Fermentation temperature monitoring

using an MCU and a single-board computer

Uppsala universitet

group 1 – karl august forsman

supervisor – dr. ping wu, department of signals and systems

Abstract

A traditional brewing style, often referred to as *farmhouse brewing*, focuses on letting a mixed flora of yeasts and bacteria free-rise in temperature, hence not using any control systems. One historically popular option is letting it ferment in a cellar due to it's stable climate. Since the compounds produced by these microorganisms are in most cases unwanted, a lot of research has been put into detecting an excluding them from the process as part of quality control. Only a small share of the research have had the focus on producing beer using these microorganisms which makes it interesting to log the temperature dynamics of a mixed fermentation as a pre-study in order to learn how the fermentation can be controlled to get satisfactory results.

The objective of this project is develop a small and portable system prototype that allows an AVR based microcontroller unit to communicate with a Raspberry Pi through serial communication. The measurements are collected from a digital temperature sensor and put into a database. The Raspberry Pi based Arch Linux server is accessed on the local network by the end user and can be visualized using the Plotly Dash web application framework.In order to prevent oxidation it is possible to set an alarm level, for detecting temperature drops, that sends an alert if triggered. The result is an open source reliable data monitoring system coupled with a web application visualization.

Contents

Glossary

ADC Analog-digital conversion
ALARM Arch Linux ARM
API Application programming interface
ARM A processor architecture
AVR A family of microcontrollers
C The C programming language
CPU Central processing unit
DB Database
DS18B20 Temperature sensor
EEPROM Electrically erasable programmable read-only memory
GCC GNU compiler collection
JSON JavaScript object notation
Linux An open-source operating system
LSP Language server protocol
MCU Micro-controller Unit
OS Operating system
Python The Python programming language
PWM Pulse width modulation
RAM Random access memory
RPi Raspberry Pi
SBC Single-board computer
SSH Secure shell protocol
UART Universal asynchronous receiver/transmitter
USB Universal serial bus
WLAN Wireless local area network
Workstation ThinkPad P14s G2 AMD Ryzen 7 PRO 5850U

1 Introduction

1.1 Background

In recent years, the number of small scale breweries has increased in both the commercial and home-brew market. This has led to a demand of advanced brewing systems operating on much smaller volumes compared to the macro-scale brewing industry. These systems are often equipped with sensors logging temperature and pressure. For the most common commercial beers, such as lagers, there exists optimized means of production where the temperature controls plays a big part in order to maximize yield and minimize the length of the production cycle.

An uncontrolled form of brewing, called *farmhouse brewing*, focuses on letting a mixed flora of yeasts and bacteria (commonly Brettanomyces, Pediococcus, Lactobacillus) free-rise in temperature, hence not using any control systems [1]. The temperature influences the fermentation since the dynamics of the mixed fermentation culture varies depending on the ambient conditions. It is important to monitor several fermentations as a pre-study to learn about how the fermentation methods can be controlled in favour of the resulting beer.

1.2 Purpose of the project

The project aims to show that a reliable homebrew fermentation temperature logging system, with data visualization, can be created using open source software and easily accessible electronics. A list of used hardware and source code of the finished product will be published to GitHub. The objective of this project is to develop a small and portable system prototype that allows an AVR based MCU to communicate with an RPi SBC through serial communication.

The RPi based server is accessed on the internet by the end user and can be visualized using external libraries compatible with the choice of database.

1.3 Work distribution and planning

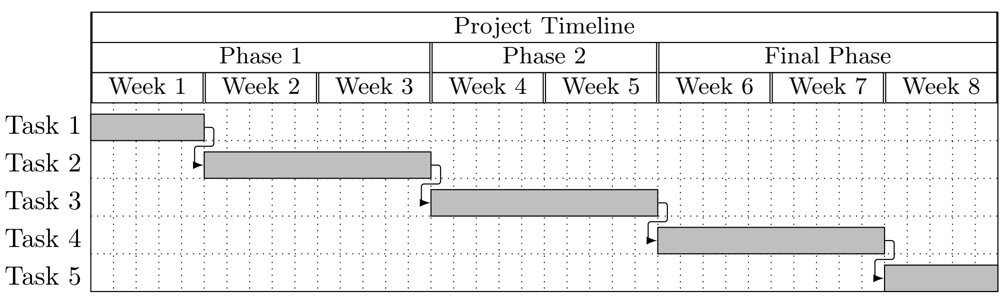

Figure 1.1: Gantt scheme showing the enumerated tasks during the project timespan

The work planning and execution is done by one person with the help of a supervisor, which is also head of the course. As seen in Fig, the duration of the project was split into three phases with the enumerated tasks

The time planning for the enumerated tasks is visualized as a Gantt scheme in Figure 1.1.

