Simulation eines Eiswasserspeichers

Projekt im Rahmen des Moduls *Mikrocontroller Anwendungen*

Toni Pohl Lukas Werner

24. März 2016

Inhaltsverzeichnis

1 Einleitung **3**

2 Szenario **4**

2.1 Eiswasserspeicher 4

2.2 Lösung 5

3 Raspberry PI **6**

3.1 Betriebssystem 6

3.2 GPIO 7

3.3 Boost 8

4 Aufbau **9**

5 Software **11**

5.1 Modell 11

5.1.1 Laden 11

5.1.2 Kühlen 11

5.2 Simulator 12

5.2.1 Logische Einheiten des Simulators 12

5.2.2 Kompilieren des Simulators 14

5.2.3 Konfiguration des Simulators 14

5.3 Steuer-Client 15

5.3.1 Konfiguration des Steuer-Clients 16

5.3.2 Kompilieren des Steuer-Clients 16

6 Zusammenfassung **17**

6.1 Probleme 17

6.2 Ausblick 18

1 Einleitung

Nachhaltigkeit spielt in der heutigen Zeit eine wichtige Rolle. Neben einer Reduktion des Stromverbrauchs in privaten Haushalten, interessieren sich auch Firmen für Möglichkeiten regenerative Energien zu nutzen. In dieser Arbeit soll für einen Landwirtschaftsbetrieb, der sich auf die Milchproduktion spezialisiert hat, ein Simulator erstellt werden, der feststellen soll, ob die Verwendung eines Eiswasserspeichers in Kombination mit einer Photovoltaik (PV) Anlage sinnvoll ist. Dies soll den eigentlichen Energieverbrauch der Milchkühlung reduzieren, indem der Eiswasserspeicher vorgeschaltet wird. Er soll die Milch für den eigentlichen Kühlvorgang vorkühlen und dadurch den Energieaufwand reduzieren.

2 Szenario

Im Landwirtschaftsbetrieb in Fuchshain wird ein Kühlsystem verwendet, um die Milch auf die richtige Temperatur zu kühlen. Dabei gelangt die gewonnene Milch in einen Milchtank, welche dann herunter gekühlt wird. In den eigentlichen Kühlvorgang kann nicht eingegriffen werden, weswegen die Vorkühlung der Milch vor dem Milchtank durch den Eiswasserspeicher realisiert werden muss. Dieser Eiswasserspeicher soll durch eine vorhandene PV Anlage gespeist werden.
Während der Produktion muss die Milch von 35 auf 4 Grad Celsius abgekühlt werden. Diese Kühlmaßname ist energieaufwändig (304342,5 kJ bei 2500 Liter Milch) [5]. Damit die Hauptkühlung entlastet werden kann, soll die Effizienz einer Vorkühlung durch einen Eiswasserspeicher ermittelt werden. Dieser Speicher würde eine Abflachung der Lastspitzen ermöglichen und somit dem Betrieb Geld einsparen. Da die Anschaffung eines Eiswasserspeichers kostspielig ist, soll vorerst durch eine Simulation die Rentabilität ermittelt werden. Dabei wird angenommen, dass eine Vorkühlung der Milch von 35 auf 17 Grad Celsius stattfindet.

2.1 Eiswasserspeicher

Der Eiswasserspeicher kann 164 kg Eis speichern. Er verfügt weiterhin über einen Kompressor der Marke Maneurop MT-22 mit 3,51 kW Leistung. Der Kompressor ist für die Erzeugung des Eises verantwortlich. Weiterhin befindet sich im Speicher die horzontale Kreiselpumpe CEA 70/3/A-V der Firma LOWARA. Die Ladezeit für eine komplette Beladung mit Eis wird mit sechs Stunden angegeben [5].
Abbildung 2.1 zeigt den Verbrauch der Anlage am 1. Februar 2015 ohne den Betrieb eines Eiswasserspeichers. Die zwei hohen Ausschläge geben die Leistung der zwei Kühlanlagen wieder. Diese Last soll durch einen Eiswasserspeicher gemindert werden, indem eine Vorkühlung stattfindet. In Kombination mit der PV Anlage erzeugt der Eiswasserspeicher einen Energiegehalt zum Vorkühlen, welche die eigentliche Last der Hauptkühlung mindern soll.

