinzufügen von Schaltvorgängen im automatischen Modus des Anwendungsprogramms
5. Definieren der Dauer des jeweiligen Schaltvorganges
6. Aufbringen der Zellen auf den Objektträger
7. Starten der Imaging-Software
8. Starten des im automatischen Modus definierten Versuchsablauf

5.1.6. Validierungs-Plan für Gebrauchstauglichkeit

Unter diesem Punkt werden qualitative und quantitative Methoden zur Validierung der Gebrauchstauglichkeit zusammengefasst. Im Zuge der Validierung der Gebrauchstauglichkeit existieren unterschiedliche Methodiken wie beispielsweise Interviews mit Benutzern des Systems oder die Auswertung von Fragebögen. Da im Zuge dieser Arbeit der Entwicklungsprozess im Vordergrund steht, ist die Ausführung der Validierung mithilfe von standardisierten Möglichkeiten im Zuge einer zukünftigen Arbeit geplant.

5.1.7. Gestaltung und technische Umsetzung der Benutzer-Produkt-Schnittstelle

Der Punkt der Gestaltung und technischen Umsetzung ist erst im Kapitel 5.7 der Norm EN-62366 gefordert. Aufgrund der Tatsache, dass das System von Grund auf neu entwickelt wurde, konnte ohne weitere Berücksichtigung bereits verwendeter

Technologien das neue System gestaltet werden. Durch Gespräche mit Endanwender, welche während der gesamten Entwicklungsdauer geführt wurden, konnte das Layout sehr gut an die unterschiedlichen Benutzungsszenarien angepasst werden. Der genaue Ablauf der Entwicklung der Software und Hardware ist in den Kapiteln 2 und 3 beschrieben.

5.1.8. Verifizierung und Validierung der Gebrauchstauglichkeit

Diese Kapitel der Norm beschreiben die Usability Evaluierungen, welche während der Entwicklung stattfanden. Im Zusammenhang mit der Entwicklung des Gesamtsystems wurde zunächst eine erste formative Evaluierung der Usability gemeinsam mit fünf potentiellen Anwendern des Systems durchgeführt. Diesen Anwendern wurden vorläufige Entwürfe der grafischen Benutzeroberfläche vorgelegt und der erste Eindruck dieses Entwurfs ausgewertet. Eine Erkenntnis dieser Befragung war die Schwierigkeit den aktuellen Status des jeweiligen Kanals sichtbar darzustellen. Aufgrund dieses Punktes wurde die Struktur des Systemes über grafische Elemente innerhalb der Benutzeroberfläche nachgebildet und somit gewährleistet, dass offene und geschlossene Kanäle auf einem Blick ersichtlich sind. Außerdem wurden mehrere Punkte bezüglich der Platzierung von Bedienelementen und der Konfiguration des Systems diskutiert und im Zuge der Erstellung mehrere Iterationen implementiert.

6. Gesamtaufbau des Perfusionssystems

In diesem Kapitel ist der gesamte Aufbau des Systems als Hauptergebnis der Arbeit dargestellt, sowohl in Form eines generierten CAD-Modells, als auch in Form des fertigen Prototypen. Zunächst erfolgt die Darstellung des fertigen prototypischen Systems in Form eines CAD-Modells, wobei die einzelnen Komponenten im Kapitel "Hardware-Entwicklung" beschrieben sind. Detaillierte Zeichnungen der einzelnen Komponenten sind im Anhang im Abschnitt A.1 abgebildet.

Ein fertiger Prototyp des Gesamtsystems wurde exemplarisch gefertigt und ist bereits unter Laborbedingungen an einem Fluoreszenzmikroskop im Einsatz. Dieser Prototyp sollte als Grundlage für Weiterentwicklungen oder mögliche Integrationen in bestehende Software- oder Hardware-Systeme dienen. Außerdem geschieht eine laufende Evaluation des Gesamtsystems im Einsatz, um in Zukunft weitere Optimierungen der Benutzeroberfläche durchzuführen.

6.1. CAD Modell des prototypischen Gesamtsystems

Die in den Abschnitten der Hardware Entwicklung beschriebenen Komponenten wurden mithilfe von CAD Software zu einem Gesamtsystem zusammengefügt. Das Ergebnis dieser Modellierung des Gesamtsystems ist in Abbildung 6.1 abgebildet, jedoch ohne dem Modell der Schlauchzusammenführung, da diese an den Schläuchen angebracht wird und die einzelnen Schläuche im Modell nicht eingezeichnet sind.

Abbildung 6.1.: Gesamtmodell des Perfusionssystems

6.2. Aufgebauter Prototyp in Verwendung

In Abbildung 6.2 ist der gefertigte Prototyp des entwickelten Perfusionssystems im Einsatz bei einem Fluoreszenzmikroskop dargestellt. Mithilfe des auf der linken Seite des Tisches befindlichen PCs wird die Ansteuerungssoftware des Perfusionssystem ausgeführt und Schaltvorgänge vorgenommen. Hier ist auch die sehr gute grafische Kontrolle der offenen Kanäle, welche mithilfe der entwickelten Ansteuerungssoftware möglich ist, sichtbar.

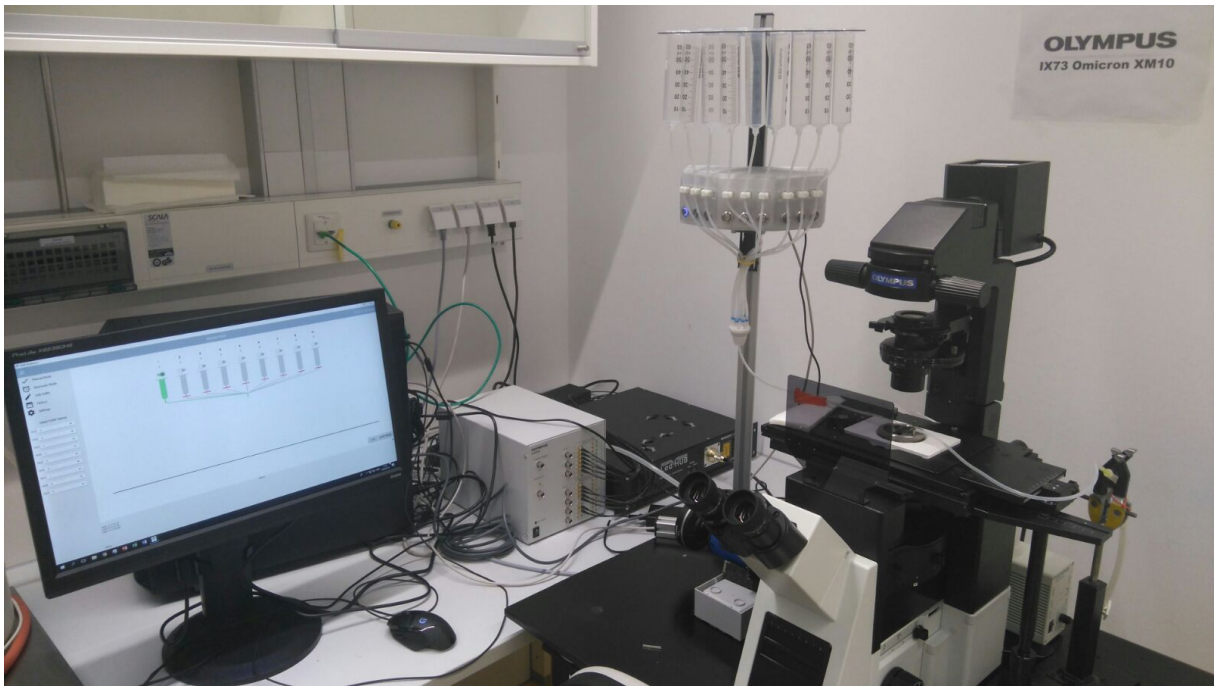


Abbildung 6.2.: Im Einsatz befindlicher Prototyp des entwickelten Perfusionssystems

Durch die Bedienung der Software können alle wichtigen Schaltvorgänge des Perfusionssystems direkt am PC durchgeführt werden. Räume, in denen Fluoreszenzmikroskopische Untersuchungen stattfinden, werden für die Dauer der Untersuchung meist komplett abgedunkelt, um Störlicht zu vermeiden. Dies erschwert jedoch die manuelle Bedienung etwaiger Taster oder Knöpfe an Geräten.

Durch den Einsatz der Anwendungssoftware ist der Vorteil der softwarebasierten Bedienung ersichtlich, da in den meisten Fällen die Bedienung des Mikroskops mithilfe einer Steuersoftware durchgeführt wird und somit die Möglichkeit besteht, die Anwendungssoftware des Perfusionssystems im Hintergrund oder auf einer zweiten Bildschirmoberfläche auszuführen.

7. Anwendungsbeispiele

In diesem Kapitel sollten mithilfe durchgeführter Live Cell Imaging Untersuchungen mögliche Anwendungsbeispiele des Systemes diskutiert werden. Ein wichtiger Punkt ist die generelle Befürwortung von Systemen, welche Perfusion im Zuge von Imaging Anwendungen ermöglichen, da einige Anwender von Live Cell Imaging Systemen manuelles Pipettieren zum Einbringen von Substanzen und Pufferlösungen einsetzen. Gerade durch manuelles Pipettieren können Ergebnisse verfälscht werden, womit die Reproduzierbarkeit von Messergebnissen erschwert wird.

7.1. Reproduzierbarkeit von Messungen der intrazellulären Kalzium-Konzentration

Zunächst wurde mithilfe von Live Cell Imaging Untersuchungen die intrazelluläre Kalzium-Konzentration ratiometrisch bestimmt. Für diese Zwecke wurde die fluoreszierende, Kalzium-sensitive, Substanz Fura-2 eingesetzt. Der Sensor Fura-2 kann in Form des membrangängigen Derivats Fura-2AM durch passive Diffusion in eine Zelle eingebracht werden, wobei anschließend die Acetoxymethylester(AM)-Gruppe durch endogene Esterasen abgespalten wird [25]. Durch diesen Abspaltvorgang wird die Kalziumionen (Ca^{2+})-abhängige Fluoreszenzeigenschaft von Fura-2 aktiviert und verliert

aufgrund des Abspaltvorgangs der AM-Gruppe seine membrangängige Eigenschaft, weshalb Fura-2 in der Zelle akkumuliert.

Durch die Beladung von HeLa-Zellen mit Fura-2 kann nun die intrazelluläre Ca^{2+} -Konzentration $([Ca^{2+}]_i)$ auf Stimulation der Zellen mit extrazellulärem Adenosin 5'-triphosphat (ATP) gemessen werden. In diesem Durchlauf wird die extrazelluläre Konzentration von ATP über zwei unterschiedliche Methoden verändert und die resultierende $([Ca^{2+}]_i)$ untersucht.

Die ersten Durchläufe umfassten jene Messungen, bei denen ATP manuell mittels Pipette appliziert wurde. Es wurden drei unabhängige Messungen durchgeführt und die Resultate dieser verglichen (Abbildung 7.1). Die Zugabe von ATP in einer finalen Konzentration von 100 μM resultierte bei zwei Messungen in einem rasanten Anstieg von $([Ca^{2+}]_i)$, wobei bei einer Messung der Anstieg zeitverzögert erfolgte. Der schwarze Pfeil in Abbildung 7.1 markiert den Zeitpunkt, zu welchem ATP hinzugefügt wurde.

Es ist ersichtlich, dass nach dem erstmaligen Hinzufügen von ATP ein starker Anstieg der intrazellulären Kalzium-Konzentration erfolgt, und dass nach einer wiederholten Zugabe von ATP keine erneute Erhöhung von $([Ca^{2+}]_i)$ ersichtlich ist. Dies erklärt sich daraus, dass beigefügtes ATP nicht mehr von den Zellen entfernt werden kann.

Bei der zweiten Methode wurde das entwickelte Perfusionssystem im automatischen Modus zur Stimulation von Zellen mit ATP verwendet. Dazu wurde der in Abbildung 7.2 dargestellte Ablauf in der Anwendungssoftware des Perfusionssystem erstellt. In den ersten 60 Sekunden erfolgt eine Perfusion der Zellen mit einer physiologischen Lösung, darauf folgend wird für die Dauer von 60 Sekunden Channel2 geöffnet, welcher zusätzlich 100 μM ATP beinhaltet. Nach einer Gesamtdauer von 180 Sekunden wird wieder Channel1 geöffnet, um den ATP Stimulus mittels Perfusion auszuwaschen.

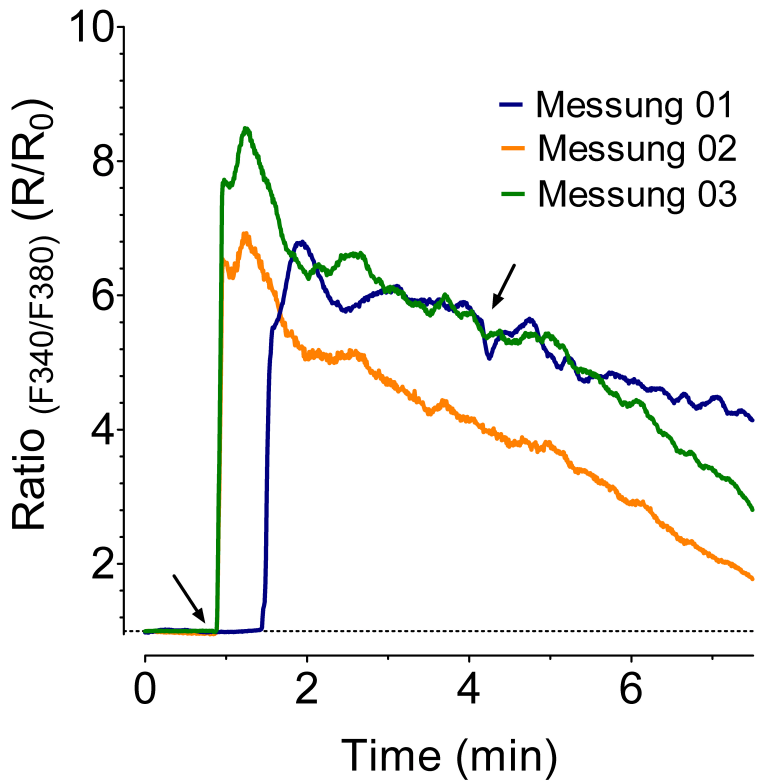


Abbildung 7.1.: Messung der intrazellulären Kalzium-Konzentration bei Zugabe von ATP mittels manuellem Pipettieren

In Analogie zum ersten Ansatz wurde auch in diesen Experimenten ein weiterer ATP Stimulus nach einer Versuchsdauer von 240 Sekunden gesetzt, was unter Verwendung des Perfusionssystems in einem erneuten Anstieg von $([Ca^{2+}]_i)$ resultiert, welcher wiederum reversibel ist.