1. Set up project repository, as well as report and documentation workflow. Formulate preliminary time plan and goals. Research suitable hardware.
2. Get started with the AVR toolchain, write Makefiles and configure a development environment. Configure ARM compatible Linux on the RPi. Write UART routines and verify that the communication is working. Extend the communication to contain temperature data from the sensor.
3. Apply a filter to the signal. Design the website, configure databases and visualization of data.
4. Extend the website with more features. Collect or generate dummy-data in order to present a proof of concept.
5. Present the project, i.e all project phases, and finalize the report.

1.4 Grading criteria

The finally revised grading criteria is

- Grade 3. The temperature is measured by the microcontroller and presented through a visualization on a website that is accessible from the internet. The website is hosted on a Raspberry Pi computer configured with an appropriate ARM compatible Linux distribution.
- Grade 4. The temperature sensor signal can be processed through a filter. If a drastic temperature is detected, an automated script sends an alert through email.
- Grade 5. In addition to the stated grading criteria, the data is accessed in a manner similar to an interactive process view that a production company might order as a web application. The data collection is reliable and handles unexpected errors that might affect the system.

The goals and specifications of the project is set during the beginning of the first phase but can be reformulated according to the progress. Changes to the specifications and/or grading criteria is discussed and decided by the project supervisor.

2 Working principle

2.1 Microcontroller

Programming of the MCU is historically conducted in the processor architecture's specific assembly language. Today, many MCUs support programming in several high-level languages where Embedded-C is one of the most common. Other alternatives, such as the Raspberry Pi Zero uses MicroPython which is an embedded-compatible dialect of Python [2]. This allows for programming MCUs using the popular Python syntax wrapping around the C language. The Embedded-C language supports inline assembly code which is handy when the programmer wants section of code to be very specific and remain untouched by compiler optimizations.