Abbildung 2.1: Verbrauch 1. Februar 2015

2.2 Lösung

Für die Realisierung des Simulators wurde ein Raspberry PI zur Verfügung gestellt. Dieser soll softwareseitig alle 15 Minuten einen Simulationsschritt durchführen. In einem Schritt wird festgestellt, ob der Speicher beladen bzw. entladen wird. Während den Schritten wird die verrichtete elektrische Arbeit anhand einer S0-Schnittstelle übertragen. Die Leistung für den Ent- und Beladevorgang wurden vorgegeben und sind als konstant anzusehen.

Die S0-Schnittstelle ist nach DIN EN 62053-31 spezifiziert und dient zur Übertragung von Verbrauchsmesswerten. Häufiger Einsatz ist die Gebäudeautomatisierung und findet in einer Vielzahl von Messgeräten Anwendung. Elektrische Impulse dienen als Maßeinheit für den Verbrauch eines Geräts. Diese Impulse müssen ein gewisses Muster aufweisen, damit sie als gültiger Impuls gewertet werden. Ein Impuls besteht aus einem *HIGH* und einem *LOW* Signal. HIGH muss ebenso wie LOW mindestens 30 Millisekunden lang sein und die auf bzw. absteigende Flanke muss kleiner als fünf Millisekunden sein. Weiterhin entsprechen in diesem Projekt 1000 Impulse einer verrichteten elektrischen Arbeit von 1000 Wattstunden.

3 Raspberry PI

Mit dem Raspberry PI bietet die Raspberry PI Foundation einen Einplatinencomputer, der für Experimente und die Entwicklung von Programmen geeignet ist. Sein günstiger Preis von weniger als 40 Dollar und die Größe einer Kreditkarte tragen zu seinem Erfolg bei. Weiterhin existieren verschiedene Varianten des Computers, welche sich hauptsächlich in der Rechenleistung unterscheiden. Ursprünglich sollte der Raspberry PI zum Experimentieren von Studenten verwendet werden, fand jedoch schnell Anklag in divseren anderen Bereichen außerhalb von Universitäten, unter anderem der Automatisierung von Abläufen [1].
Für diese Arbeit wurde ein Raspberry PI 2 vorgeschlagen, da er eine kostengünstige Alternative darstellt, um hardwarenahe Software zu verwirklichen. Hierbei spielen besonders die General Purpose Input/Output Pins eine wichtige Rolle. Weiterhin verfügt der Raspberry PI über eine Ethernet Schnittstelle, welche zur Kommunikation dient. Da es sich um einen Einplatinencomputer und nicht um einen klassischen Mikrocontroller handelt, findet sich auf dem Raspberry PI ein Betriebssystems wieder, welches eine Erleichterung für die Softwareentwicklung darstellt. Hierzu zählt insbesondere die Umsetzung des TCP/IP Stacks.

3.1 Betriebssystem

Die Betriebssysteme für den Raspberry PI sind zahlreich. Am häufigsten ist jedoch eine Linux-Distribution anzutreffen. In diesem Projekt kommt Raspbian zum Einsatz, ein auf Debian angepasstes Betriebssystem. Die Installation von Raspbian findet mit dem NOOBS-Installer, ein von der Raspberry PI Foundation bereitgestellter Installationsassistent, statt [2]. Das Betriebssystem vereinfacht die Softwareentwicklung dahingehend, dass es verschiedene Schnittstellen für den Zugriff auf die Hardware anbietet. Die für dieses Projekt am wichtigsten Schnittstellen sind einerseits die GPIO Pins und andererseits der Zugriff auf das Netzwerk.

3.2 GPIO

Von den insgesamt 40 Pins sind 26 GPIO Pins. Bei den anderen handelt es sich Pins für die Stromversorgung. Abbildung 3.1 zeigt die Belegung der Pins des Raspberry PI 2. Durch die GPIO Pins wird in diesem Projekt die benötigte S0-Schnittstelle implementiert. Diese Schnittstelle sendet Impulse, welche einen Leistungswert repräsentieren.