Es wurden in Summe sechs Messungen durchgeführt, wobei alle Messungen unter zeitgleicher Zugabe von ATP erfolgten. In Abbildung 7.3 sind die Ergebnisse der Fura 2-Messungen mit automatisierter Perfusion ersichtlich. Auf der Abszisse ist die Zeit in Minuten aufgetragen, wobei die gesamte Messdauer acht Minuten betrug. Da Fura 2 ein ratiometrischer Sensor ist, kann mittels zweier Lichtquellen, welche

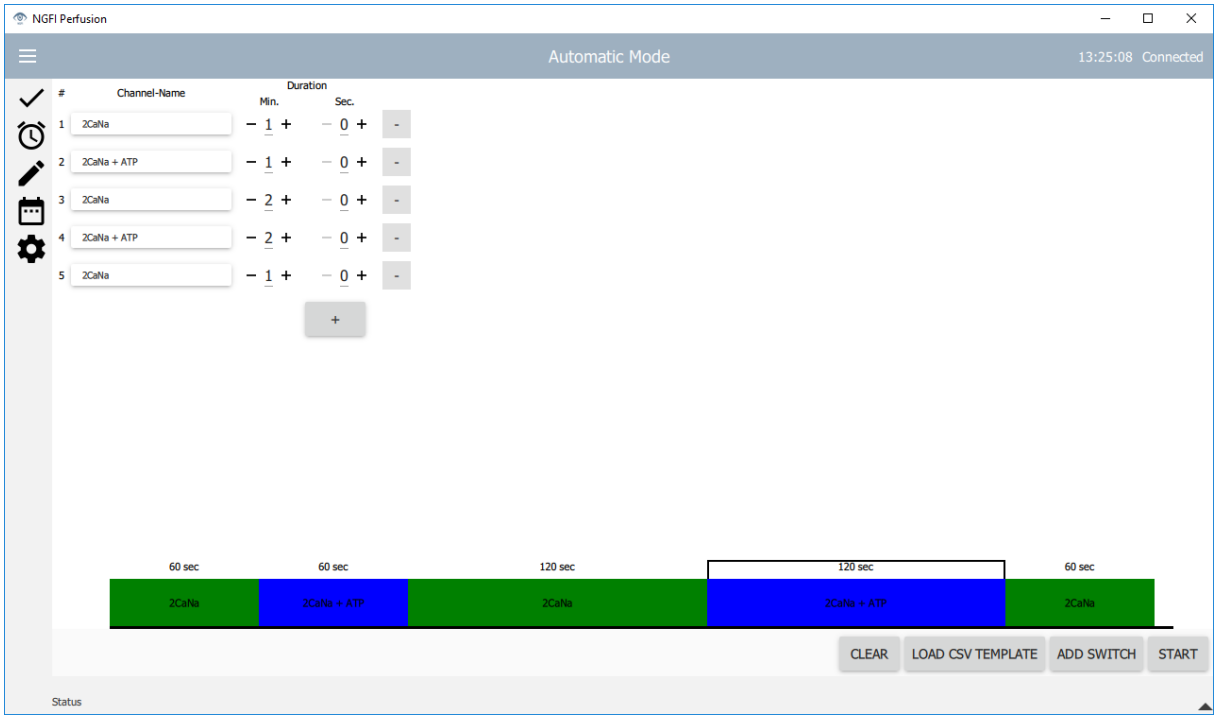


Abbildung 7.2.: Automatischer Modus der Anwendungssoftware bei der Messung der intrazellulären Kalzium-Konzentration

unterschiedliche Wellenlängen aufweisen, eine wechselnde Anregung erfolgen. Im durchgeführten Versuch erfolgte die Anregung abwechselnd mit einer Wellenlänge von 340 und 380 nm, wobei anschließend die Emissionsintensität bei 510 nm der unterschiedlichen Anregungen in Verhältnis gebracht wird.

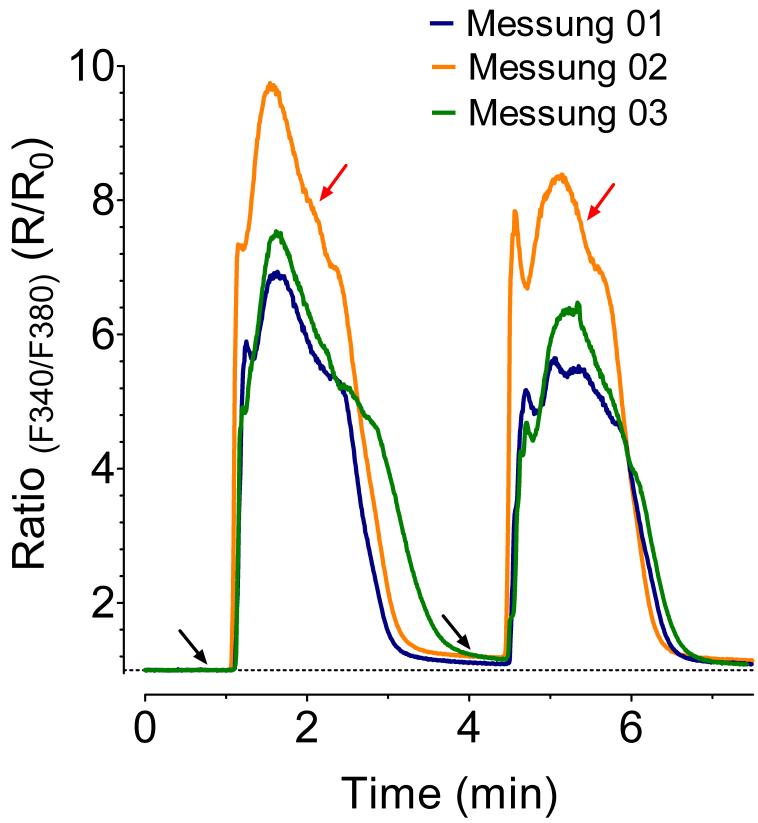


Abbildung 7.3.: Messung der intrazellulären Kalzium-Konzentration bei Zugabe von ATP mittels Perfusionssystem

7.2. Reproduzierbarkeit von Messungen der Kalium-Konzentration in vitro

Bei der zweiten Untersuchung wurde eine Messung der Kalium-Konzentration ([K+]) mittels eines K+ sensitiven, fluoreszierenden Sensors durchgeführt und die unterschiedlichen Methoden der Beigabe von Substanzen durch Perfusion oder Pipettieren verglichen. Zur Messung der [K+] wurde ein genetisch codierter Fluoreszenz Sensor verwendet, welcher für die Detektion von Kalium *in vitro* und *in vivo* eingesetzt werden kann [26]. Das Messprinzip des Sensors beruht auf einer Änderung der Verhältnisses

der Emissionsintensität zweier Wellenlängen (FRET/CFP), welches direkt proportional zur K+ Konzentration ist. Zunächst wurden sechs Proben vorbereitet, welchen den fluoreszierenden Sensor in Form eines aufgereinigten Proteins enthalten. Das Protein wurde in Agarose immobilisiert, weshalb dieses auch in Lösung beziehungsweise unter Perfusion stationär am verwendeten Objektträger verblieb.

Zur Abhandlung des ersten Durchlaufs wurde, ähnlich wie bereits bei den Messungen der $[Ca^{2+}]_i$, eine manuelle Zufuhr einer K+ Lösung zu einer finalen Konzentration von 5 mM durchgeführt, welches 60 Sekunden nach Beginn der Messung hinzugefügt wurde (schwarzer Pfeil in Abbildung 7.4). Bei Messung zwei und drei der durchgeführten Messungen führte die manuelle Zufuhr zu einem raschen Anstieg des gemessenen [K+] (Abbildung 7.4). Trotz zeitgleicher Zufuhr von K+ auch bei der ersten Messung dauerte es bei dieser wesentlich länger, bis ein signifikanter Anstieg messbar wurde.

Um den gleichen Effekt nochmals in umgekehrter Reihenfolge, nicht K+ Zufuhr sondern K+ Pufferung zu testen, wurde nach zirka 180 Sekunden nach K+ Zufuhr Poly-Natrium-Styrensulfonat (PNSS) zugeführt, welches eine höhere Affinität für K+ als der K+ sensitive Sensor besitzt (roter Pfeil). Interessanterweise waren die daraus erhaltenen Resultate wesentlich heterogener als jene der K+ Zufuhr. So wurde beispielsweise in Messung 1 keine K+ Pufferung festgestellt, in Messung 2 eine sehr langsame und verspätete, und Messung 3 musste aufgrund eines Fehlers bei der Applikation gestoppt werden, was für eine Fehleranfälligkeit der manuellen Zugabe spricht.

In Abbildung 7.5 sind die Intensitäten des K+ sensitiven Proteins in Abhängigkeit von der Zeit aufgetragen. Der eingefügte schwarze Pfeil markiert die Zugabe eines Puffers mit 5 mM K+ auf den Objektträger mittels Perfusionssystem, der rote Pfeil markiert erneut den Zeitpunkt an dem PNSS hinzugegeben wurde. Die drei Messungen zur Bestimmung der Kalium-Konzentration weisen einen ähnlichen Verlauf des Ratio Signal

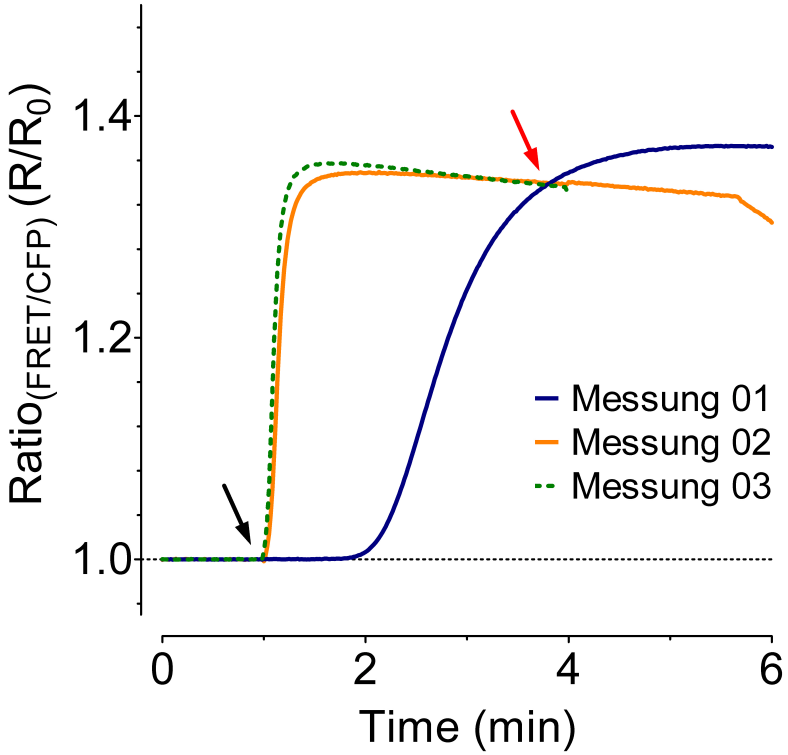


Abbildung 7.4.: Messung der Kalium-Konzentration bei Zugabe von Kalium mittels manuellem Pipettieren

Anstiegs auf, bis ein konstanter K+ Level im Ansatz erreicht wurde. Hierbei zeigen alle drei unabhängig voneinander durchgeführten Messungen ein sehr homogenes und gleichzeitig auftretendes Signal. Ein sehr ähnlicher Kurvenverlauf der drei Messungen zeigte sich auch nach Zugabe von PNSS, welches in einem rasant absteigenden K+ Signal des Sensors resultierte.

Das in der Anwendungssoftware erstellte Protokoll ist in Abbildung 7.6 dargestellt. Ähnlich wie bereits bei den durchgeführten $[Ca^{2+}]_i$ Messungen, zeigte sich auch anhand der K+ Messungen, welche mithilfe eines K+ sensitiven, fluoreszierenden Sensors durchgeführt wurden, ein deutlicher Unterschied zwischen Messungen, die manu-

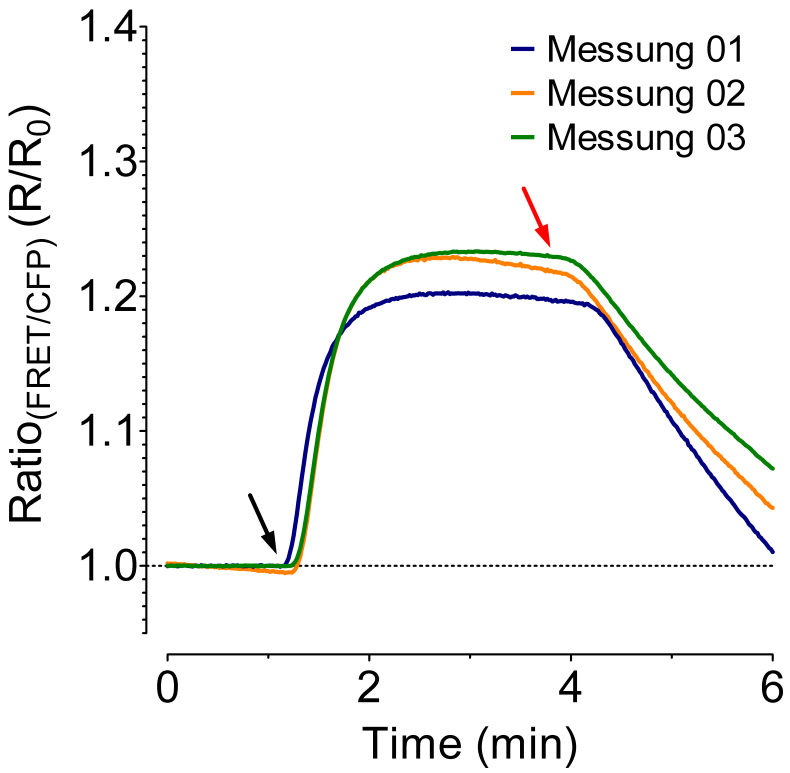


Abbildung 7.5.: Messung der Kalium-Konzentration bei Zugabe von Kalium mittels Perfusionssystem

eller K+ Zufuhr oder K+ Zufuhr mittels Perfusionssystems unterlagen. Die Vorteile des Perfusionssystems und dessen Reproduzierbarkeit bei unabhängig voneinander durchgeführten Messungen wird bei dem Vergleich der beiden Methoden deutlich ersichtlich.