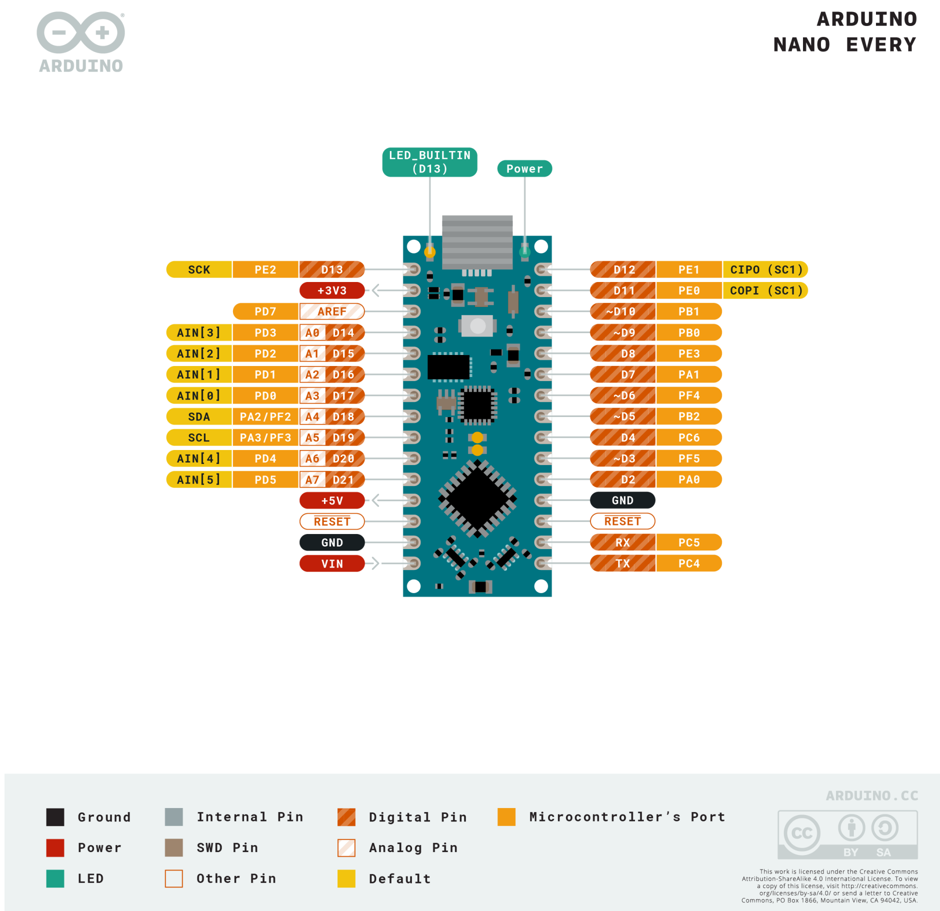

Figure 2.1: Arduino Nano board layout [3]

According to the Figure 2.1, the ATmega328p (on the Arduino Nano) has different properties for analog or digital signals depending on the pin number. The Arduino is more or less a plug-and-play device if the proper drivers are installed for the USB interface connected to the programmer.

2.2 Single-board computer

SBCs are a often cheap, lightweight and low-power alternative that contains many features of a PC on the system size of a debit card. One of the most notable manufacturers is Raspberry Pi whose latest models uses a multiprocessing 64-bit CPU, 1GB RAM and WLAN/Bluetooth networking capabilities. It is very useful for developing systems where a general purpose computers can fit inside a robot or scattering multiple units as a data collection network. There exists a multitude of lightweight open-source OS variants for the ARM architecture.

2.3 Digital temperature sensor

A digital temperature sensor is an embedded system that is able to convert an analog reading of some physical property that varies proportionally when exposed to different temperatures. As the ADC conversion is done inside the sensor, the emphasis is not on the exact physical mechanisms of the but focuses on communicating using drivers written in Embedded-C.

2.4 UART communication

An MCU can communicate with an SBC using the serial communication protocol UART [4]. Depending on the application, or hardware specifications, it is possible to customize the UART signal window to contain several combination of start, stop and parity bits.

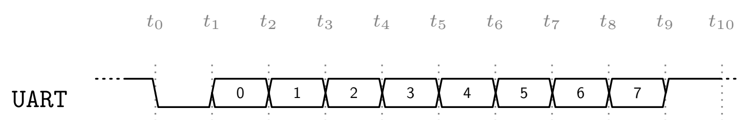

Figure 2.2: The bitwise UART communication pattern

As seen in 2.2, this example consists of a low start bit, eight data bits and a high stop bit.

2.5 Web application

A common way of constructing the front-end usually consists of

1. HTML – constructing the web page document structure. A very simple text-based website can be constructed only using HTML. The page is then interpreted depending on the user's system defaults.
2. CSS – each element in the HTML document can either be styled individually or globally by defining a style-sheet. This handles elementary attributes such as colours, fonts, spacing and padding.
3. Scripting – in order to perform work that handles computation of variables and automated tasks, it is often executed by some kind of script language. A common language for this purpose is JavaScript, or any of its super-sets such as TypeScript. This can also include calls to advanced APIs in order to for example get, process or present data in a certain way.

By using scripts, that can handle real-time information from the user and web browser, it is possible to create highly dynamic web pages that graphically scales well to both stationary and mobile devices.

3 Implementation

3.1 Overview of the system

(a) System overview

(b) Submerged sensor

Figure 3.1: The interconnected devices forms the monitoring system. As a safety precaution, the sensor is encapsulated in a food grade tube.

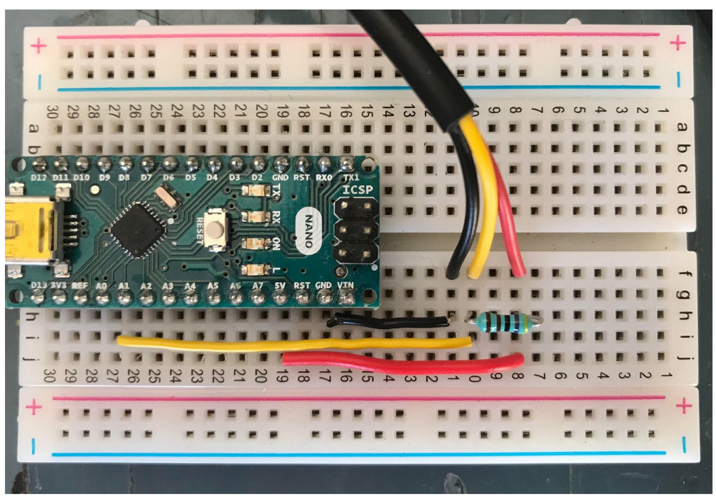

Figure 3.2: The Arduino and DS18B20 connected through a breadboard

3.2 Hardware and components

The software is written and compiled on a workstation grade Lenovo ThinkPad P14s Gen 2 with an AMD Ryzen 7 PRO 5850U.