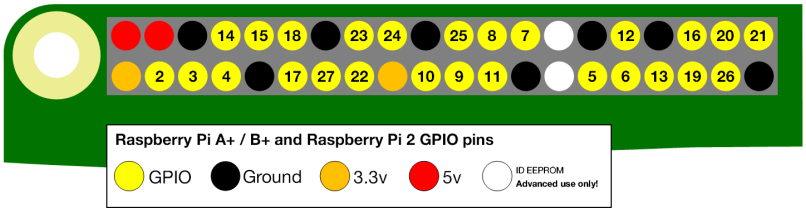

Abbildung 3.1: Pin Belegung Raspberry PI 2

Damit die GPIO Pins in einem C/C++ Programm angesprochen werden können, wird die Bibliothek *Wiring Pi*[1] verwendet [3]. Die Nummerierung der GPIO Pins unterschiedet sich zu der ursprünglichen Anordnung und ist aus Abbildung 3.2 ersichtlich.

P1: The Main GPIO connector							
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin
		3.3v	1	2	5v		
8	Rv1:0 - Rv2:2	SDA	3	4	5v		
9	Rv1:1 - Rv2:3	SCL	5	6	0v		
7	4	GPIO7	7	8	TxD	14	15
		0v	9	10	RxD	15	16
0	17	GPIO0	11	12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13	14	0v		
3	22	GPIO3	15	16	GPIO4	23	4
		3.3v	17	18	GPIO5	24	5
12	10	MOSI	19	20	0v		
13	9	MISO	21	22	GPIO6	25	6
14	11	SCLK	23	24	CE0	8	10
		0v	25	26	CE1	7	11
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin

Abbildung 3.2: Wiring Pi Pinanordnung

Listing 3.1 zeigt ein einfaches Programm, welches *Wiring Pi* verwendet. Zeile vier zeigt eine Initialiserungsprozedur, welche einmalig aufgerufen werden muss, um auf die GPIO Pins zuzugreifen. Anschließend wird in Zeile fünf angegeben, ob auf dem Pin 0 lesend oder schreibend zugegriffen wird. Anschließend wird in einer Endlosschleife abwechselnd der Pin auf *HIGH* oder *LOW* gesetzt, jeweils mit einer Pause von 500 Millisekunden.