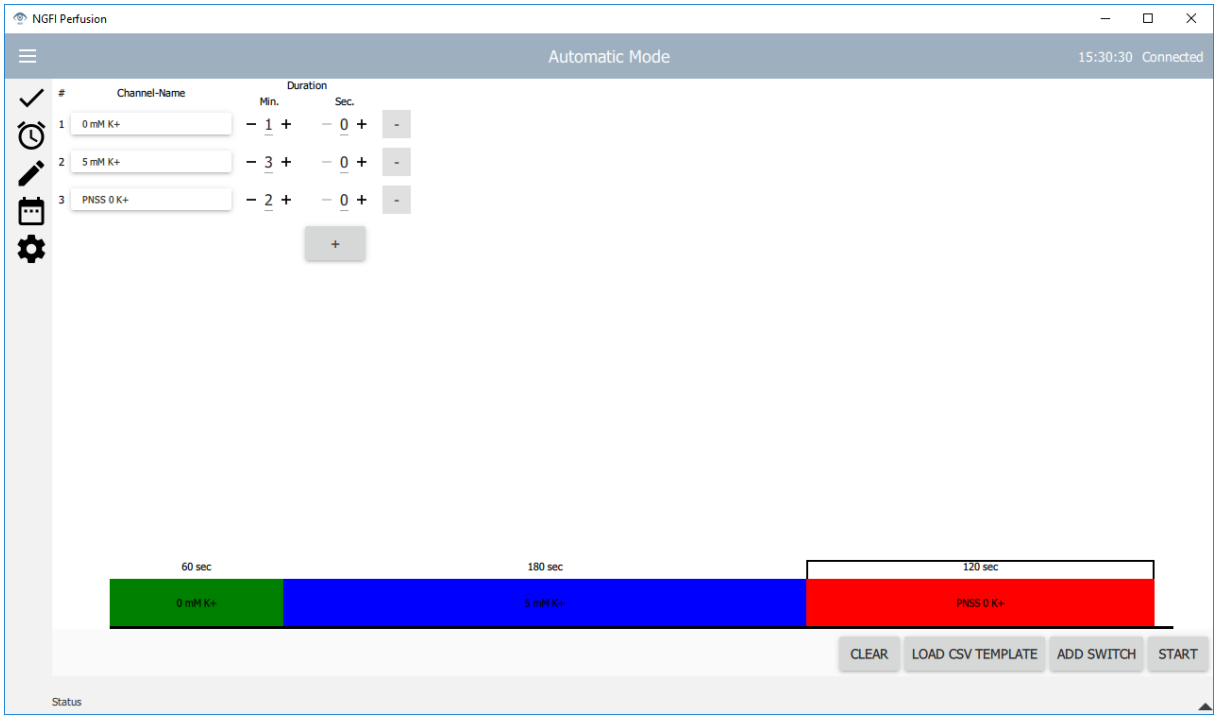


Abbildung 7.6.: Automatischer Modus der Anwendungssoftware bei der Messung der Kalium-Konzentration

8. Diskussion und Schlussfolgerung

Das Ergebnis dieser Masterarbeit stellt ein fertig entwickeltes Perfusionssystem dar, welches automatisiert und komfortabel über eine anwenderfreundliche Software bedient werden kann. In diesem Abschnitt sollten die Vorteile und Nachteile des Systems gegenübergestellt und mögliche Verbesserungen und Erweiterungen beschrieben werden. Diese weiteren Aspekte sind grob in entwicklungsspezifische und anwendungsspezifische Aspekte unterteilt.

8.1. Entwicklungsspezifische Aspekte

Das entwickelte Perfusionssystem hat den großen Vorteil, dass der gesamte Aufbau kompakt und stabil gehalten wurde. Somit ist es möglich, das System auf einen Labortisch zu platzieren, ohne viel Platz in Anspruch zu nehmen. Eine mögliche Erweiterung wäre der Einbau einer motorisierten Höhenverstellbarkeit, um die Flussrate komfortabel über die Anwendungssoftware regeln zu können. Als Motor kann hier beispielsweise ein Schrittmotor dienen, welcher über eine Gewindestange mit dem Gehäuse verbunden ist. Durch den Fluss der Pufferlösungen verringert sich der Füllstand im Inneren der Pufferbehältnisse, somit auch der hydrostatische Druck und

die resultierende Flussgeschwindigkeit, was zu einer füllstandabhängigen Beeinflussung der Zellreaktionen führt. Diese Auswirkung könnte durch eine automatisierte Veränderung der Höhe der Pufferbehältnisse während des Untersuchungszeitraums ausgeglichen werden.

Um eine Integration und Synchronisation mit anderen Laborgeräten oder Steuerungen zu ermöglichen, wäre als mögliche Weiterentwicklung auch eine zusätzliche externe Ansteuerungsmöglichkeit mittels TTL-Signale möglich. Durch diese Weiterentwicklung könnten unterschiedliche Systeme für Live Cell Imaging Anwendungen zeitgesteuert und synchron angesteuert werden. Lösungen, welche diese mögliche synchrone Ansteuerung bieten, sind bereits im Einsatz [27].

Durch eine Kombination des entwickelten Perfusionssystems mit einer Rollerpumpe ist es möglich, durch die Synchronisation der Pumpe und des Schaltens der Klemmventile die Genauigkeit der Flussgeschwindigkeit zu erhöhen. Es würde außerdem die Möglichkeit bestehen, die Flussgeschwindigkeit der zum Einsatz kommenden Substanzen komfortabel zu regulieren. Dazu ist eine Erweiterung der elektronischen Schaltung zur Ansteuerung der Pumpe und ein Hinzufügen weiterer Funktionen zur Anwendungssoftware notwendig.

Eine weitere Möglichkeit der Ansteuerung wäre ein mögliches Schalten der einzelnen Kanäle durch manuelles Betätigen der an der Vorderseite des Gehäuses angebrachten Taster. Um diese Funktionalität einzubauen, ist es notwendig die elektronische Schaltung zu erweitern und die Kontakte der vorhandenen Taster mit den Eingängen des Mikrocontrollers zu verbinden. Außerdem ist es erforderlich, dass alle Schaltvorgänge, welche über Druck des jeweiligen Tasters durchgeführt werden, zur Anwendungssoftware übermittelt werden. Durch diese Erweiterung könnte sich jedoch der Vorteil ergeben, dass das System komplett ohne grafische Benutzeroberfläche beziehungsweise im Stand-Alone Betrieb bedienbar wäre, jedoch würden die Vorteile der Softwarelösung,

wie die Erstellung automatisierter Durchläufe und Dokumentation der Schaltvorgänge, nicht mehr gegeben sein.

Für eine adäquatere Fertigung der elektronischen Schaltung ist es ratsam, ein Re-Design der entwickelten Platine durchzuführen. Es könnte die Entwicklung einer Mehrlagigen Platine durchgeführt werden, bei der eine Komponente mit bereits vorhandenen USB-Controller zum Einsatz kommt, beispielsweise ein Mikrocontroller der Serie Atmega32u4 [28]. Durch Verwendung von SMD Bauteilen und mehreren Signallagen, ist es möglich, ohne den Einsatz einer zusätzlichen Prototyping-Plattform, die Funktionalität mithilfe maschineller Fertigung und Bestückung einer Platine zu gewährleisten.

8.2. Anwendungsspezifische Aspekte

Durch den Einsatz des automatisierten Perfusionssystem ergibt sich der Vorteil der Austauschbarkeit der geplanten Messungen und Versuche und es besteht die Möglichkeit diese Vorgänge als Vorlage zu speichern. Bei dem im Zuge dieser Arbeit entwickelten Systems ist es außerdem möglich, die automatisierten Durchläufe eines Versuchs mit Kommentaren zu versehen und sowohl in die interne Datenbank zu speichern, als auch auf Dateiebene zu exportieren. Durch diese Möglichkeit können Schaltvorgänge und Bezeichnungen der verwendeten Pufferlösungen archiviert und gemeinsam mit den Messergebnissen extern abgespeichert werden. Somit erhöht sich zum einen die Verfügbarkeit der Daten und es wird eine Einhaltung der ”Guten Laborpraxis” in Hinblick auf die Dokumentation ermöglicht. Die OECD schreibt vor allem Schritte zur Aufbewahrung und Dokumentation der Versuchsparameter vor, wobei ein Schwerpunkt auf die Integrität und der Abspeicherung der relevanten Daten liegt. Außerdem

ist es erforderlich, dass Pläne für die Sicherung der Daten und die Wahrung der Datenintegrität und des Datenzugriffs erstellt werden [29].

Das entwickelte System erleichtert das Einhalten dieser angeführten Punkte der Datensicherheit, da die automatisierten Durchläufe sowohl in der internen Datenbank, als auch in eine exportierbare Datei gespeichert werden können. Somit besteht die Möglichkeit, einzelne dokumentierte Versuche extern abzulegen und die Wahrscheinlichkeit eines möglichen Datenverlusts somit zu reduzieren. Beim Einsatz konventioneller Systeme wurde die Dokumentation der Schaltvorgänge und der Pufferlösungen meist händisch vorgenommen. Durch die entwickelte Anwendungssoftware kann ein Standardvorgehen (SOP) definiert werden. Diese Beschreibung des Standardvorgehens kann den grundlegenden Ablauf bei der Nutzung des Perfusionssystems definieren, ausgehend von der Benennung der einzelnen Kanäle, zur Ausführung und anschließenden Abspeicherung der automatisierten Durchläufe. Die Einhaltung dementsprechender Standardvorgehen führt zu einer einheitlichen Dokumentation der Versuche und der zum Einsatz kommenden Substanzen, was zu einer Verbesserung der Nachvollziehbarkeit von Messergebnissen führt.

Literatur

[1] *The American Heritage Medical Dictionary*. Houghton Mifflin Harcourt (2004).

[2] Baker, M.: »Cellular imaging: Taking a long, hard look«. In: *Nature* (2010).

[3] Stephens, D.: »Light Microscopy Techniques for Live Cell Imaging«. In: *Science* 300 (2003).

[4] Ettinger, A., Wittmann, T.: »Fluorescence Live Cell Imaging«. In: *Methods in Cell Biology* (2014).

[5] Gries, S.: *Grundlagen der Infusionstechnik*. URL: `http://www.cbg-mittelhessen.de/resources/DL+Grundlagen+der+Infusionstechnik.pdf` (besucht am 22. 03. 2018).

[6] Backhaus, C.: *Usability-Engineering in der Medizintechnik*. Springer (2010).

[7] *EN ISO 62366-1 Medizinprodukte, Anwendung der Gebrauchstauglichkeit auf Medizinprodukte*. (2008).

[8] Gebhardt, A., Hötter, J.-S.: *Additive Manufacturing - 3D Printing for Prototyping and Manufacturing*. Carl Hanser Verlag (2016).

[9] Lienig, J., Brümmer, H.: *Elektronische Gerätetechnik - Grundlagen für das Entwickeln elektronischer Baugruppen und Geräte*. Springer Vieweg (2014).

[10] *BMT Fluid Control Solutions GmbH, PS / PSK Serie - Schlauchquetschventil*. URL: `https://www.pumpen-ventile.at/schlauchquetschventile-ps-psk-serie/` (besucht am 22. 03. 2018).

[11] Lockridge, J.: *Reap the Benefits of Economizers for Solenoid/Relay Drivers*. URL: http://www.electronicdesign.com/power/reap-benefits-economizers-solenoidrelay-drivers (besucht am 22. 03. 2018).

[12] Grace, T.: *Programming and Interfacing Atmel AVR Mikrocontrollers*. Cengage Learning PTR (2016).

[13] *EN ISO 61010-1 Sicherheitsbestimmungen für elektrische Mess-,Steuer-, Regel- und Laborgeräte - Teil 1: Allgemeine Anforderungen*. (2010).

[14] Scherz, P., Monk, S.: *Practical Electronics for Inventors*. McGraw-Hill Education (2016).

[15] Horrowitz, P., Hill, W.: *The Art of Electronics*. 3. Aufl. McGraw-Hill Education (2015).

[16] Leitgeb, N.: *Sicherheit von Medizingeräten Recht-Risiko-Chancen*. 2. Aufl. Springer Vieweg (2015).

[17] *EN ISO 62304 Medizingeräte-Software - Software-Lebenszyklus-Prozesse*. (2006).

[18] Johner, C.: *Basiswissen Medizinische software : aus- und weiterbildung zum certified Professional für medical software*. dpunkt.verlag (2015).

[19] Rust, P.: »Creation of an IEC 62304 compliant software development plan«. In: *Journal of software : evolution and process* 28.11 (Nov. 2016). An optional note, S. 1005–1010.

[20] Lischner, R.: *Exploring C++ 11 Problems and Solutions Handbook*. Apress L. P (2014).

[21] Eilebrecht, K., Starke, G.: *Patterns kompakt - Entwurfsmuster für effektive Software-Entwicklung*. 3. Aufl. Spektrum Akademischer Verlag Heidelberg (2010).

[22] Newman, C.: *SQLite*. Sams (2004).

[23] Guillaume, L., Penea, R.: *Mastering Qt 5*. Packt Publishing (2016).

[24] Fischer, H. , et al.: *Mensch und Computer 2015 - Usability Professionals*. De Gruyter (2015).

[25] Grynkiewicz, G.: »A new generation of Ca2+ indicators with greatly improved fluorescence properties«. In: *Journal of Biological Chemistry* 260.6 (1985).

[26] Bischof, H. , et al.: »Novel genetically encoded fluorescent probes enable real-time detection of potassium in vitro and in vivo«. In: *Nature Communications* (2017).

[27] Olympus: *U-RTC / U-RTCE Realtime-Controller*. URL: https://www.olympus-lifescience.com/de/advanced-imaging-solutions/rtc-rtce/ (besucht am 22. 03. 2018).

[28] Inc., M. T.: *Datasheet Atmega32u4*. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf (besucht am 22. 03. 2018).

[29] *OECD Series on Principles of Good Laboratory Practice (GLP) and Compliance Monitoring - No. 17: Application of GLP Principles to Computerised Systems*. (2016).

[30] *Arduino AG, Arduino Leonardo Schematic*. URL: https://www.arduino.cc/en/uploads/Main/arduino-leonardo-schematic_3b.pdf (besucht am 22. 03. 2018).