The database and web application is hosted on a Raspberry Pi 3B which has a 64-bit ARM Cortex v8 processor running Arch Linux ARM [5]. During the development, the RPi is connected to a router and accessed locally through the SSH protocol from the workstation.

The MCU is an Arduino Nano board equipped with the 32-bit ATmega328p microprocessor. It has an on-board serial to USB interface which is connected through a USB cable. This connection provides the binary files when flashing, UART data transfer as well as supplying 5V power for driving the board.

A DS18B20 temperature sensor is connected to the MCU through a breadboard. The reference voltage is and data signal is connected through a 4.7kΩ pull-up resistor.

3.3 Software and development tools

3.3.1 Neovim

Neovim is completely terminal based and it is possible to clone the configuration file (`.init.vim`, often called a *dotfile*) to the RPi and remotely access an IDE-like text editor without forwarding any graphics through the SSH connection [6].

Since the AVR library uses a lot of macros, it is handy to get help from an LSP. Neovim has built in LSP support and while the programmer might not use the `clang` compiler for the actual compilation, it can be used with the `ccls` language server.

```
clang
-target
avr
-mmcu=atmega328p
-DF_CPU=16000000UL
-DBAUD=9600
-nostdinc
-ffreestanding
-isystem/usr/avr/include
-I/home/auan/Project/DS18B20_UART/inc
%h -x
%h c-header
```

Listing 3.1: The hidden `.ccls` file in the project root directory sets the compiler flags of interest

3.3.2 Cross-compilation

In order efficiently utilize the workstation, the Embedded-C code is cross-compiled using the AVR-GCC toolchain. Given the information of what MCU model the program is going to run on, known as the target, GCC is able to compile accordingly. As with non-embedded C programs, GCC is able to optimize the code by setting the appropriate compiler flags. Optimizing the code has several gains such as creating faster programs that require less space.

```
FILENAME     = main
HEADER1      = onewire
HEADER2      = uart
F_CPU        = 16000000UL
PORT         = /dev/ttyUSB0
```

```
6  DEVICE       = atmega328p
7  PROGRAMMER   = avrisp
8  BAUD         = 57600
9  CFLAGS         = -Wall -Os -ffreestanding
10 COMPILE      = avr-gcc $(CFLAGS) -mmcu=$(DEVICE) -DBAUD=$(BAUD) -DF_CPU
     =$(F_CPU) -I$(INCLUDE)
11 INCLUDE      = ./inc
12 OBJECTS      = obj/$(FILENAME).o obj/$(HEADER1).o obj/$(HEADER2).o
13
14
15 default: main.elf upload clean
16
17 $(FILENAME).elf: $(OBJECTS)
18     $(COMPILE) -o $(FILENAME).elf $(OBJECTS)
19     avr-objcopy -j .text -j .data -O ihex $(FILENAME).elf $(FILENAME).
     hex
20     avr-size --format=avr --mcu=$(DEVICE) $(FILENAME).elf
21
22 obj/$(FILENAME).o: src/$(FILENAME).c
23   $(COMPILE) -c src/$(FILENAME).c -o obj/$(FILENAME).o
24 obj/$(HEADER1).o: src/$(HEADER1).c inc/$(HEADER1).h
25   $(COMPILE) -c src/$(HEADER1).c -o obj/$(HEADER1).o
26 obj/$(HEADER2).o: src/$(HEADER2).c inc/$(HEADER2).h
27   $(COMPILE) -c src/$(HEADER2).c -o obj/$(HEADER2).o
28
29 upload:
30   avrdude -v -p $(DEVICE) -c $(PROGRAMMER) -P $(PORT) -b $(BAUD) -U
      flash:w:$(FILENAME).hex:i
31
32 .PHONY clean:
33   rm obj/*.o
34   rm $(FILENAME).elf
35   rm $(FILENAME).hex
```

Listing 3.2: GNU Makefile for compiling and flashing the program to the MCU

In Listing 3.2, only the binary `.elf` file from line 21 is flashed as a `.hex` file on line 32.