[1] http://wiringpi.com

```
#include <wiringPi.h>
int main (void)
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;)
  {
    digitalWrite (0, HIGH) ; delay (500) ;
    digitalWrite (0,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

Listing 3.1: WiringPi Beispiel

3.3 Boost

Die C++ Bibliothek *Boost*[2] bietet diverse Funktionalitäten für die Entwicklung an. In diesem Projekt wird *Boost* dazu verwendet, um auf das Netzwerk zuzugreifen, Nebenläufigkeit zu ermöglichen und um Konfigurationsdateien einzulesen. Die Netzwerkfunktionalitäten werden für eine Client-Server Architektur verwendet, um den Simulator zu steuern. Der Client sendet Kommandos, um die Simulation zu beeinflussen. Dadurch lässt sich einstellen, ob gekühlt oder geladen wird.
Threads für die Nebenläufigkeit finden bei den einzelnen Komponenten des Programm Anklang. Da die Komponenten quasi parallel laufen müssen werden verschiedene Threads benötigt. Das Verarbeiten der Kommandos des Clients, die Simulation selbst und die Übertragung der Leistung anhand der S0-Schnittstelle laufen je in einem eigenen Thread.

[2]http://www.boost.org

4 Aufbau

Abbildung 4.1 zeigt den physischen Aufbau des Projekts. Auf der einen Seite ist der Raspberry PI erkennbar. Auf der anderen Seite befindet sich ein Arduino UNO, welcher einen Datenlogger darstellen soll. Da zur Zeit der Entwicklung der explizit dafür vorgesehene Datenlogger nicht verfügbar war, wurden mittels eines Arduino UNO dessen Funktionalitäten nachgebaut. Der Arduino UNO zählt die ankommenden Impulse und summiert diese auf. Per Konsole werden die Werte an den Entwickler gegeben, sodass er testen kann, ob die gewünschte Anzahl von Impulsen angekommen ist. Im Betrieb soll jedoch ein richtiger Datenlogger zum Einsatz kommen.
Den Kern des Aufbaus bildet der Optokoppler vom Typ KB Knighbright KB 817. Der Optokoppler trennt die beiden Stromkreise galvanisch voneinander. Diese Trennung sichert den Raspberry PI gegen zu hohe Spannungen ab und verhindert somit seine Zerstörung.

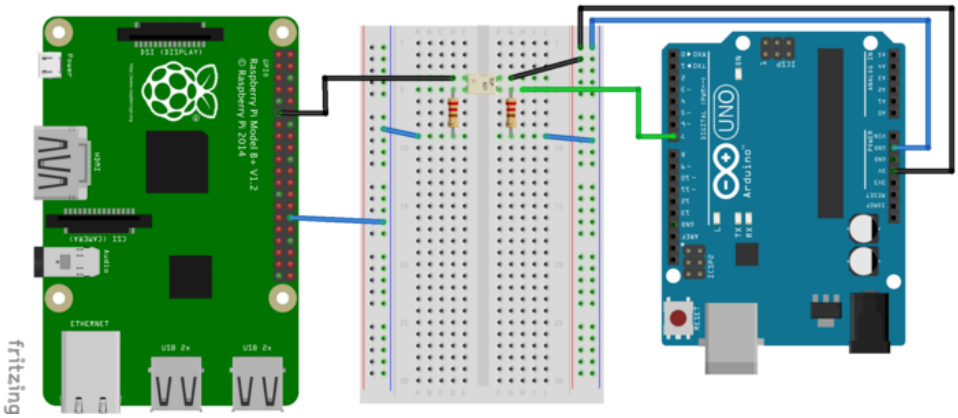

Abbildung 4.1: Schematischer Aufbau

Im eigentlichen Betrieb läuft der Simulator mit dem e.manager der Firma enerserve. Der e.manager erlaubt es Anlagen zu überwachen und deren Verbrauch zu messen. Dies geschieht über diverse Schnittstellen, darunter auch die S0-Schnittstelle. Da der e.manager mit einer Spannung von 12 Volt betrieben wird, muss dieser galvanisch vom Raspberry