[31] *Arduino AG, Arduino Uno Schematic*. URL: https://www.arduino.cc/en/uploads/Main/arduino-uno-schematic.pdf (besucht am 22. 03. 2018).

A. Anhang

A.1. CAD Zeichnungen

A.1.1. Gehäuse-Deckel

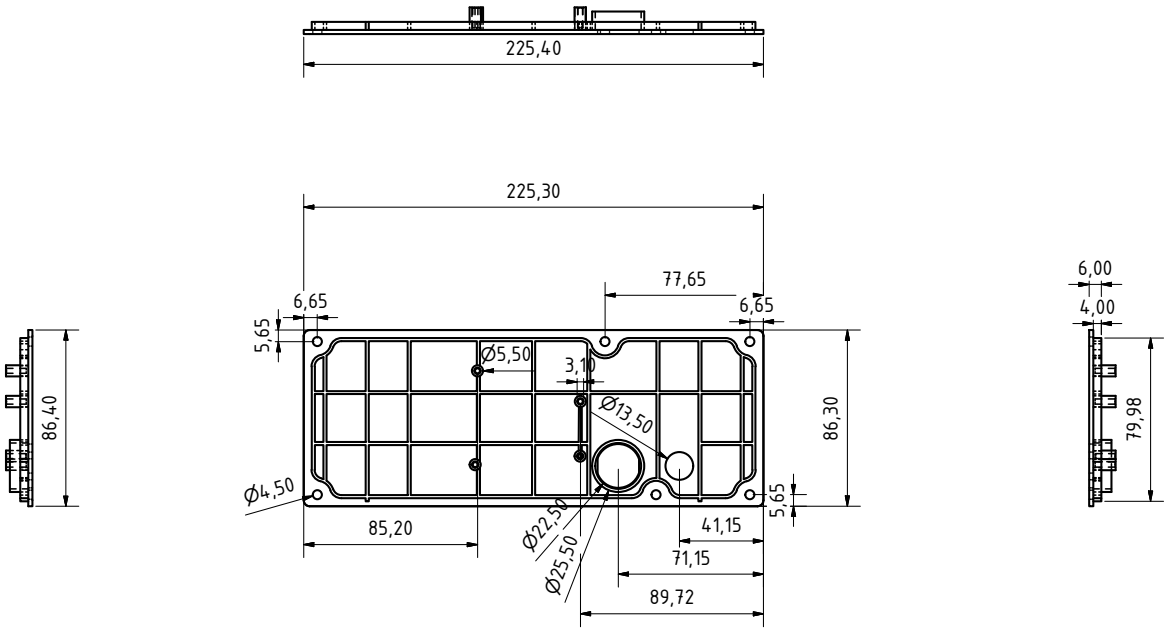


Abbildung A.1.: Bemaßte Zeichnung des Gehäuse-Deckels

A.1.2. Gehäuse

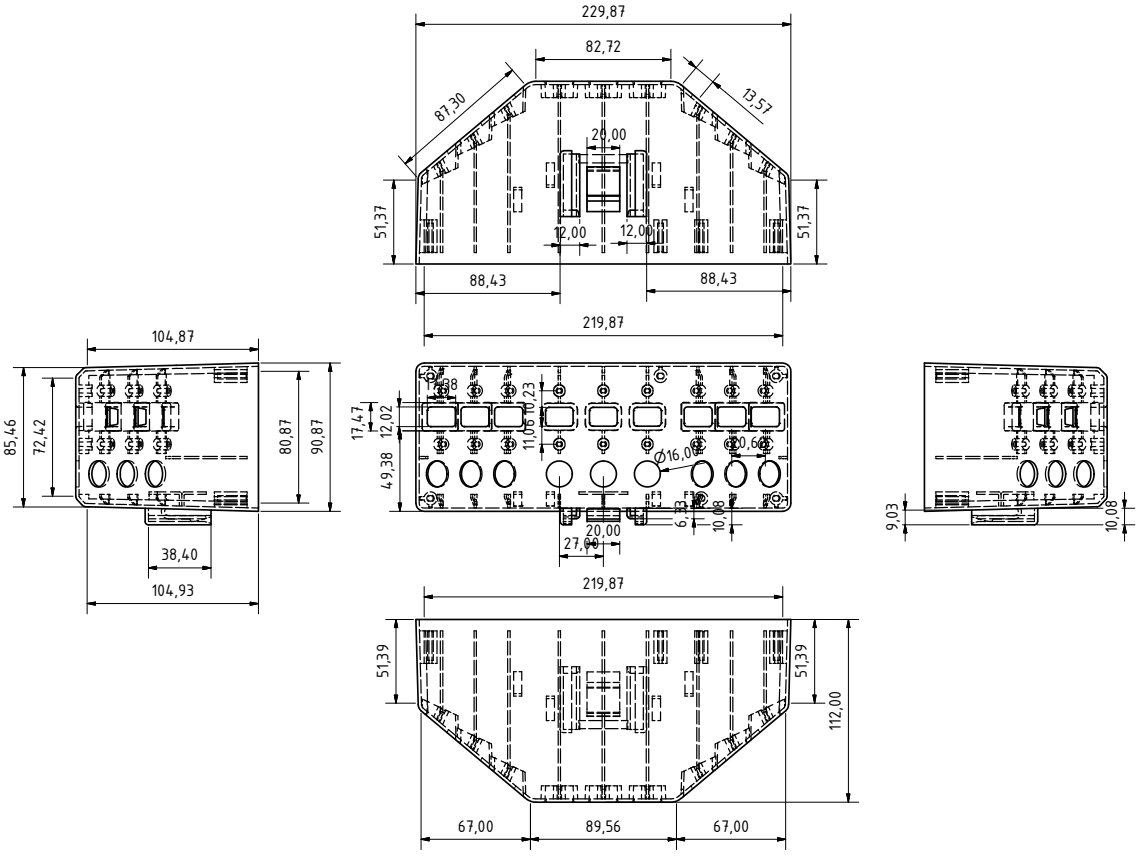


Abbildung A.2.: Bemaßte Zeichnung des Gehäuses

A.1.3. Spritzenhalterung

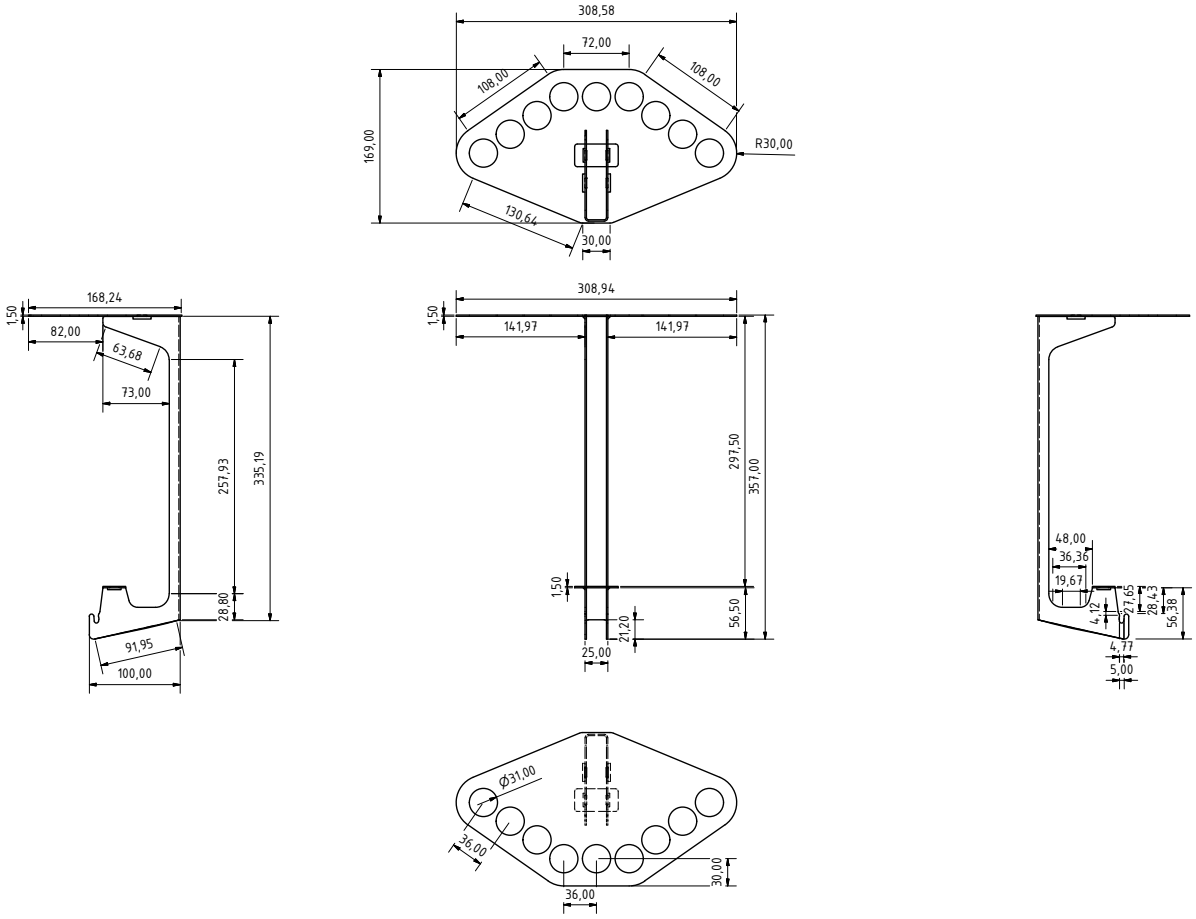


Abbildung A.3.: Bemaßte Zeichnung der Spritzenhalterung

A.1.4. Zusammenführung

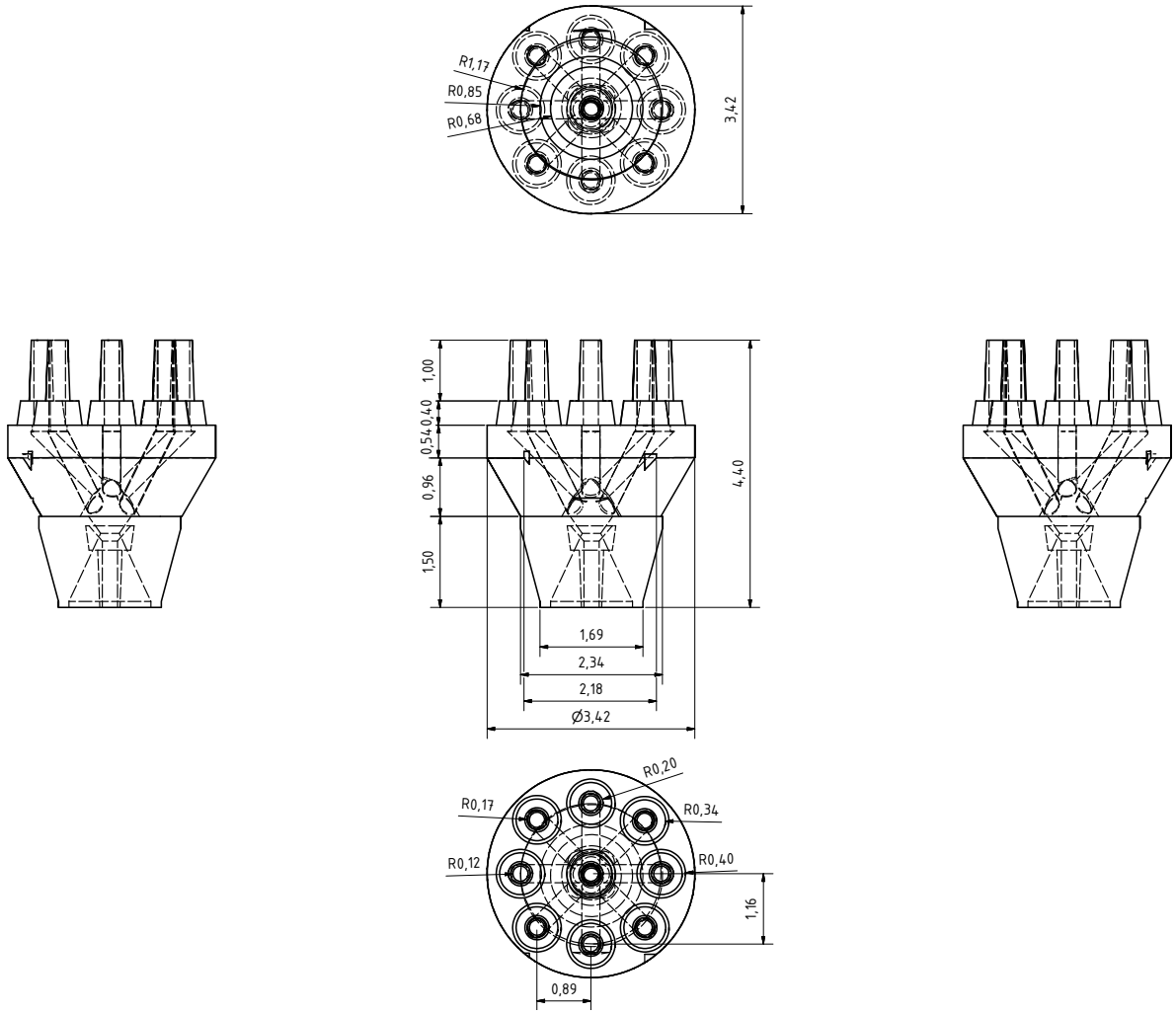


Abbildung A.4.: Bemaßte Zeichnung der Schlauchzusammenfuehrung

A.2. Messungen der Erwärmung eines einzelnen Ventils bei unterschiedlichen Ansteuerungsarten

	12V DC konstant			12V DC Start, nach 1sec 5V Haltespannung			12V DC Start, nach 1 sec PWM 40% On Time
Minute	Spannung [V]	Strom [mA]	Temperatur [°C]	Spannung [V]	Strom [mA]	Temperatur [°C]	Temperatur[°C]
1	12,15	240	27,8	12	242	25	25,2
2	12,15	225	32,6	5	100	25,8	26,7
3	12,15	220	37,6	5	100	27	28,2
4	12,15	216	42,2	5	100	27,9	28,9
5	12,15	211	47,4	5	100	28,3	29,8
6	12,15	208	51,6	5	99	28,5	30,6
7	12,15	206	54,7	5	99	28,9	30,8
8	12,15	204	57,7	5	99	29	31,1
9	12,15	203	59	5	99	29,2	31,4
10	12,15	201	62,5	5	99	29,3	31,7
11	12,15	199	63	5	99	29,4	32,1
12	12,15	198	64,9	5	99	29,4	32,3
13	12,15	197	68,2	5	99	29,5	32,6
14	12,15	196	68,5	5	99	29,7	32,7
15	12,15	195	70,1	5	99	29,7	32,8
16	12,15	194	71,1	5	99	29,8	33
17	12,15	193	74	5	99	29,9	33,3
18	12,15	193	76	5	99	29,9	33,5
19	12,15	193	81	5	99	30	33,6
20	12,15	193	82,3	5	99	30	33,7