3.3.3 Org-mode

Org-mode is a text based software written in Emacs Lisp [7]. It is highly extensible and can be customized in order to create TODO lists, schedule meetings and write journal entries in order to document the project work.

3.3.4 AVR-GCC Toolchain

The code is compiled and uploaded to the MCU using a technique known as *cross compilation* which utilizes the speed of a workspace computer in order to compile and upload the binary `.hex` file. Compilation commands provided by the AVR-GCC toolchain [8] is called from a GNU Makefile which also handles hardware specific parameters such as CPU clock frequency, Baud rate and optimizer flags.

3.3.5 Git

Git, described by its man-page as *the stupid content tracker* is a versioning tool originally created by Linus Torvalds [9] when developing the Linux operating system. The whole project is stored locally within a git repository and hosted remotely by GitHub in order to synchronize the work between the Raspbery Pi and the workstation.

3.3.6 VimTeX

The report is written in LaTeXand compiled using the Neovim VimTeX plug-in [10].

3.4 Implementation

3.4.1 UART drivers

Calculating the correct register values, given the Baud rate, can be done directly from the file `setbaud.h`. To enable the communication, the external interrupts, `SEI()`, must be set in the C program. In Listing 3.3 below

```
UBRR0H = UBRRH_VALUE;
UBRR0L = UBRUBRR0L = UBRRL_VALUE;

UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0) | (1 << TXCIE0);
```

Listing 3.3: Enabling UART on the ATmega328p

the transfer and receive registers are also activated according to the ATmega328p datasheet [11] where

$$UBRR = \frac{f_{osc}}{16 \cdot BAUD} - 1. \tag{3.1}$$

Using a fixed clock frequency, it is possible to look up the rate of error given a set Baud rate.

Table 3.1: Baud and error rate

	$f_{osc} = 16000000$ Hz	
Baud	UBRRn	Error
9600	103	0.2%
14400	68	0.6%
57600	18	2.1%

In Table 3.1, the error rate for a Baud rate of 9600 produces is low and suffices since the small amounts of data is sent on an hourly basis.

3.4.2 DS18B20 drivers

The DS18B20 digital temperature sensor is manufactured by Maxim Integrated and the drivers used are based on Martin Thomas' modifications of Peter Dannegger's work (see credits in Appendix) on implementing the 1-wire library.

```
#define OW_MATCH_ROM     0x55
#define OW_SKIP_ROM      0xCC
#define OW_SEARCH_ROM    0xF0
#define OW_RSCRATCHPAD   0xBE
```

```
#define OW_CONVERTTEMP   0x44

#define OW_SEARCH_FIRST 0xFF        // start new search
#define OW_PRESENCE_ERR 0xFF
#define OW_DATA_ERR     0xFE
#define OW_LAST_DEVICE  0x00        // last device found
```

Listing 3.4: The hex code instructions for the DS18B20 sensor defined as macros in `onewire.h`

While Maxim Integrated does not provide or support driver source code, it is possible to write the drivers from reading guides on how to interface the hardware [12]. The 1-Wire protocol has, as the name suggests, one data wire for receiving instructions defined in Listing 3.4 and transferring the requested data. The schematic and flow of data is visualized in Figure 3.3. The data is written to the scratchpad and converted from a 12-bit two's complement binary resolution into a decimal floating point value with 0.0625 precision.

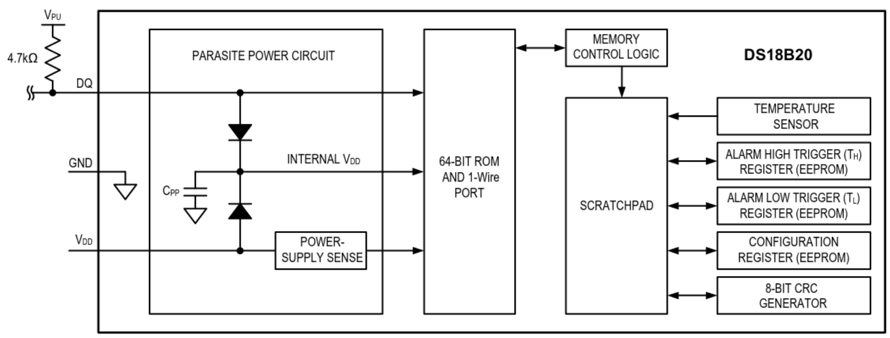

Figure 3.3: Schematic of the DS18B20 sensor [12]

3.4.3 MongoDB

MongoDB is an open source document database suitable for storing time-series data. This acts as the web application back-end and can also store configuration options set by the user. The database can be set up to run a daemonized process, which allows the user to get or set data whenever the server is powered on. Using the Python module PyMongo [13], it is possible to insert entries using JSON formatting.