PI getrennt werden, da dieser mit 5 Volt betrieben wird. Dies geschieht mit dem bereits erwähnten Optokoppler. Abbildung 4.2 zeigt den Aufbau mit dem Raspberry PI und dem e.manager.

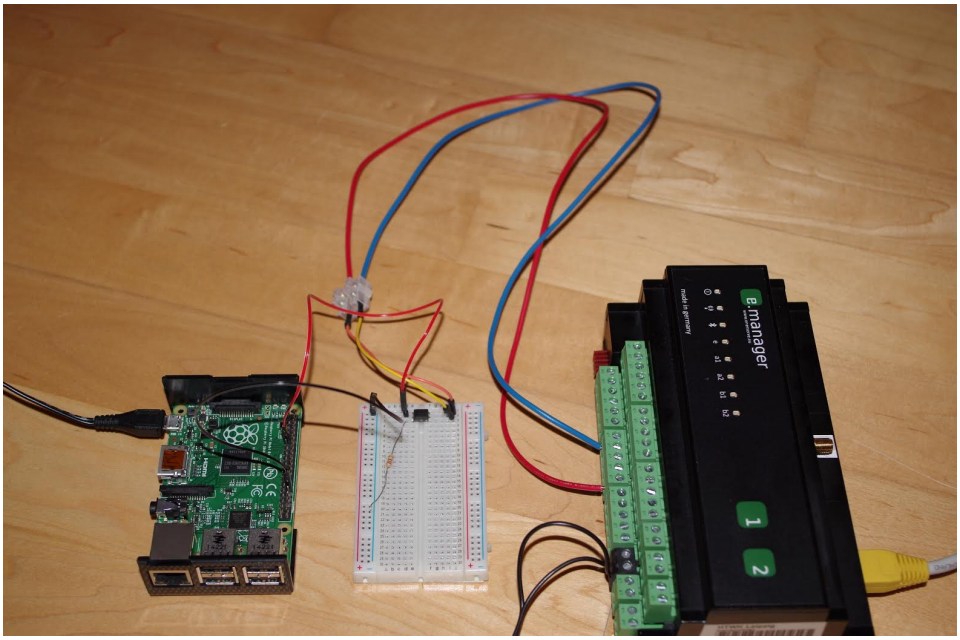

Abbildung 4.2: Aufbau mit e.manager

Die ermittelten Daten des e.managers können über ein Webinterface abgerufen werden. Das Portal von enerserve (http://portal.enerserve.eu) erlaubt es dem Benutzer den Verlauf der einzelnen Geräte zu verfolgen. Neben einer Übersicht des aktuellen Verbrauchs, liefert das Webinterface auch historische Daten. Dabei wird zwischen Tages-, Monats- und Jahresansicht unterschieden. Weiterhin bietet das Interface eine Konfiguration der Anschlüsse.

5 Software

5.1 Modell

Wenn ein Eiswasserspeicher zum Vorkühlen von Milch eingesetzt werden soll, laufen dabei komplexe physikalische Vorgänge ab, die nur schwer mathematisch zu beschreiben sind. Deshalb war als Grundlage für den Simulator ein physikalisches Modell gegeben, das diesen Prozess näherungsweise abbildet [5]. Im Folgenden wird der Energiegehalt Q des Eiswasserspeichers betrachtet und auf Basis dessen die Berechnungen ausgeführt.

5.1.1 Laden

Das Erzeugen von Eis durch die Kältemaschine innerhalb des Eiswasserspeichers wird im Rahmen des Projektes als *Laden* bezeichnet. Die Kältemaschine soll von außen ein- und ausgeschaltet werden können.
Seien Q_s der aktuelle Energiegehalt in kJ, $Q_{s_{max}}$ der maximale Energiegehalt in kJ, m_s die Speicherkapazität des Eiswasserspeichers in kg, w_e die Schmelzwärme von Eis in kJ/kg, t_l die Regenerationszeit in Stunden und t_d die Dauer eines Simulationsschrittes in Minuten. Die Regeneration pro Simulationsschritt Q_l in kJ lässt sich nun durch die Formel 5.1 berechnen.

$$Q_l = \frac{Q_{s_{max}} t_d}{60 t_l} = \frac{m_s w_e t_d}{60 t_l} \tag{5.1}$$

Das Ergebnis dieser Berechnung wird in jedem Simulationsschritt zum aktuellen Energiegehalt des Eiswasserspeichers addiert. Die dafür aufgewendete elektrische Arbeit wird als konstant angenommen und liegt bei 3,57 kWh [5].

5.1.2 Kühlen

Wenn die Kreiselpumpe angeschaltet ist, wird das Eiswasser umgewälzt und somit die Milch gekühlt. Auch dieser Prozess soll von außen gestartet und gestoppt werden können.

