Inhaltsverzeichnis

1 Einleitung

Nachhaltigkeit spielt in der heutigen Zeit eine wichtige Rolle. Neben einer Reduktion des Stromverbrauchs in privaten Haushalten, interessieren sich auch Firmen für eine Möglichkeit regenerative Energien zu nutzen. In dieser Arbeit soll für einen Landwirtschaftsbetrieb, der sich auf die Milchproduktion spezialisiert hat, ein Simulator erstellt werden. Dieser Simulator soll feststellen, ob die Verwendung eines Eiswasserspeichers in Kombination von einer Photovoltaik (PV) Anlage sinnvoll ist.

1.1 Problemstellung

Während der Produktion muss die Milch von 35 auf 4 Grad Celsius abgekühlt werden. Diese Kühlmaßnahme benötigt viel Energie (304342,5 kJ bei 2500 Liter Milch). Ein Simulator eines Eiswasserspeichers soll ermitteln, ob eine Abflachung der Lastspitzen durch eine Vorkühlung möglich ist. Die Vorkühlung soll die Milch von 35 auf 17 Grad Celsius runterkühlen.

1.2 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.

1.3 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 oder entladen wird. Während

den Schritten wird die benötigte Leistung anhand einer S0-Schnittstelle übertragen. Die Leistung für den Ent- und Beladevorgang wurden vorgegeben.5

2 Raspberry PI

Mit dem Raspberry PI bietet die Raspberry PI Foundation einen Einplatinencomputer bereit, der für Experimente und 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.
Für diese Arbeit wurde ein Raspberry PI 2 vorgeschrieben, da er eine kostengünstige Alternative darstellt, um hardwarenahe Software zu verwirklichen. Hierbei tragen 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 TCPI/IP Stacks.

2.1 Betriebssystem

Die Betriebssysteme für den Raspberry PI sind zahlreich. Am häufigsten ist jedoch eine Linux-Distribution anzutreffen. In diesem Projekt kommt Raspbian, ein auf Debian angepasstes Betriebssystem. Die Installation von Raspbian findet mit dem NOOBS-Installer, ein von der Raspberry PI Foundation bereitgestellter Installationsassistent, statt. 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.

2.2 GPIO

Von den insgesamt 40 Pins sind 26 GPIO Pins. Bei den anderen handelt es sich Pins für die Stromversorgung. Abbildung 2.1 zeigt die Belegung der Pins des Raspberry PI 2. Durch die GPIO Pins wird in diesem Projekt eine S0-Schnittstelle implementiert. Diese Schnittstelle sendet Impulse, welche einen Leistungswert repräsentieren. Als Beispiel könnten 1000 Impulse die Leistung von 1 kW darstellen.

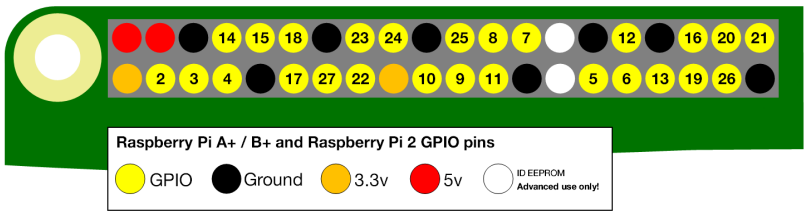

Abbildung 2.1: Pin Belegung Raspberry PI 2

Damit die GPIO Pins in einem C/C++ Programm angesprochen werden können, wird die Bibliothek *WiringPi* verwendet. Die Nummerierung der GPIO Pins unterschiedet sich zu der ursprünglichen Anordnung und ist aus Abbildung 2.2 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 2.2: WiringPi Pinanordnung

Listing 2.1 zeigt ein einfaches Programm, welches *WiringPi* 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 null 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.

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

Listing 2.1: WiringPi Beispiel

2.3 Boost

Die C++ Bibliothek *boost* bietet diverse Funktionalitäten für die Entwicklung an. In diesem Projekt wird *boost* dazu verwendet, um auf das Netzwerk zuzugreifen und um Nebenläufigkeit zu ermöglichen.

3 Schaltung

Abbildung 3.1 zeigt den Schaltplan der 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 Datenlogger nicht vorhanden war, wurde 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.
Der Kern des Schaltplans bildet der Optokoppler vom Typ KB Knighbright KB 817. Der Optokoppler trennt die beiden Stromkreise voneinander galvanisch. Diese Trennung sicheret den Raspberry PI gegen hohen Spannungen ab und verhindert somit seine Zerstörung.

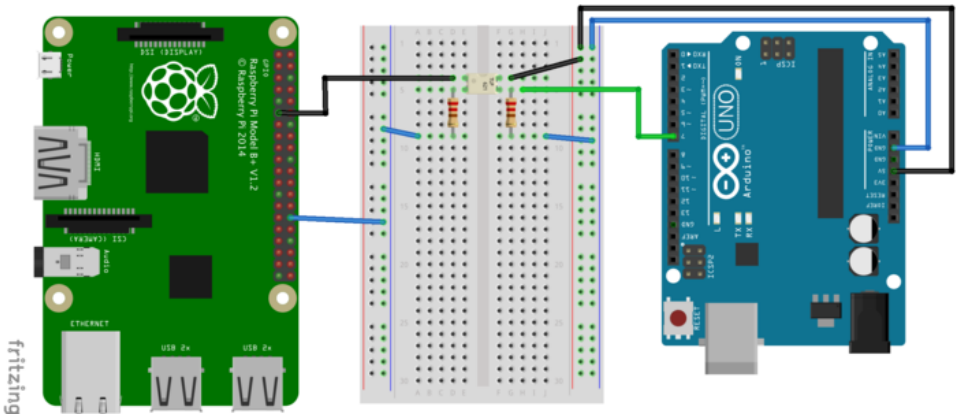

Abbildung 3.1: Schaltung

4 Software

4.1 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. Dieses soll im Folgenden erläutert werden.

4.1.1 Laden

4.1.2 Kühlen

4.2 Simulator

Um das physikalische Modell wie im vorherigen Abschnitt beschrieben abzubilden, wurde im Rahmen dieses Projektes eine komplexe C++-Applikation entwickelt. Diese läuft zeitdiskret ab, ist konfigurierbar und bietet verschiedene Schnittstellen nach außen an. Sie lässt sich in verschiedene logische Einheiten unterteilen, welche in den nachfolgenden Teilabschnitten genauer betrachtet werden sollen.

4.2.1 Eiswasserspeicher

Der Eiswasserspeicher ist die zentrale Einheit im Simulator. Er setzt das komplette mathematische Modell um.

4.2.2 Steuer-Server

Der Simulator soll wie in Abschnitt 4.2 erläutert lediglich den Teil des Eiswasserspeichers übernehmen. Das heißt, dass es eine Schnittstelle

4.2.3 Konfiguration des Simulators

Die Konfigurationsmöglichkeiten sind in Tabelle 4.1 aufgelistet, dabei sind sämtliche Zahlenwerte sind als *Integer* anzugeben.

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 2.2	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 4.1: Konfiguration des Simulators

4.3 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 2).

4.3.1 Konfiguration des Steuer-Clients

Auch der Steuer-Client ist über eine mitgelieferte INI-Datei konfigurierbar. Hierzu wird ebenfalls die *Boost program_ options* verwendet. Tabelle 4.2 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 4.2: Konfiguration des Steuer-Clients

5 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.