3.4.4 Systemd service

Many Linux distributions use the Systemd init daemon as the first process booting and spawning other processes [14]. While many crucial programs runs in the background by Systemd as default, it is quite straightforward to add a another service that in this case runs the Python temperature reading script. Systemd has several options to restart the service on failure, start without the need to login and redirect `stdout` to the OS log called the *Linux Journal*. This simplifies tracking how often the service behaves unexpectedly during long timespans.

```
[Unit]
# Human readable unit name
Description=Reads serially from '/dev/ttyUSB*' and puts in MongoDB

[Service]
```

```
# Command that executes script
ExecStart=/usr/bin/python /home/alarm/Project/serial_temp_to_db.py
# Redirect print() to the Linux journal
Environment=PYTHONUNBUFFERED=1
# Able to notify that the service is ready
Type=notify
Restart=always

[Install]
# Start service at boot
WantedBy=default.target
```

Listing 3.5: The systemd service managing data collection

3.4.5 Plotly Dash web application

The web application consists of

1. A live graph showing the specified range of most recent data points. It is updated by a callback function listening to a timer.
2. A graph showing the historical data of measurements. The user can pick start and stop times from in the span of the registered dates in the database. The data is updated if the page is refreshed.
3. The user can set alarm levels which sends and alert email if a drastic temperature drop is detected.

Dash is an open source data visualization framework that runs in a web application. It is maintained by Plotly and build upon using their graphing software with the same name.

```
 import dash dependencies
 import modules

 external_stylesheets = ['stylesheet.css']

 app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

 # The application layout
 app.layout = html.Div(children =[
     html.H1('Large Heading'),
     html.H3('Smaller Heading'),
     html.Div([
         dcc.Graph(id='graph-id', animate=False),
         dcc.Interval(
             id='interval-component',
             interval=1*UPDATE_INTERVAL
             )
         ])
     )

# Callback function decoration
@app.callback(
    Output("update-figure", "figure"),
    Input("interval-component", "n_intervals"),
```

```
)

# Figure manipulation
def update_figure():
  return figure

if __name__ == '__main__':
    app.run_server(debug=True, host="0.0.0.0")
```

Listing 3.6: Pseudocode showing the structure of a Plotly Dash app

Each website object can be decorated with a callback function that either updates due to a time interval or through user interaction. In the example Listing 3.6, the graph is updated using a timer interval but can be changed to text fields or interactive clickable elements. The application is running as a server on port `8080` and the address `0.0.0.0` allows any unit on the local network to access the application web page by specifying the IP address followed by the correct port number. Since Dash is well documented for using Pandas, the data is loaded into a dataframe object which allows many ways of manipulation in areas such as statistics and signal processing. The data ordered by a time stamp, and is sorted in natural order, where the start and stop date can be chosen by the using a date picker.

3.4.6 Gmail API alert

One of the brewers worst enemies is oxidation. For homebrewers, it is very common to use an airlock that lets carbon dioxide escape when it reaches a certain over-pressure. A problem with this is caused by the exact opposite phenomena – when the beer and gas volume shrinks due to decreasing temperature which leads to oxygen entering through back pressure. An alert feature is implemented for storing fermentation vessels in an environment with fluctuating temperature and uses a parameter set by the user to the define how sensitive the alerting mechanism is to temperature drops. If a large enough drop is detected, the Systemd service uses the web based Gmail API to send an alert.

4 Result and discussion

The resulting product is an open-source temperature monitoring system suitable for beer fermentation. The user of the web application can filter through time-series data by filtering the start and end date using callback functions that communicates with the database. In order to set an alarm level for detecting a temperature drop, these callback functions are also used to write a non-volatile setting to the database. This is helping the brewer to avoid oxidation through the airlock caused by back pressure. Standard features provided by the Plotly library lets the user zoom and mouse-hover above the data points in order to receive immediate information about the temperature reading.

