Programm Aufbau und Übersicht

In diesem Abschnitt wird auf das geschriebene Softwareprogramm und die Konfiguration des Rasbery-Pi's eingegangen.

Anforderung

Mit Rasbery-Pi, RS232 Erweiterungs-Platine und AD/DA Expansion Board soll die Thermokammer auf vorgegebene Temperaturen geregelt werden.

Konfiguration

Auf dem Rasbery-Pi wurde das Raspbian Betriebssystem installiert. In diesem wurde die bcm2835-1.52 Bibliothek installiert um das AD/DA Expansion Board in C++ ansprechen zu können.

Um die RS232 Erweiterungs-Platine nutzen zu können, musste im Konfigurationsmenü des Pi's das Serialinterface auf enabled gesetzt werden. Danach wurde die Hardwareschnittstelle ttyAMA0 in /dev aufgeführt. Um diese im Programm nutzen zu können, musste noch der Systemdienst serial-getty@ttyAMA0.service beendet werden, da dieser sonst die Schnittstelle für sich beansprucht und diese blockiert. Hierzu wurde in der ~/.bashrc die folgende Zeile hinzugefügt:

sudo systemctl stop serial-getty@ttyAMA0.service

Dieses Kommando wird nach dem Start des Pi's ausgeführt und beendet den Systemdienst.

Um das erstellte Programm auf dem Pi auszuführen, muss diese mit Admin Rechten gestartet werden. Hierzu wird das sudo-Schlüsselwort vom Terminal aus verwendet.

sudo ./Programm

Zur neuen Kompilierung des Programms kann im Terminal das folgende Kommando verwendet werden:

g++ main.cpp pid.cpp serial.cpp gpio.cpp -std=c++11 -lbcm2835

Erstellte Software

Zur Regelung der Thermokammer wurde ein C++ Programm geschrieben. Dieses wurde in 3 Klassen unterteilt die im folgenden beschrieben werden.

AD/DA Expansion Board Ansteuerung (gpio.cpp)

In der *ADDA_GPIO* Klasse wird das Setup des AD/DA Expansion Boards vorgenommen und die Funktionen bereitgestellt mit denen die Ausgänge angesteuert und die Eingänge ausgelesen werden können.

Die Klasse baut dazu auf dem Beispiel AD-DA_test.c der waveshare.com Wiki auf, welches die bcm2835-1.52 Bibliothek zur Ansteuerung des Boards verwendet.

Das bestehende C Programm wurde in eine C++ Klasse „gpio.cpp" umgeschrieben und mit einer passenden Headerdatei „gpio.h" erweitert.

Nach außen hin werden die öffentlichen Funktionen set_output_voltage() und get_AD_voltage() bereitgestellt, um aktuelle Messwerte zu erfassen und Regelwerte auszugeben.

Im Konstruktor der Klasse wird die Initialisierung des Boardes und der Kanäle durchgeführt. Im Destruktor wird die Verbindung zum Board wieder getrennt.

RS232 Kommunikation (serial.cpp)

Die *Serial* Klasse basiert auf dem Beispiel zur Verwendung des uart Serialports von raspberry-projects.com/pi/programming-in-c/uart-serial-port/using-the-uart.

Hier werden die Funktionen zum Initialisieren, Senden und Auslesen der seriellen Schnittstelle implementiert. Das Beispiel Programm wurde ebenfalls in eine C++ Klasse mit Headerdatei umgeschrieben.

Im Konstruktor wird die serielle Schnittstelle auf dem Pi geöffnet und dabei die folgenden Parameter verwendet: 8 Bit Daten breite, Baud rate = 9600 Bit/s und keine Paritätsprüfung.

Im Destruktor der Klasse wird die serielle Schnittstelle wieder geschlossen.

Die Funktionen zum Senden und Auslesen von Daten wurden so angepasst das diese einfache Strings als Rückgabewert und Parameter verwenden. Damit ist es im weiteren Programm einfacher die empfangenen Daten nach Schlüsselwörtern zu durchsuchen.

Sollte die Funktion zum Auslesen später für längere Strings verwendet werden muss gegebenenfalls das Charakter Array in dieser Funktion in seiner Länge angepasst werden. Dieses ist im Moment 256 Charakter groß.