Seien c_p die spezifische Wärmekapazität der Milch $\frac{kJ}{kg \cdot K}$, m_1 und m_2 die Volumenströme der Vakuumpumpen, die die Milch durch den Eiswasserspeicher führen in l/min, T_m die Eingangstemperatur, T_w die Ausgangstemperatur der Milch in Grad Celsius und ρ_m deren Dichte. Dann beschreibt Q_w die Abnahme des Energiegehaltes pro Simulationsschritt. Formel 5.2 zeigt die Berechnung von Q_w.

$$Q_w = t_d \rho_m c_p (m_1 + m_2)(T_m - T_w) \tag{5.2}$$

Dieser Betrag wird pro Simulationsschritt von der aktuellen Kühlleistung des Speichers subtrahiert. Die dafür aufgewendete elektrische Arbeit wird als konstant angenommen und liegt bei 0,5 kWh [5].

5.2 Simulator

Um das physikalische Modell wie im vorherigen Abschnitt beschrieben abzubilden, wurde im Rahmen dieses Projektes eine C++-Applikation entwickelt. Diese simuliert die physikalischen Prozesse des Eiswasserspeichers, läuft zeitdiskret ab, ist konfigurierbar und bietet verschiedene Schnittstellen nach außen an.

5.2.1 Logische Einheiten des Simulators

Der Simulator lässt sich in verschiedene logische Einheiten unterteilen, welche in den nachfolgenden Teilabschnitten genauer betrachtet werden sollen. Eine übergeordnete Einheit (Klasse *Simulator*) dient als Koordinator für den zeitdiskreten Ablauf.

Eiswasserspeicher

Der Eiswasserspeicher ist die zentrale Einheit im Simulator. Er setzt das komplette physikalische Modell um und wird durch die Klasse *Reservoir* repräsentiert. Diese besitzt eine Methode `load()` zum Laden des Speichers, sowie eine Methode `cool()` zum Kühlen. Weiterhin gibt es eine Methode `step()` zum Ausführen eines zeitdiskreten Simulationsschrittes. Diese entscheidet anhand des aktuellen Zustands, ob die Methoden zum Laden und Kühlen ausgeführt werden sollen. Um den Zustand zu ändern, gibt es die beiden Methoden `toggleLoading()` und `toggleCooling()`, womit das Laden respektive das Kühlen an- und ausgeschaltet werden kann.

Steuer-Server

Der Simulator soll wie in Abschnitt 5.2 erläutert lediglich den Teil des Eiswasserspeichers übernehmen. Das heißt, dass es eine Schnittstelle geben soll, über die das Laden und Kühlen von außen an- und ausgeschaltet werden kann. Diese Schnittstelle bietet der Steuer-Server an, der mit den Klassen *ControlServer* und *TcpSession* realisiert wurde. Mit ihm kann sich ein Client (siehe Abschnitt 5.3) verbinden und mit entsprechenden Befehlen den Zustand des Eiswasserspeichers verändern.

S0-Schnittstelle

Die S0-Schnittstelle des Simulators richtet sich nach der in Abschnitt 2.2 vorgestellten Definition. Sie läuft permanent und prüft in jedem Schritt, ob aktuell gekühlt bzw. geladen wird. Dementsprechend werden Impulse in regelmäßigen Abständen über den entsprechenden Pin am Raspberry PI gesendet.