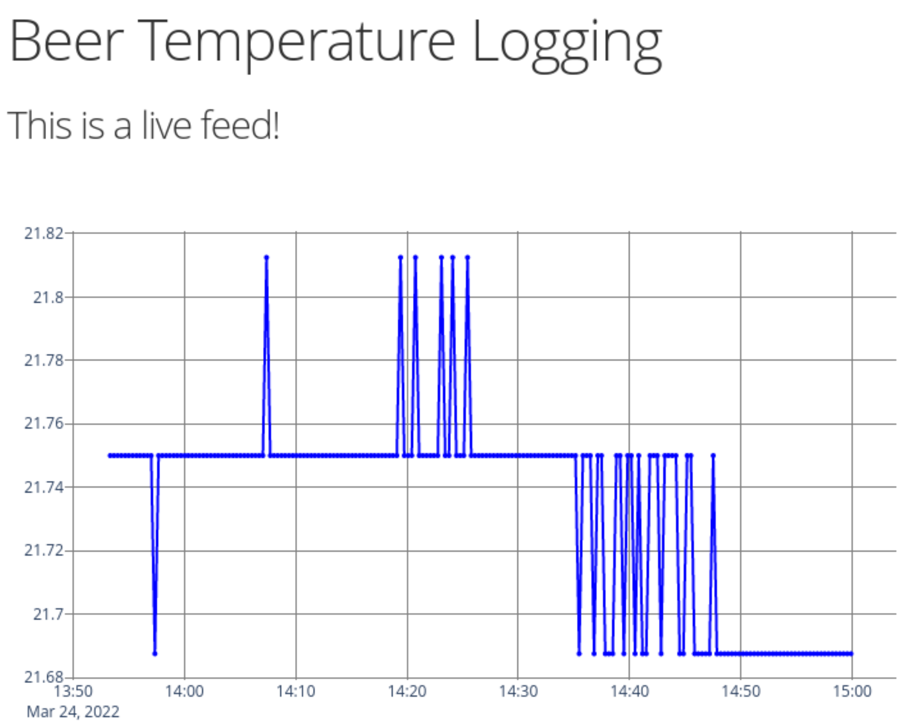

Figure 4.1: Live data graphed in Plotly Dash

The first part of the website grid is a live data feed that is shows the wanted amount of the latest data in Listing 4.1.

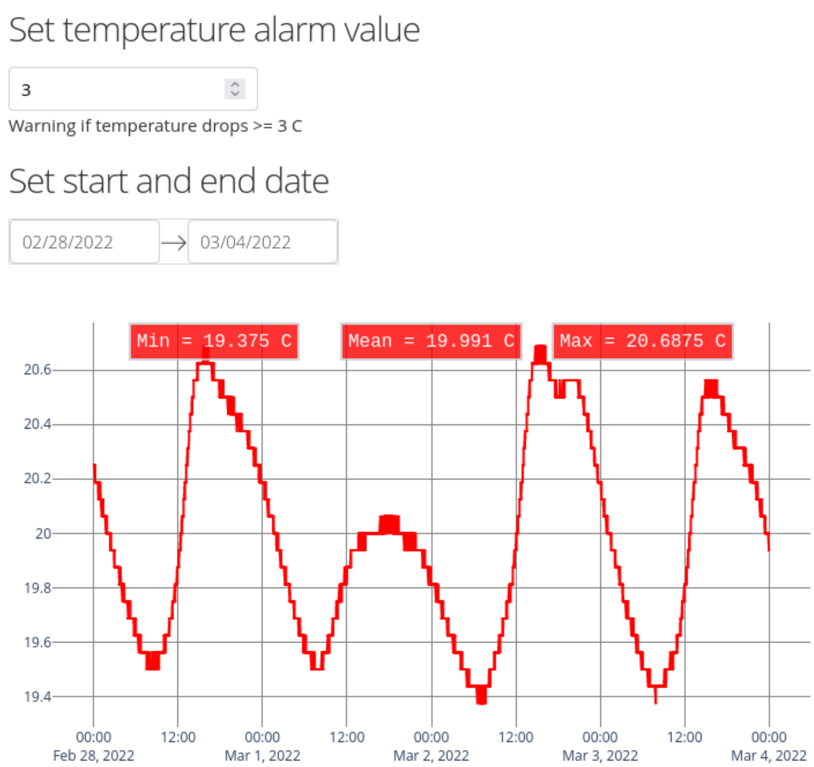

Figure 4.2: The time series temperature data graphed in Plotly Dash

In Figure 4.1, the date picker above the graph gets the first and last dates of the time series. It handles input exceptions and only accept values in that specified range. The temperature drop alert puts the value in the database and is ready to send the alert if the next temperature reading triggers the alarm.