A.3. Mikrocontroller Software

```
#include <SoftPWM.h>
#include <SoftPWM_timer.h>

//define the Number of Valves (Channels)
#define CHANNEL_SIZE 9

//include SoftPWM for using
//PWM at Pins where no Hardware PWM is available

// Software PWM Pins: 4,7,8 -> channelPin: 1,4,5
// Hardware-PWM Pins: 3,5,6,9,10,11 -> channelPin: 0,2,3,6,7,8
//number of received chars
const byte numChars = 4;

//char array that stores the all received chars
char receivedChars[numChars];

//checks if new data arrived
boolean newData = false;

//pwmWaitMillis , before pwm is activated
static unsigned long pwmWaitMillis;

//set delayPWM to default 99, to check
//if delayed pwm has to be activated
```

```
int delayPWM = 99;

// use Pins from 3-11 for Channel switching
int channelPins[] = {
  3, 4, 5, 6, 7, 8, 9, 10, 11
};

boolean statePins[] = {
  false,false,false,false,false,false,false,false,false
};
void setup() {
  //set all channelPins to Output
  //and at LOW Values, so the Transistors are closed
  for (int i = 0; i < CHANNEL_SIZE; i++)
  {
  // Set the mode of the Relay Pins to OUTPUT and close all
    pinMode(channelPins[i], OUTPUT);

  SoftPWMSet(channelPins[i], 0);
  }

  // Initialize the SoftPWM
  SoftPWMBegin();

  // Create and set pin 13 to 0 (off)
  SoftPWMSet(4, 0);
```

```
  //init the Serial Port
  Serial.begin(115200);
  Serial.flush();
  Serial.println("Perfusion");

}

void loop() {


  recvWithEndMarker();
  processReceivedData();

//checks if a delayed PWM Signal should be set
if( (long)( millis() - pwmWaitMillis ) >= 0 && delayPWM != 99)
  {
    Serial.println("Delay");

    delayed_pwm(delayPWM);
    delayPWM = 99;
  }
}

void recvWithEndMarker() {
  static byte ndx = 0;
  char endMarker = '\n';
  char rc;
```

```
  // if (Serial.available() > 0) {
  while (Serial.available() > 0 && newData == false) {
    rc = Serial.read();

    if (rc != endMarker) {
      receivedChars[ndx] = rc;
      ndx++;
      if (ndx >= numChars) {
        ndx = numChars - 1;
      }
    }
    else {
      receivedChars[ndx] = '\0'; // terminate the string
      ndx = 0;
      newData = true;
    }
  }
}


void processReceivedData() {
  if (newData == true) {
    //Serial.print(receivedChars);
    //switch the first letter after the initiator value
    switch (receivedChars[1]) {
      //ascii 'o' on second place
```

```
case 111:
  //open Ventil with Number
  channel_open(receivedChars[2] - 49);
  break;
//ascii 'c' on second place
case 99:
  //close Ventil with Number
  //subtract 49 to get the integer
  //number of the Valve from the Ascii Value
  channel_close(receivedChars[2] - 49);
  break;
//ascii 't' on second place
//toggle the Valve
case 116:
  if (receivedChars[2] == 97) {
    //if ascii 'a' is on third place, toggle all
    channel_toggle_all();
  } else {
    //else use pin_nr
    channel_toggle(receivedChars[2] - 49);
  }



default:
  //no data matching;
  break;
```

```
    }
    newData = false;
  }
}



//Function toggles a Valve, if it was open it will be closed
int channel_toggle(int channelnr) {
  boolean status = statePins[channelnr];

  if (status == true) {
    channel_close(channelnr);
    return 0;
  } else {
    channel_open(channelnr);
    return 1;
  }
}

//Function toggles all Valves
int channel_toggle_all() {
  int status = digitalRead(channelPins[1]);
  if (status == 1) {
    all_channels_close() ;
    return 0;
  } else {
```

```
    all_channels_open() ;
    return 1;
  }
}


// Function to close all Channels
// Channel close at LOW Voltage
void all_channels_close() {
  for (int i = 0; i < CHANNEL_SIZE; i++)
  {
    digitalWrite(channelPins[i], LOW); // Set the Pins Low
  }
}

// Function to open all Channels
// Channel open at HIGH Voltage
void all_channels_open() {
  for (int i = 0; i < CHANNEL_SIZE; i++)
  {
    digitalWrite(channelPins[i], HIGH); // Set the Pins High
  }
}

void delayed_pwm(int channelnr) {
  Serial.println(channelnr);
  //analogWrite(channelPins[delay_pin], 40);
```

```
//if(channelnr== 1 || channelnr == 4 || channelnr == 5)
//{
  //use Software PWM Open
  SoftPWMSet(channelPins[channelnr], 80);
//}else
//{
//   Serial.println("DelayPWM");
//   //use Hardware PWM Open
//    analogWrite(channelPins[channelnr],40);
//}

}


//Function to close Channel with channelnr
void channel_open(int channelnr) {

//if(channelnr== 1 || channelnr == 4 || channelnr == 5)
//{
  Serial.println("Software PWM OPEN");
  //use Software PWM Open
  SoftPWMSet(channelPins[channelnr], 255);
//}else
//{
//   Serial.println("Hardware PWM OPEN");
//   //use Hardware PWM Open
//   analogWrite(channelPins[channelnr], 255);
```

```
//}
  statePins[channelnr] = true;


  delayPWM=channelnr;
  pwmWaitMillis = millis() + 50;  // initial setup delay
}

//Function to close Channel with channelnr
void channel_close(int channelnr) {
  statePins[channelnr] = false;

//if(channelnr== 1|| channelnr == 4 || channelnr == 5)
//{
  //use Software PWM Open
  SoftPWMSet(channelPins[channelnr], 0);
  Serial.println("Software PWM CLOSE");
//}else
//{
//  //use Hardware PWM Open
//  analogWrite(channelPins[channelnr], 0);
//  Serial.println("Hardware PWM CLOSE");
//}


}
```

A.4. Schaltpläne der verwendeten Prototyping Plattformen

A.4.1. Arduino Leonardo

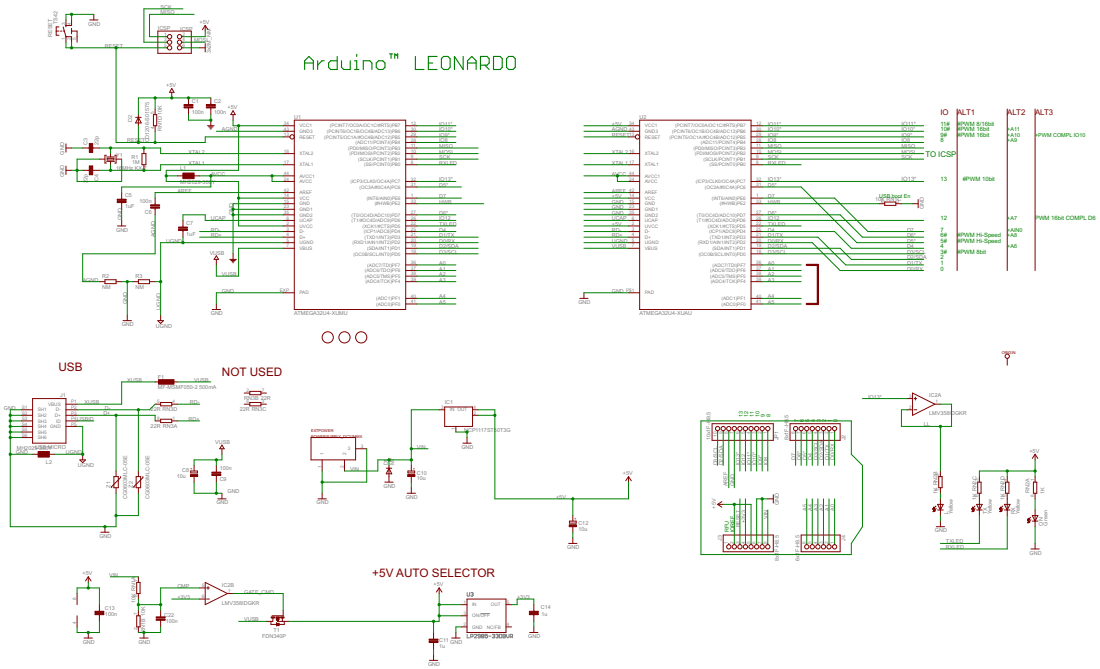


Abbildung A.5.: Schaltplan des Prototyping-Boards Arduino Leonardo [30]

A.4.2. Arduino Uno

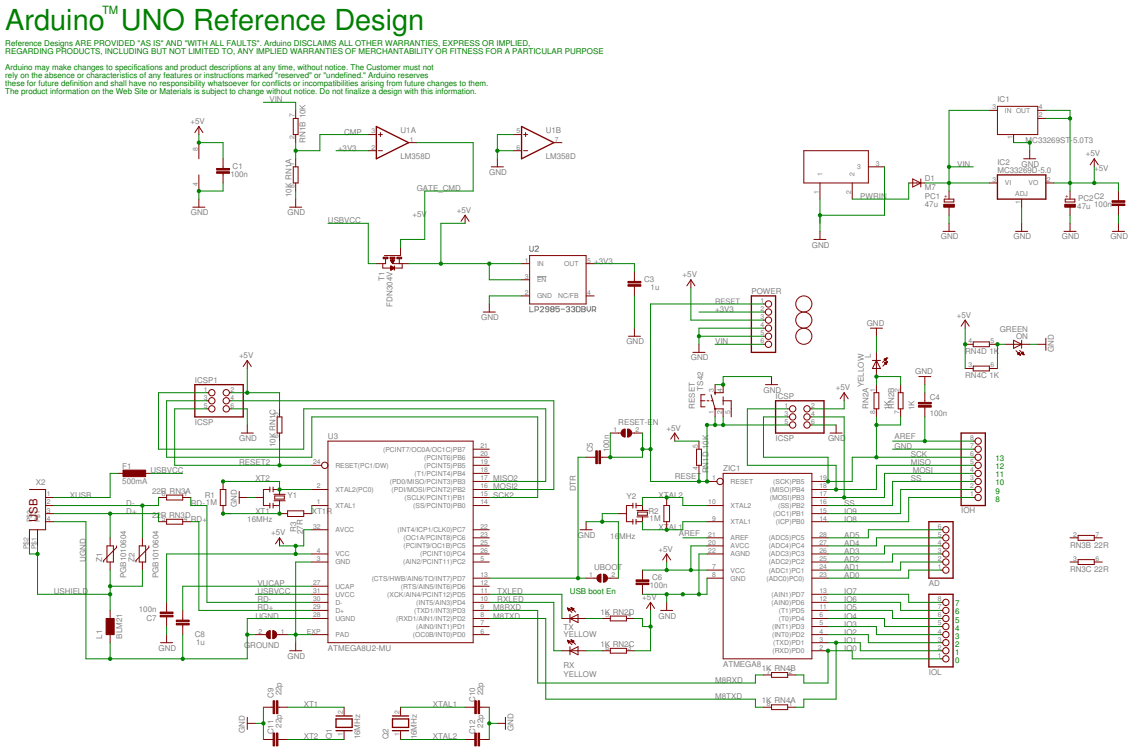


Abbildung A.6.: Schaltplan des Prototyping-Boards Arduino Uno [31]

A.5. Dokumentation der Anwendungssoftware

Perfusion

Generated by Doxygen 1.8.14

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Channel Class Reference

Public Member Functions

- Channel ()

  The Standard Constructor of the Channel Class.
- int getchannelNr ()

  Getter for current Channel Number.
- void setChannelNr (int mChannelNr)

  Setter for the Channel Number.
- bool getState ()

  Getter for the current State.
- void setState (bool mState)

  Setter for the current State.
- QString getchannelName ()

  Getter for the Current Channel Name.
- void setchannelName (QString mchannelName)

  Setter for the Channel Name.
- QList< QTime > * getOnSwitchesPt ()

  Getter for the On Switches.
- QList< QTime > * getOffSwitchesPt ()

  Getter for the Off Switches.
- void appendOnSwitch (int min, int sec)

  Setter for the Channel Name.
- void appendOffSwitch (int min, int sec)

  Append a new Time where the Channel should be Closed.
- void **remOnSwitch** (int index)
- void **remOffSwitch** (int index)
- bool getAutoUsed ()

  Getter for the Automatic Used Boolean Variable.
- void setAutoUsed (bool mAutoUsed)

  Setter for the Automatic Used Boolean Variable.
- void removeSwitches ()

  Removes all timed Switches.

Private Attributes

- int channelNr

  holds the Number of the Current Channel, value from 1 to 9
- bool state

  holds the state of the Current Channel, false is closed, true is open
- QString channelName

  holds the current Channel Name
- QList< QTime > onSwitches

  List of the times when the Channel should be switched on.
- QList< QTime > offSwitches

  List of the times when the Channel should be switched on.
- bool autoUsed

  holds the state if the Channel is used for Auto Mode, 0 if it is not used, 1 if its used

4.1.1 Member Function Documentation

4.1.1.1 appendOffSwitch()

```
void Channel::appendOffSwitch (
            int min,
            int sec )
```

Append a new Time where the Channel should be Closed.

Parameters

min	holds the Time in Minutes
sec	holds the Time in Seconds

Returns

void

4.1.1.2 appendOnSwitch()

```
void Channel::appendOnSwitch (
            int min,
            int sec )
```

Setter for the Channel Name.

Parameters

mchannelName	holds the new Channel Name

Returns

void

4.1.1.3 getAutoUsed()

```
bool Channel::getAutoUsed ( )
```

Getter for the Automatic Used Boolean Variable.

Returns

the Boolean Variable if the Channel is used for the Automatic Mode

4.1.1.4 getchannelName()

```
QString Channel::getchannelName ( )
```

Getter for the Current Channel Name.

Returns

the current assigned Channel Name

4.1.1.5 getchannelNr()

```
int Channel::getchannelNr ( )
```

Getter for current Channel Number.