Ein solcher Impuls ist in idealisierter Form in Abbildung 5.2.1 dargestellt. Er besteht aus einer *HIGH*-Phase mit Stromfluss und einer *LOW*-Phase ohne Stromfluss. Zwischen den zwei Phasen muss laut Spezifikation immer eine Zeit von mindestens 30 ms gewartet werden.

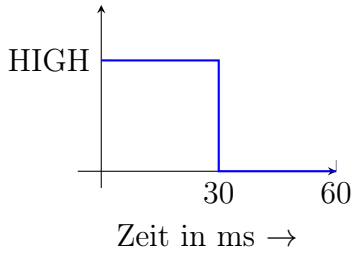

Abbildung 5.1: Idealisierter S0-Impuls

Die Anzahl der Impulse pro Zeiteinheit werden durch eine Wartezeit t_{S0} zwischen zwei Impulsen bestimmt. Hierzu müssen die Anzahl der Impulse pro Leistungsverbrauch n_{S0} und die Dauer einer Impulsphase t_p gegeben sein. Weiterhin müssen die Zeitdauer eines Simulationsschrittes in Millisekunden t_d bekannt sein und die verrichtete elektrische Arbeit Q_w berechnet werden. Die Berechnung von t_{S0} wird dann wie in Formel 5.3 durchgeführt.

$$t_{S0} = \frac{t_d}{n_{S0} Q_w - 2t_p} \tag{5.3}$$

5.2.2 Kompilieren des Simulators

Um den Simulator zu kompilieren, gibt es einige Systemvoraussetzungen, die zunächst erfüllt werden müssen. Da die Applikation unter Linux (siehe Abschnitt 3.1) laufen soll, empfiehlt es sich, diese auch unter demselben Betriebssystem zu übersetzen. Zunächst wird ein C++-Compiler benötigt; in diesem Projekt wurde die *GNU Compiler Collection* (kurz: GCC) verwendet. Diese ist unter Linux in der Regel in den Paketquellen zu finden. Alternativ kann man sie auch manuell installieren[1]. Anschließend werden noch die Programmbibliotheken *Boost* und *Wiring Pi* benötigt. Sind alle Voraussetzungen erfüllt, lässt sich die Applikation mit dem Befehl `make` kompilieren.

5.2.3 Konfiguration des Simulators

Viele der Werte, die in Abschnitt 5.1 für das physikalische Modell verwendet werden, sind veränderlich. So unter anderem der Zeitschritt des Simulationsverfahren oder das Fassungsvermögen des zu simulierenden Eiswasserspeichers. Um dies zu ermöglichen, sollte die Applikation konfigurierbar sein, ohne diese stets neu kompilieren zu müssen. Um dies zu ermöglichen, werden sogenannte *Initialisierungsdateien* (kurz: INI-Dateien) verwendet. Dies sind einfache Textdateien, deren Struktur aus Abschnitten, Eigenschaften und Werten besteht [4]. Zum Einlesen und Parsen der INI-Dateien wird die *Boost program_ options* Bibliothek verwendet.
Ein einfaches Beispiel einer Konfigurationsdatei ist in Listing 5.1 zu sehen.