PID-Regler Implementierung (pid.cpp)

Die PID-Regler Klasse basiert auf dem Beispiel zur Implementierung eins PID-Reglers in C++ von https://gist.github.com/bradley219/5373998. Dieses wurde vereinfacht und mit einem variablen delta t erweitert.

Im Konstruktor der Klasse werden die Regelparameter Maximal- und Minimalregelwert (+ 5V und -5 V) sowie die Reglerkonstanten Kp, Kd und Ki übergeben.

Um den nächsten Regelwert zu ermitteln, wird über die calculate() Funktion der aktuelle Sollwert mit den Istwert verglichen und zusammen mit der verstrichenen Zeit seit der letzten Berechnung verrechnet. Der Rückgabewert der Funktion entspricht dem neuen Regelwert.

Main

Im Main-Teil des Programms werden die 3 zuvor beschriebenen Klassen instantiiert und zur Regelung der Temperatur eingesetzt. Die Regelparameter des PID- Reglers sind hier hinterlegt und werden beim Instantiieren des Reglers übergeben.

In einer do-while Schleife befindet sich der Hauptteil des Programms.

Hier wird die serielle Schnittstelle ausgelesen und bei einer vorhandenen Nachricht, diese auf Schlüsselwörter durchsucht. So kann dem Programm mitgeteilt werden, welche Soll -temperatur einzustellen ist, ob mit PID oder 2 Punkt Regelung gearbeitet werden soll und ob die Messung nun beendet werden kann.

Es wird in jedem Zyklus der Schleife die aktuelle Spannung am Temperatursensor ausgelesen und in die aktuelle Temperatur umgerechnet. Die Umrechnung der gemessenen Spannung in eine Temperatur wird in der convert_voltage_to_temperature() Funktion vorgenommen. Die hierbei verwendeten Umrechnungsfaktoren wurden aus einem Vergleich des Temperatursensors mit einem kalibrierten Sensor berechnet.

Die gemessene Spannung und die daraus berechnete Temperatur sowie die aktuelle Messzeit wird jeweils an einen Vektor angeknüpft.

Mit der gemessenen Temperatur und der berechneten Zeitdifferenz wird die Regelfunktion des PID- Reglers aufgerufen und ein neuer Regelwert berechnet.

Falls die PID-Regelung verwendet wird, wird der berechnete Regelwert zunächst überprüft ob dieser positiv oder negativ ist. Ein positiver Regelwert wird verwendet, um die Wärmepads anzusprechen, ein negativer Regelwert wird als Absolutwert an das Peltieelement weiter gereicht.

Falls die 2 Punkt Regelung eingestellt wurde wird beim Überschreiten des Sollwertes das Peltieelement mit voller Leistung eingeschaltet. Beim Unterschreiten des Sollwertes werden die Wärmepads mit voller Leistung eingeschaltet.

In jedem Programmzyklus wird eine Nachricht über die serielle Schnittstelle an den PC geschickt in der sich die aktuelle Messzeit und Temperatur befindet.

Zum Ende der Main Funktion werden Wärmepads und Peltieelement wieder abgeschaltet und die Messwerte- Vektoren in eine CSV- Datei abgespeichert.

Interaktion über Powershell

Um mit dem erstellten Programm arbeiten zu können, müssen vom PC aus die Messungsparameter über die serielle Schnittstelle an das Pi übertragen werden. Hierzu kann unter Windows die Powershell verwendet werden. Die vom Programm akzeptierten Parameter sowie die Initialisierung wird im Folgenden aufgeführt.

Serienelle Verbindung öffnen:

```
$port= new-Object System.IO.Ports.SerialPort COM1,9600,None,8
$port.Open()
```

Solltemperatur dem Programm übermitteln (Punkt als Kommastelle verwenden):

```
$port.WriteLine("temp=40.55")
```

PID oder 2 Punkt Regelung verwenden:

```
$port.WriteLine("pid")
```

oder

```
$port.WriteLine("2p")
```

Aktuellen Messwert auslesen:

```
$port.DiscardInBuffer()
$port.ReadLine()
```

Messung beenden:

```
$port.WriteLine("ende")
```

Verbindung schließen:

```
$port.Close()
```