Returns

the current Channel Number

4.1.1.6 getOffSwitchesPt()

```
QList< QTime > * Channel::getOffSwitchesPt ( )
```

Getter for the Off Switches.

Returns

A Pointer to the List of current Timers to switch the Channel Off

4.1.1.7 getOnSwitchesPt()

```
QList< QTime > * Channel::getOnSwitchesPt ( )
```

Getter for the On Switches.

Returns

A Pointer to the List of current Timers to switch the Channel On

4.1.1.8 getState()

```
bool Channel::getState ( )
```

Getter for the current State.

Returns

the current State of the Channel

4.1.1.9 removeSwitches()

```
void Channel::removeSwitches ( )
```

Removes all timed Switches.

Returns

void

4.1.1.10 setAutoUsed()

```
void Channel::setAutoUsed (
            bool mAutoUsed )
```

Setter for the Automatic Used Boolean Variable.

Parameters

mAutoUsed	holds the Boolean Value if the Channel is Used for the Automatic Mode

Returns

void

4.1.1.11 setchannelName()

```
void Channel::setchannelName (
            QString mchannelName )
```

Setter for the Channel Name.

Parameters

mchannelName	holds the new Channel Name

Returns

void

4.1.1.12 setChannelNr()

```
void Channel::setChannelNr (
            int mChannelNr )
```

Setter for the Channel Number.

Parameters

mchannelName	holds the new Channel Number

Returns

void

4.1.1.13 setState()

```
void Channel::setState (
            bool mState )
```

Setter for the current State.

Parameters

mState	holds the new State of the Channel

Returns

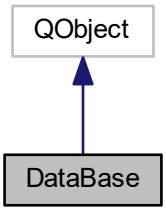
void

The documentation for this class was generated from the following files:

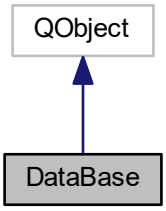
- C:/Perfusion/channel.h
- C:/Perfusion/channel.cpp

4.2 DataBase Class Reference

Inheritance diagram for DataBase:



Collaboration diagram for DataBase:



Public Slots

- bool insertRunIntoTable (const QVariantList &data)

  Insert a Run Template in the Table.
- bool insertRunIntoTable (const QString date, const QString time, const QString dur, const QString channel↩ Names, const QString description, const QString C1On, const QString C1Dur, const QString C2On, const QString C2Dur, const QString C3On, const QString C3Dur, const QString C4On, const QString C4Dur, const QString C5On, const QString C5Dur, const QString C6On, const QString C6Dur, const QString C7On, const QString C7Dur, const QString C8On, const QString C8Dur, const QString C9On, const QString C9Dur)

  Insert a Run Template in the Table.
- bool insertSwitchIntoTable (const QVariantList &date)

  Insert a History Switch into the Table.
- bool insertSwitchIntoTable (const QString date, const QString time, const int channelNr, const bool state, const QString channelName)

  Insert a History Switch in the Table.
- bool insertPufferIntoTable (const QVariantList &data)

  Insert a Puffer Name in the Table.
- bool insertPufferIntoTable (const QString &pname)

  Insert a Puffer Name in the Table.
- bool removePuffer (const int id)

  Removes a Puffername from the Puffer Table.

Public Member Functions

- DataBase (QObject ∗parent=0)

  Standard Constructor.
- ∼DataBase ()

  Standard Destructor.
- void connectToDataBase ()

  Checks if a DataBase exists, if there is a existing Database the DB is opened, if there is no existing DB a new DB is created with the Function restoreDataBase.

Private Member Functions

- bool openDataBase ()

  Opens the Database.
- bool restoreDataBase ()

  Restores the Database and creates Tables.
- void closeDataBase ()

  Close the current Database.
- bool createTable ()

  Create the Table for the Puffer Names.
- bool createRunTable ()

  Create the Table for the Run-Templates.
- bool createHistoryTable ()

  Create the Table for the History Entries.

Private Attributes

- QSqlDatabase db

  Object that holds the current Database.
- QDateTime dateTimeNow

  Object that holds the current Time for storing Date and Time.

4.2.1 Member Function Documentation

4.2.1.1 closeDataBase()

```
void DataBase::closeDataBase ( )  [private]
```

Close the current Database.

Returns

void

4.2.1.2 connectToDataBase()

```
void DataBase::connectToDataBase ( )
```

Checks if a DataBase exists, if there is a existing Database the DB is opened, if there is no existing DB a new DB is created with the Function restoreDataBase.

Returns

void

4.2.1.3 createHistoryTable()

```
bool DataBase::createHistoryTable ( )  [private]
```

Create the Table for the History Entries.

Returns

true if the Table could be created

4.2.1.4 createRunTable()

```
bool DataBase::createRunTable ( )  [private]
```

Create the Table for the Run-Templates.

Returns

true if the Table could be created

4.2.1.5 createTable()

```
bool DataBase::createTable ( )  [private]
```

Create the Table for the Puffer Names.

Returns

true if the Table could be created

4.2.1.6 insertPufferIntoTable [1/2]

```
bool DataBase::insertPufferIntoTable (
            const QVariantList & data )  [slot]
```

Insert a Puffer Name in the Table.

Parameters

&data	a VariantList of the appended Data

Returns

true if Data could be appended

4.2.1.7 insertPufferIntoTable [2/2]

```
bool DataBase::insertPufferIntoTable (
            const QString & pname )  [slot]
```

Insert a Puffer Name in the Table.

Parameters

&pname	the Name of the New Puffer

Returns

true if the Puffer name could be append

4.2.1.8 insertRunIntoTable [1/2]

```
bool DataBase::insertRunIntoTable (
            const QVariantList & data )  [slot]
```

Insert a Run Template in the Table.

Parameters

&data	Variant List of all SwitchValues and Durations

Returns

true if the data could be append

4.2.1.9 insertRunIntoTable [2/2]

```
bool DataBase::insertRunIntoTable (
            const QString date,
            const QString time,
            const QString dur,
            const QString channelNames,
            const QString description,
            const QString C1On,
            const QString C1Dur,
            const QString C2On,
            const QString C2Dur,
            const QString C3On,
            const QString C3Dur,
            const QString C4On,
            const QString C4Dur,
            const QString C5On,
            const QString C5Dur,
            const QString C6On,
            const QString C6Dur,
            const QString C7On,
            const QString C7Dur,
            const QString C8On,
```

```
            const QString C8Dur,
            const QString C9On,
            const QString C9Dur )  [slot]
```

Insert a Run Template in the Table.

Returns

true if the data could be append

4.2.1.10 insertSwitchIntoTable [1/2]

```
bool DataBase::insertSwitchIntoTable (
            const QVariantList & data )  [slot]
```

Insert a History Switch into the Table.

Parameters

&data	holds the Data of the History Switch

Returns

true if the data could be append

4.2.1.11 insertSwitchIntoTable [2/2]

```
bool DataBase::insertSwitchIntoTable (
            const QString date,
            const QString time,
            const int channelNr,
            const bool state,
            const QString channelName )  [slot]
```

Insert a History Switch in the Table.

Returns

true if the data could be append

4.2.1.12 openDataBase()

```
bool DataBase::openDataBase ( )  [private]
```

Opens the Database.

Returns

true if the Database could be opened, false, if there was a error

4.2.1.13 removePuffer

```
bool DataBase::removePuffer (
            const int id )  [slot]
```

Removes a Puffername from the Puffer Table.

Returns

true if the data could be removed

4.2.1.14 restoreDataBase()

```
bool DataBase::restoreDataBase ( )  [private]
```

Restores the Database and creates Tables.

Returns

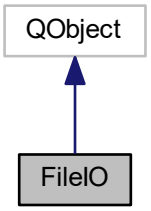
true if the Database could be restored

The documentation for this class was generated from the following files:

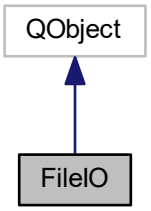
- C:/Perfusion/database.h
- C:/Perfusion/database.cpp

4.3 FileIO Class Reference

Inheritance diagram for FileIO:



Collaboration diagram for FileIO:



Public Slots

- void setSource (QUrl source)

  set the Path to the current used CSV File
- QList< QString > readCSV ()

  Reads the whole CSV File.

Public Member Functions

- FileIO (QObject *parent=0)

  Constructor.
- ~FileIO ()

  Destructor.

Private Attributes

- QUrl file_source

  the Path to the CSV File

4.3.1 Member Function Documentation

4.3.1.1 readCSV

```
QList< QString > FileIO::readCSV ( )  [slot]
```

Reads the whole CSV File.

Returns

the Value of the File in a QList, where a line is a entry of the list

4.3.1.2 setSource

```
void FileIO::setSource (
            QUrl source )  [slot]
```

set the Path to the current used CSV File

Parameters

source	holds the source Path of the used File

Returns

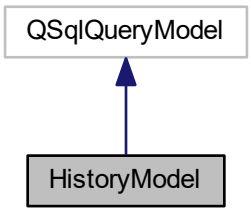
void

The documentation for this class was generated from the following files:

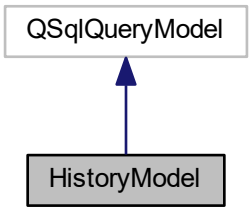
- C:/Perfusion/fileio.h
- C:/Perfusion/fileio.cpp

4.4 HistoryModel Class Reference

Inheritance diagram for HistoryModel:



Collaboration diagram for HistoryModel:



Public Types

- enum **Roles** {
  IdRole = Qt::UserRole + 1, **dateRole**, **timeRole**, **chNrRole**,
  stateRole, **chNameRole** }

Public Slots

- void **updateModel** ()
- int **getId** (int row)
- void setSelDate (QString selectDate)

  get the Data for the selected Date

Public Member Functions

- **HistoryModel** (QObject ∗parent=0)
- QVariant **data** (const QModelIndex &index, int role=Qt::DisplayRole) const

Public Attributes

- QString **selDate**

Protected Member Functions

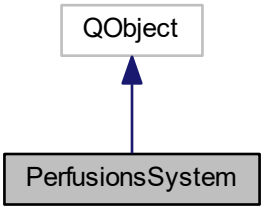
- QHash< int, QByteArray > **roleNames** () const

The documentation for this class was generated from the following files:

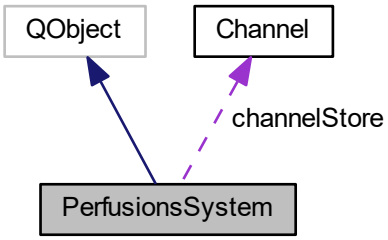
- C:/Perfusion/historymodel.h
- C:/Perfusion/historymodel.cpp

4.5 PerfusionsSystem Class Reference

Inheritance diagram for PerfusionsSystem:



Collaboration diagram for PerfusionsSystem:



Public Slots

- void switchChannelOn (int channelNr)

  switches Channel with given channelNr immediately on
- void switchChannelOff (int channelNr)

  switches Channel with given channelNr immediately off
- void **appendOnSwitch** (int channelNr, int onT)
- void appendOffSwitch (int channelNr, int offT)

  append the Time Value, where a Channel should be closed
- void startRun ()

  start the programmed Run
- void deleteSeriesElement (int chnr, int startmin, int endmin)

  delete a Element of the Timer List
- void setAutoState (int chnr, bool mRunState)

  set the State of the Automatic Mode of the Channel-if the channel is used
- bool getAutoState (int chnr)

  gets the State of the Automatic Mode of the Channel
- void setCName (int chnr, QString mChannelName)

  set the Name of the Channel with the used Puffer
- QString getCName (int chnr)

  get the Name of the Channel
- bool getState (int channelNr)

  get the current State of the Channel
- void connect ()

  Start a Serial Connection to the Mikrcontroller.
- void searchComName ()

  searches the Com Name of the Perfusion System
- void setmaxTimeRun (int maxT)

  sets the maximal Time of the Automatic Run
- int getmaxTimeRun ()

  gets the maximal Time of the Automatic Run
- void stopTimers ()

  stopp all planned Timers immediately
- void removeSwitches ()

  removeallSwitches and stop the Timers for all Channels

Public Member Functions

- PerfusionsSystem (QObject *parent=0)

  the Standard Constructor of the PerfusionsSystem Object, creates 9 Channel Objects and stores all Channel in the Array channelStore

Private Attributes

- QSerialPort * serial

  Variable that holds a pointer to the current used Serial Port.
- Channel channelStore [9]

  Array that holds all 9 Channel - Objects.
- QString comPortName

  holds the current PortName
- int maxTime

  the maximal Time of the current Automatic Run
- QList< QTimer * > timerList

  List with all Timers that open or close a Channel.

4.5.1 Constructor & Destructor Documentation

4.5.1.1 PerfusionsSystem()

```
PerfusionsSystem::PerfusionsSystem (
            QObject * parent = 0 )  [explicit]
```

the Standard Constructor of the PerfusionsSystem Object, creates 9 Channel Objects and stores all Channel in the Array channelStore

create the 9 Channel Objects and hold them in an array for later using set the Channel Names from Channel0 to Channel9 and the Channel Numbers

4.5.2 Member Function Documentation

4.5.2.1 appendOffSwitch

```
void PerfusionsSystem::appendOffSwitch (
            int channelNr,
            int offT )  [slot]
```

append the Time Value, where a Channel should be closed

Parameters

channelNr	is the Nummer of the Channel, which should be closed
offT	is the Time in seconds, where the Channel should be closed

Returns

void

4.5.2.2 connect

```
void PerfusionsSystem::connect ( )  [slot]
```

Start a Serial Connection to the Mikrcontroller.