```
1  [FooBar]
2  prop = foobar
3  prop.foo = foo
4  prop.bar = bar
5  [Baz]
6  other = value
```

Listing 5.1: INI-Datei Beispiel

Die Konfigurationsmöglichkeiten des Simulators sind in Tabelle 5.1 aufgelistet, dabei sind sämtliche Zahlenwerte als *Integer* anzugeben.

[1] https://gcc.gnu.org/install

Schlüssel	**Mögliche Werte**	**Beschreibung**
controlserver.port	$1024 < \text{port} < 65535$	Port des Steuer-Servers
controlserver.secret.token	beliebig	Geheimer Schlüssel
milk.temp.target	> 0	Zieltemperatur der Milch
milk.temp.input	> 0	Eingangstemperatur der Milch
simulator.time.step	> 0	Zeitschritt in Minuten
simulator.log.level	>= 10 für *ERROR* >= 20 für *WARN* >= 30 für *INFO* >= 40 für *DEBUG* >= 50 für *TRACE*	Log-Level des Simulators
simulator.debug	true/false	Falls true, wird der Zeitschritt auf 10 Sekunden abgesenkt
snull.pin	S. Abschnitt 3.2	Pin für den Ausgang zur S0-Schnittstelle
snull.watt.per.pulse	> 0	Anzahl der Pulse pro Watt
snull.watt.per.load	> 0	Leistung beim Laden
snull.watt.per.cool	> 0	Leistung beim Kühlen
reservoir.capacity	> 0	Kapazität des Speichers
reservoir.loadingtime	> 0	Ladezeit in Stunden
reservoir.pumps.flow	> 0	Volumenstrom der Pumpen in l/min

Tabelle 5.1: Konfiguration des Simulators

5.3 Steuer-Client

Um den im vorherigen Abschnitt vorgestellten Steuer-Server einfach und sicher bedienen zu können, wurde im Rahmen dieses Projektes der *Steuer-Client* entwickelt. Dies ist eine einfache C++-Applikation, die unabhängig vom Simulator gestartet und bedient werden

kann. Dabei ist es möglich, dass Simulator und Steuer-Client auf physisch getrennten Maschinen laufen, da diese über TCP/IP miteinander kommunizieren. Dies wird bereits durch den Raspberry PI und das darauf laufende Betriebssystem sichergestellt (siehe Kapitel 3).

5.3.1 Konfiguration des Steuer-Clients

Auch der Steuer-Client ist über eine mitgelieferte INI-Datei konfigurierbar. Hierzu wird ebenfalls die *Boost program_ options* Bibliothek verwendet. Tabelle 5.2 zeigt die Konfigurationsmöglichkeiten des Steuer-Clients.

Schlüssel	**Mögliche Werte**	**Beschreibung**
server.host	beliebig	Hostname bzw. IP des Rechners, auf dem der Simulator läuft
server.port	$1024 < \text{port} < 65535$	Port des Rechners, auf dem der Simulator läuft
server.secret.token	beliebig	Geheimer Schlüssel des Servers

Tabelle 5.2: Konfiguration des Steuer-Clients

5.3.2 Kompilieren des Steuer-Clients

Der Steuer-Client lässt sich nahezu analog zum Simulator kompilieren (siehe Abschnitt 5.2.2). Einzig die *Wiring Pi* Bibliothek wird nicht benötigt. Wurden alle Voraussetzungen für den Simulator bereits geschaffen, so lässt sich der Steuer-Client ebenfalls mit dem Befehl `make` kompilieren.

6 Zusammenfassung

Die Ziele im Rahmen des Projektes wurden erreicht. Das heißt, es wurde erfolgreich eine Simulationseinheit umgesetzt, die sowohl software- als auch hardwareseitig den Anforderungen entspricht. Sie kann auf einem Raspberry PI installiert in eine Kühlanlage integriert werden und somit Daten für den Stromverbrauch eines Eiswasserspeichers sowie dessen Einfluss auf ein solches System erstellen. Mithilfe des Steuer-Servers können Kältemaschine für das Laden und Kreiselpumpe für das Kühlen unabhängig voneinander über das Internet bedient, d.h. an- und ausgeschaltet werden. Der Koordinator sorgt für den zeitdiskreten Ablauf und der Speicher führt das zugrunde liegende physikalische Modell aus. Es ist im Moment nicht möglich zu beurteilen, wie realistisch die Daten sind, die der Simulator generiert, da der direkte Vergleich mit einem realen System im Rahmen des Projektes nicht möglich war.

6.1 Probleme

Während der Umsetzung des Projektes traten auch diverse Probleme auf, die nachfolgend erläutert werden sollen. Eine größere Einschränkung war durch relativ hohe Antwortzeiten beim E-Mail Verkehr gegeben. Hier musste teilweise mehrere Tage auf wichtige Antworten und Hinweise gewartet werden. Auch Terminabsprachen waren somit nicht kurzfristig möglich. Ein weiteres Problem war die späte Ausgabe der Hardware, d.h. des Raspberry PI und des Datenloggers. So konnte erst relativ spät mit der konkreten Umsetzung der gesetzten Lösung begonnen werden. Außerdem waren dadurch keine Langzeittests möglich, durch die man eventuell aufschlussreichere Informationen über die Einsatzfähigkeit der Lösung hätte bekommen können. Hierbei ist dann zu Ende des Projektes aufgefallen, dass sich die Messwerte des Datenloggers leicht von denen, die vom Simulator gesendet werden, unterscheiden. Aufgrund fehlender Zeit konnte dies nicht weiter verfolgt und gelöst werden.

6.2 Ausblick

Es gibt noch verschiedene Möglichkeiten, weitere Erweiterungen in den Simulator einzubauen. In der aktuellen Verfassung kann beispielsweise nicht entschieden werden, ob die Vakuumpumpen an- oder ausgeschaltet sind. Es wird somit davon ausgegangen, dass diese dauerhaft an sind, was das Ergebnis leicht verfälschen könnte. Weiterhin ist das Modell im Moment noch sehr ungenau, da viele Parameter und Variablen als konstant angesehen werden. So könnte man unter anderem den Stromverbrauch des Eiswasserspeichers mit einem realistischeren Modell berechnen oder die Außentemperatur mit in die Berechnungen einbeziehen.

Literaturverzeichnis

[1] Raspberry PI Foundation. https://www.raspberrypi.org/about (22.03.2016).

[2] Raspberry PI Foundation. https://www.raspberrypi.org/help/noobs-setup (22.03.2016).

[3] Wiring Pi, GPIO Interface library for the Raspberry Pi. http://wiringpi.com (21.03.2016).

[4] Microsoft, Configure an Ini File Item. https://technet.microsoft.com/en-us/library/cc731332.aspx (23.03.2016).

[5] Kusow, Andreas, ohne Titel, Hochschule für Technik, Wirtschaft und Kultur, Leipzig, 2015.