Returns

void

4.5.2.3 deleteSeriesElement

```
void PerfusionsSystem::deleteSeriesElement (
            int chnr,
            int startT,
            int endT )  [slot]
```

delete a Element of the Timer List

Parameters

chnr	is the ChannelNummer of the Timer Element that should be deleted
startT	is the Time where the Channel should be opened
endT	is the Time where the Channel should be closed

The QListIterator constructor takes a QList as argument. After construction, the iterator is located at the very beginning of the list (before the first item)

4.5.2.4 getAutoState

```
bool PerfusionsSystem::getAutoState (
            int chnr )  [slot]
```

gets the State of the Automatic Mode of the Channel

Parameters

chnr	the ChannelNummer of the unknown Channel State

Returns

the state if the Channel is used in the Automatic Mode

4.5.2.5 getCName

```
QString PerfusionsSystem::getCName (
            int chnr )  [slot]
```

get the Name of the Channel

Parameters

chnr	the ChannelNummer where the Name should be known

Returns

the current Channel Name

4.5.2.6 getmaxTimeRun

```
int PerfusionsSystem::getmaxTimeRun ( )  [slot]
```

gets the maximal Time of the Automatic Run

Parameters

chnr	the ChannelNummer where the Name should be known

Returns

the maximal Time of the Automatic run

4.5.2.7 getState

```
bool PerfusionsSystem::getState (
            int channelNr )  [slot]
```

get the current State of the Channel

Parameters

channelNr	is the Nummer of the Channel where the current State should be known

Returns

the current State of the Channel, false means closed and true means open

4.5.2.8 removeSwitches

```
void PerfusionsSystem::removeSwitches ( )  [slot]
```

removeallSwitches and stop the Timers for all Channels

Returns

void

<iterate over the Array with all Channels get the Pointers of the Start and Stop Times

4.5.2.9 searchComName

```
void PerfusionsSystem::searchComName ( )  [slot]
```

searches the Com Name of the Perfusion System

Returns

void

< holds Information about all Serial Ports of the Computer

search all Serial Ports for a correct description of the Device

4.5.2.10 setAutoState

```
void PerfusionsSystem::setAutoState (
            int chnr,
            bool mRunState )  [slot]
```

set the State of the Automatic Mode of the Channel-if the channel is used

Parameters

chnr	the ChannelNummer where the State should be set

Returns

void

4.5.2.11 setCName

```
void PerfusionsSystem::setCName (
            int chnr,
            QString mChannelName )  [slot]
```

set the Name of the Channel with the used Puffer

Parameters

chnr	the ChannelNummer where the Name should be set
mChannelName	the new Name of the Channel

Returns

void

4.5.2.12 setmaxTimeRun

```
void PerfusionsSystem::setmaxTimeRun (
            int maxT )  [slot]
```

sets the maximal Time of the Automatic Run

Parameters

maxT	the maximal summed Time

Returns

void

4.5.2.13 startRun

```
void PerfusionsSystem::startRun ( )  [slot]
```

start the programmed Run

Returns

void

<Iterator that holds the Timer with the Time where the Channel should be open

The next() function returns the next item in the list and advances the iterator.

<connect the signal and the Slot of the Timers, use a Lambda Function and use the variable of the chcount

convert the Timer Value in Milliseconds

append the current Timer Object to a Timer List

Iterator that holds the Timer with the Time where the Channel should be closed

4.5.2.14 stopTimers

```
void PerfusionsSystem::stopTimers ( )  [slot]
```

stopp all planned Timers immediately

Returns

void

4.5.2.15 switchChannelOff

```
void PerfusionsSystem::switchChannelOff (
            int channelNr )  [slot]
```

switches Channel with given channelNr immediately off

Parameters

channelNr	is the Nummer of the Channel, which should be closed

Returns

void

4.5.2.16 switchChannelOn

```
void PerfusionsSystem::switchChannelOn (
            int channelNr )  [slot]
```

switches Channel with given channelNr immediately on

Parameters

channelNr	is the Nummer of the Channel, which should be openend

Returns

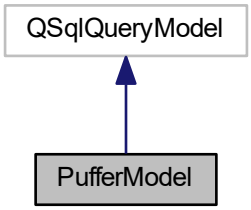
void

The documentation for this class was generated from the following files:

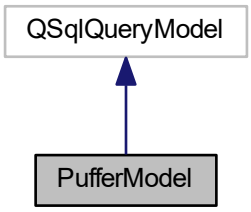
- C:/Perfusion/perfusionssystem.h
- C:/Perfusion/perfusionssystem.cpp

4.6 PufferModel Class Reference

Inheritance diagram for PufferModel:



Collaboration diagram for PufferModel:



Public Types

- enum **Roles** { **IdRole** = Qt::UserRole + 1, **pNameRole** }

Public Slots

- void **updateModel** ()
- int **getId** (int row)

Public Member Functions

- **PufferModel** (QObject ∗parent=0)
- QVariant **data** (const QModelIndex &index, int role=Qt::DisplayRole) const

Protected Member Functions

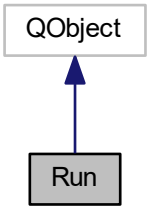
- QHash< int, QByteArray > **roleNames** () const

The documentation for this class was generated from the following files:

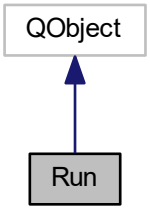
- C:/Perfusion/puffermodel.h
- C:/Perfusion/puffermodel.cpp

4.7 Run Class Reference

Inheritance diagram for Run:



Collaboration diagram for Run:



Public Slots

- int **getGesDur** () const
- void **setGesDur** (int value)
- int **getTotalSwitches** () const
- void **setTotalSwitches** (int value)
- QString **getDescription** () const
- void **setDescription** (const QString &value)
- void **appendSwitchComa** (int channelNr, QString startTime, QString duration)
- void **appendSingleSwitch** (int channelNr, QTime startTime, QTime duration)
- void **appendSingleSwitch** (int channelNr, int startTime, int duration)
- void **appendCSVString** (QList< QString > CSVString)
- void **calcTotalSwitches** ()
- QTime **getCDur** (int listValue)
- QTime **getStart** (int listValue)
- int **getCNumber** (int listValue)
- int **getStartInt** (int listValue)
- int **getCDurInt** (int listValue)

- void **clearAll** ()
- const QVariantList **storeDB** ()
- void **setDate** (QString dateN)
- void **setTime** (QString timeN)
- void **setChannelName** (int channelNr, QString channelName)
- void **convertChannelNames** ()

Public Member Functions

- **Run** (QObject *parent=0)

Private Attributes

- int gesDur

  holds the summed Duration
- QString channelNames [9]

  holds all Channel Names used for the Run
- QString channelNamesString

  holds all Channel Names in a String
- QList< Switch > switchList

  List with all appended Switches from the GUI.
- int totalSwitches

  holds the count of all Switches
- QString description

  holds the Description of the current run
- QString date

  holds the current Date
- QString time

  holds the current Time

The documentation for this class was generated from the following files:

- C:/Perfusion/run.h
- C:/Perfusion/run.cpp

4.8 Switch Class Reference

Public Member Functions

- int **getChannelNr** ()
- void **setChannelNr** (int value)
- QTime **getStartTime** ()
- void **setStartTime** (QTime value)
- QTime **getDuration** ()
- void **setDuration** (QTime value)

Private Attributes

- int channelNr

  holds the Channel Number of the current Switch
- QTime startTime

  holds the Time where the Channel should be opened
- QTime duration

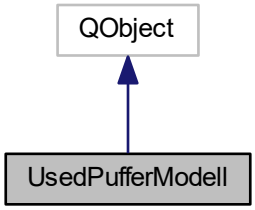
  holds the Duration how long the Channel should be left open

The documentation for this class was generated from the following files:

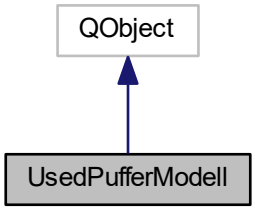
- C:/Perfusion/switch.h
- C:/Perfusion/switch.cpp

4.9 UsedPufferModell Class Reference

Inheritance diagram for UsedPufferModell:



Collaboration diagram for UsedPufferModell:



Signals

- void **curPufferListChanged** ()
- void **countChanged** ()

Public Member Functions

- **UsedPufferModell** (QObject ∗parent=0)
- const QStringList **curPufferList** ()
- void **setCurPufferList** (const QStringList &curList)
- int **count** ()
- void **setCount** (int cnt)
- void **changeName** (int index)
- Q_INVOKABLE void **addElement** (const QString &element)
- Q_INVOKABLE void **removeElement** (int index)
- Q_INVOKABLE void **changeElementName** (int index, const QString &elementName)

Properties

- QStringList **curPufferList**
- int **count**

Private Attributes

- QStringList m_curPufferList

  holds all currently used Puffer
- int m_count

  Variable holds the Sum of all Used Puffers.

The documentation for this class was generated from the following files:

- C:/Perfusion/usedpuffermodell.h
- C:/Perfusion/usedpuffermodell.cpp

Chapter 5

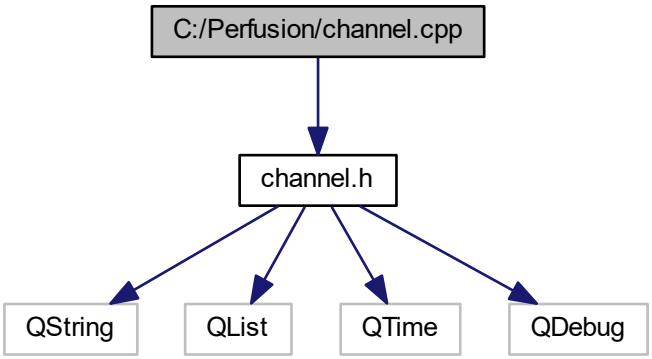
File Documentation

5.1 C:/Perfusion/channel.cpp File Reference

Class Channel models a switchable Channel, that can be switched on and off.

```
#include "channel.h"
```

Include dependency graph for channel.cpp:



5.1.1 Detailed Description

Class Channel models a switchable Channel, that can be switched on and off.

Author

Hubert Tockner

5.1.2 DESCRIPTION

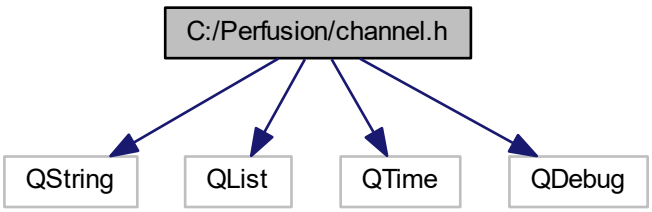
This is the Class that models Channel Objects

5.2 C:/Perfusion/channel.h File Reference

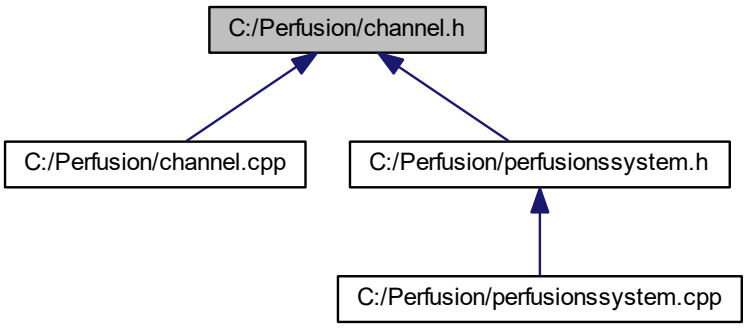
Header File of the Channel Class.

```
#include <QString>
#include <QList>
#include <QTime>
#include <QDebug>
```

Include dependency graph for channel.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Channel

5.2.1 Detailed Description

Header File of the Channel Class.

Author

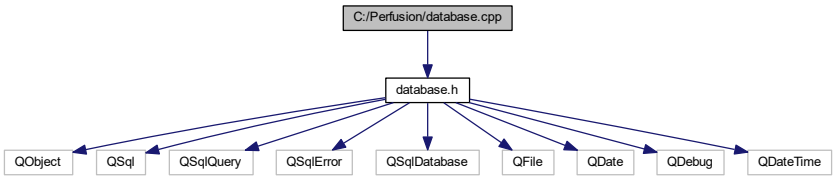
Hubert Tockner

5.3 C:/Perfusion/database.cpp File Reference

Class that can be used for the Creation of the used Database and Transitions.

```
#include "database.h"
```

Include dependency graph for database.cpp:



5.3.1 Detailed Description

Class that can be used for the Creation of the used Database and Transitions.

Author

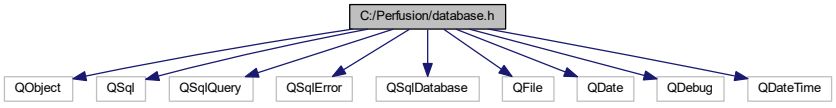
Hubert Tockner

5.4 C:/Perfusion/database.h File Reference

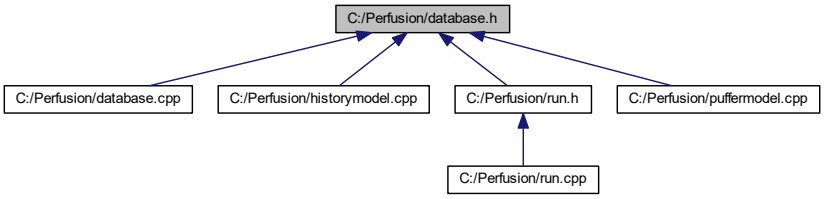
Header File for the DataBase Class.

```
#include <QObject>
#include <QSql>
#include <QSqlQuery>
#include <QSqlError>
#include <QSqlDatabase>
#include <QFile>
#include <QDate>
#include <QDebug>
#include <QDateTime>
```

Include dependency graph for database.h:



This graph shows which files directly or indirectly include this file:



Classes

- class DataBase

Macros

- #define DATABASE_HOSTNAME "PerfDatabase"

  Define Constants.
- #define **DATABASE_NAME** "Perf.db"
- #define **TABLE** "PufferTable"
- #define **TABLE_description** "Description"
- #define **TABLE_date** "Date"
- #define **TABLE_time** "Time"
- #define **TABLE_dur** "Duration"
- #define **TABLE_pname** "PufferName"
- #define **TABLE_cnames** "ChannelNames"
- #define **TABLE_RUN** "RunTable"
- #define **TABLE_C1ON** "C1OnSwitches"
- #define **TABLE_C1DUR** "C1Duration"
- #define **TABLE_C2ON** "C2OnSwitches"
- #define **TABLE_C2DUR** "C2Duration"
- #define **TABLE_C3ON** "C3OnSwitches"
- #define **TABLE_C3DUR** "C3Duration"
- #define **TABLE_C4ON** "C4OnSwitches"
- #define **TABLE_C4DUR** "C4Duration"
- #define **TABLE_C5ON** "C5OnSwitches"
- #define **TABLE_C5DUR** "C5Duration"
- #define **TABLE_C6ON** "C6OnSwitches"
- #define **TABLE_C6DUR** "C6Duration"
- #define **TABLE_C7ON** "C7OnSwitches"
- #define **TABLE_C7DUR** "C7Duration"
- #define **TABLE_C8ON** "C8OnSwitches"
- #define **TABLE_C8DUR** "C8Duration"
- #define **TABLE_C9ON** "C9OnSwitches"
- #define **TABLE_C9DUR** "C9Duration"

5.4.1 Detailed Description

Header File for the DataBase Class.

Author

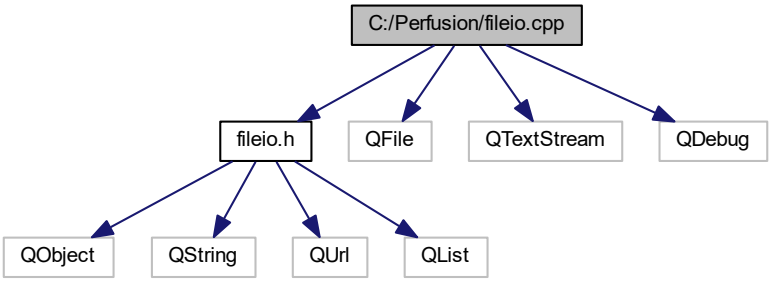
Hubert Tockner

5.5 C:/Perfusion/fileio.cpp File Reference

Class for the FileIO Object that can be used to store and read CSV Files.

```
#include "fileio.h"
#include <QFile>
#include <QTextStream>
#include <QDebug>
```

Include dependency graph for fileio.cpp:



5.5.1 Detailed Description

Class for the FileIO Object that can be used to store and read CSV Files.

Author

Hubert Tockner

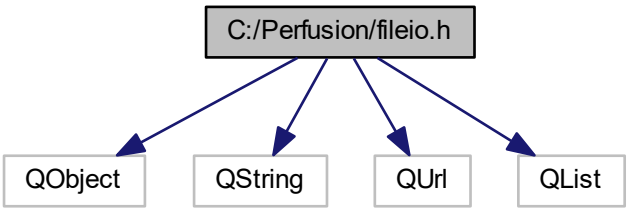
5.6 C:/Perfusion/fileio.h File Reference

Header File for the FileIO Class.

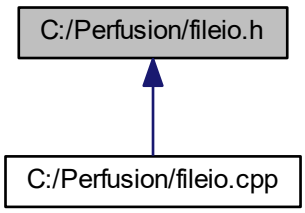
```
#include <QObject>
#include <QString>
#include <QUrl>
```

```
#include <QList>
```

Include dependency graph for fileio.h:



This graph shows which files directly or indirectly include this file:



Classes

- class FileIO

5.6.1 Detailed Description

Header File for the FileIO Class.

Author

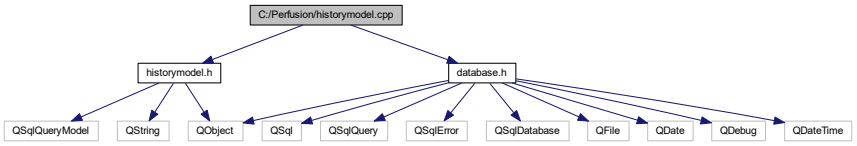
Hubert Tockner

5.7 C:/Perfusion/historymodel.cpp File Reference

Class that gets the History Switches from the Database.

```
#include "historymodel.h"
#include "database.h"
```

Include dependency graph for historymodel.cpp:



5.7.1 Detailed Description

Class that gets the History Switches from the Database.

Author

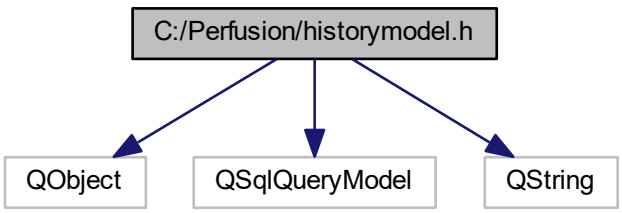
Hubert Tockner

5.8 C:/Perfusion/historymodel.h File Reference

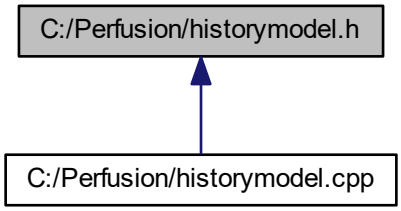
Header File for the History Model.

```
#include <QObject>
#include <QSqlQueryModel>
#include <QString>
```

Include dependency graph for historymodel.h:



This graph shows which files directly or indirectly include this file:



Classes

- class HistoryModel

5.8.1 Detailed Description

Header File for the History Model.

Author

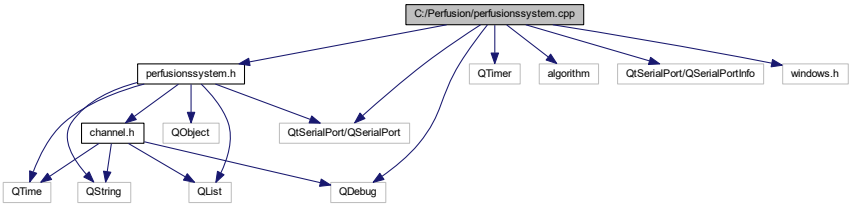
Hubert Tockner

5.9 C:/Perfusion/perfusionssystem.cpp File Reference

Class PerfusionSystem holds the Objects of the Channels, Serial Ports and Timers.

```
#include "perfusionssystem.h"
#include <QDebug>
#include <QTimer>
#include <algorithm>
#include <QtSerialPort/QSerialPort>
#include <QtSerialPort/QSerialPortInfo>
#include "windows.h"
```

Include dependency graph for perfusionssystem.cpp:



5.9.1 Detailed Description

Class PerfusionSystem holds the Objects of the Channels, Serial Ports and Timers.

Author

Hubert Tockner

5.9.2 DESCRIPTION

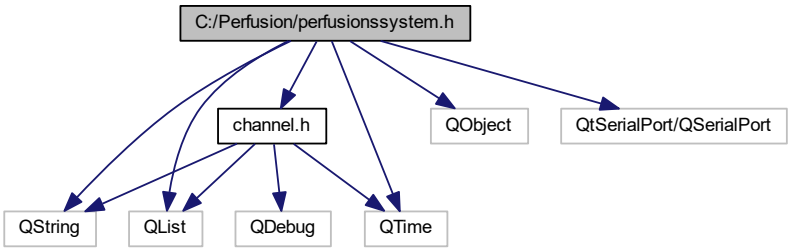
This is the Main Class that holds all Components. A Array with 9 Channel Objects holds the current status of all Channels. The variable serial can be used to write Commands to the Serial Port.

5.10 C:/Perfusion/perfusionssystem.h File Reference

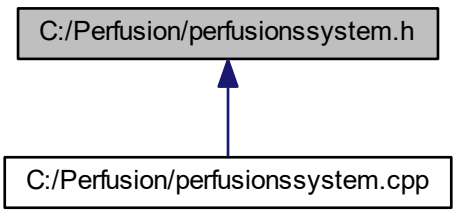
Header File of the PerfusionSystem Class.

```
#include "channel.h"
#include <QString>
#include <QObject>
#include <QtSerialPort/QSerialPort>
#include <QList>
#include <QTime>
```

Include dependency graph for perfusionssystem.h:



This graph shows which files directly or indirectly include this file:



Classes

- class PerfusionsSystem

5.10.1 Detailed Description

Header File of the PerfusionSystem Class.

Author

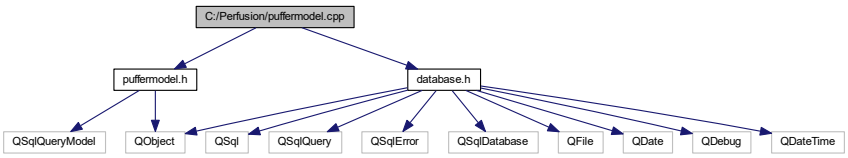
Hubert Tockner

5.11 C:/Perfusion/puffermodel.cpp File Reference

Class that models all Puffer.

```
#include "puffermodel.h"
#include "database.h"
```

Include dependency graph for puffermodel.cpp:



5.11.1 Detailed Description

Class that models all Puffer.

Author

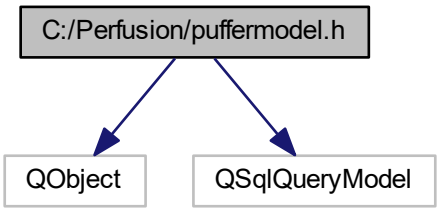
Hubert Tockner

5.12 C:/Perfusion/puffermodel.h File Reference

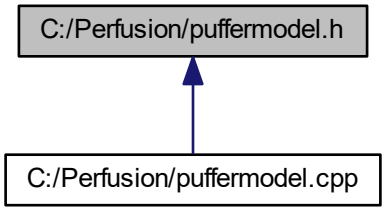
Header File for the PufferModel Class.

```
#include <QObject>
#include <QSqlQueryModel>
```

Include dependency graph for puffermodel.h:



This graph shows which files directly or indirectly include this file:



Classes

- class PufferModel

5.12.1 Detailed Description

Header File for the PufferModel Class.

Author

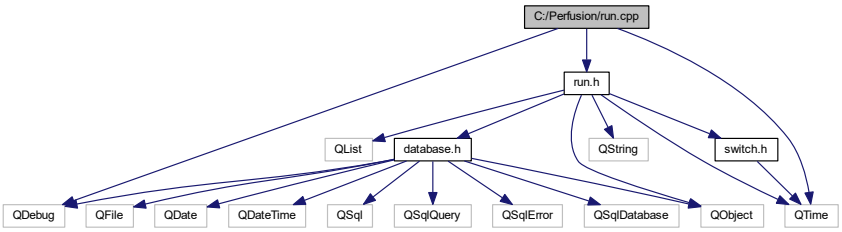
Hubert Tockner

5.13 C:/Perfusion/run.cpp File Reference

Class that models an automatic Run.

```
#include "run.h"
#include <QDebug>
#include <QTime>
```

Include dependency graph for run.cpp:



5.13.1 Detailed Description

Class that models an automatic Run.

Author

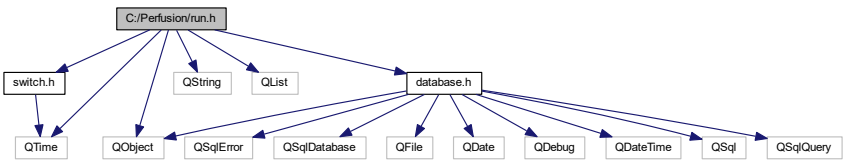
Hubert Tockner

5.14 C:/Perfusion/run.h File Reference

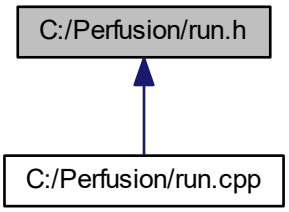
Header File for the Run Class.

```
#include <QObject>
#include "switch.h"
#include <QString>
#include <QList>
#include <QTime>
#include <database.h>
```

Include dependency graph for run.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Run

5.14.1 Detailed Description

Header File for the Run Class.

Author

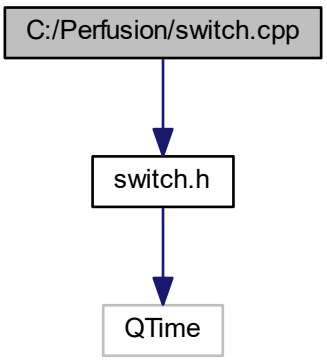
Hubert Tockner

5.15 C:/Perfusion/switch.cpp File Reference

Class that holds a single Switch.

```
#include "switch.h"
```

Include dependency graph for switch.cpp:



5.15.1 Detailed Description

Class that holds a single Switch.

Author

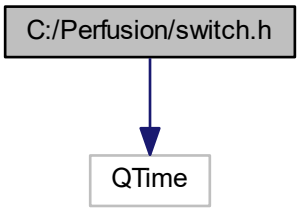
Hubert Tockner

5.16 C:/Perfusion/switch.h File Reference

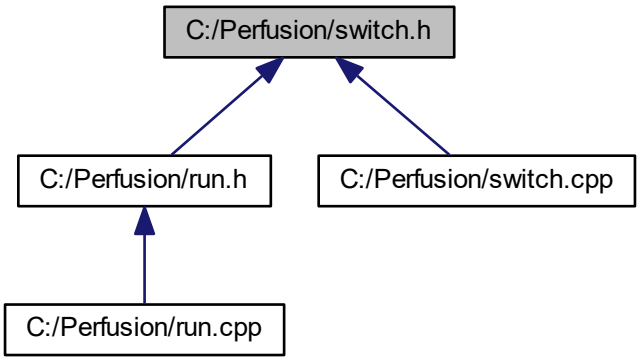
Header File for the Switch Class.

```
#include <QTime>
```

Include dependency graph for switch.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Switch

5.16.1 Detailed Description

Header File for the Switch Class.

Author

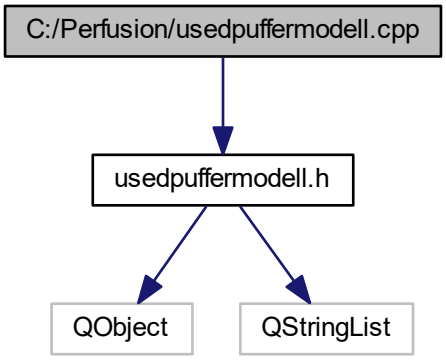
Hubert Tockner

5.17 C:/Perfusion/usedpuffermodell.cpp File Reference

Class that models the Currently Used Puffer.

```
#include "usedpuffermodell.h"
```

Include dependency graph for usedpuffermodell.cpp:



5.17.1 Detailed Description

Class that models the Currently Used Puffer.

Author

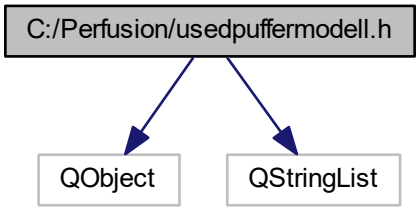
Hubert Tockner

5.18 C:/Perfusion/usedpuffermodell.h File Reference

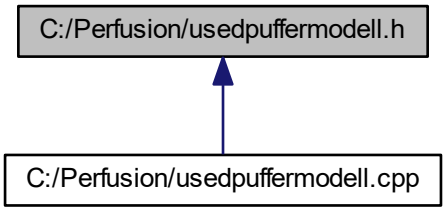
Header File for the UsedPufferModell.

```
#include <QObject>
#include <QStringList>
```

Include dependency graph for usedpuffermodell.h:



This graph shows which files directly or indirectly include this file:



Classes

- class UsedPufferModell

5.18.1 Detailed Description

Header File for the UsedPufferModell.

Author

Hubert Tockner

Index