Viewing the unfiltered data points gives an overall view of the temperature fluctuations during longer time spans. When viewing in smaller time windows, the signal is in need of filtering by the MCU to reduce noise and provide a smoother curve.

```
Mar 16 06:27:42 alarm systemd[359]: Starting Reads serially from /dev/ttyUSB0 and puts in MongoDB...
Mar 16 06:27:49 alarm python[18967]: Running tests on port: /dev/ttyUSB0
Mar 16 06:27:49 alarm python[18967]: Baud rate = 9600
Mar 16 06:27:50 alarm python[18967]: Correct reading: 19.9375
Mar 16 06:27:50 alarm python[18967]: Correct reading: 19.9375
Mar 16 06:27:51 alarm python[18967]: Correct reading: 19.4375
Mar 16 06:27:51 alarm python[18967]: Passed init tests on 3 correct readings and 0 faulty readings
Mar 16 06:28:11 alarm python[18967]: Current temp drop parameter: 5
Mar 16 06:29:31 alarm python[18967]: Warning message sent
Mar 16 06:29:31 alarm python[18967]: 17f916f36c421171
```

Listing 4.1: Logging entries in the Linux Journal as a result of the running system

The system is quite easy to debug since all of the print statements are redirected to the Linux Journal. An example of the running system starting and sending an alarm is seen in Listing 4.1.

5 Conclusions and futher work

The system provides a user-friendly front end where the collected temperature data of the beer can be visualized, accessed and monitored through custom settings. If the temperature drops and triggers the alarm, an email is sent containing information about the difference of the two latest temperature readings. The choice of the main two programming languages, Python and C, proved to be sufficient with a few additions of basic Linux scripting. The dependencies for the C programs was easy to track. Python, on the other hand, relies on many module dependencies which makes it hard to track. This is a trade-off when wrapping many functionalities consistently within one programming or scripting language.

5.1 Improvements

While there exist a multitude of possible extensions to the system, many of the existing features can be improved or optimized.

1. Pre-processing the data through a filter on the MCU is more efficient compared to post-processing larger parts on the dataset while also running the web application. The post-processing was made using the `savgol` filter from the Numpy Python library and could have been improved by tuning an optimal resampling size.
2. If the data is filtered by the RPi, the trace is toggled by the user and only computed when chosen. When filtering a large time-series, the user should be able to set the use choose a dynamic window size.
3. Since the MCU only polls for an input every hour it could be possible to implement a rule to let the Linux kernel power the USB connecting to the MCU shortly before asking for a measurement.
4. The server can be deployed to be accessed through the internet and lets the user remotely monitor the temperature in real time. This was not investigated due to security issues.
5. Presenting the statistics of the data in a separate area.

References

[1] "Mixed Fermentation," Milk The Funk Wiki. (), [Online]. Available: http://www.milkthefunk.com/wiki/Mixed_Fermentation (visited on Mar. 24, 2022).

[2] "MicroPython," GitHub. (), [Online]. Available: https://github.com/micropython (visited on Mar. 24, 2022).

[3] "Pinout-NANOevery_latest.png (PNG Image, 2571 × 2572 pixels) — Scaled (36%)." (), [Online]. Available: https://content.arduino.cc/assets/Pinout-NANOevery_latest.png (visited on Mar. 22, 2022).

[4] A. Trevennor, *Practical AVR Microcontrollers: Games, Gadgets, and Home Automation with the Microcontroller Used in the Arduino*. Apress, Nov. 27, 2012, 400 pp., ISBN: 978-1-4302-4447-9.

[5] "Raspberry Pi 3 | Arch Linux ARM." (), [Online]. Available: https://archlinuxarm.org/platforms/armv8/broadcom/raspberry-pi-3 (visited on Mar. 22, 2022).

[6] "Documentation - Neovim." (), [Online]. Available: https://neovim.io/doc/general/ (visited on Mar. 22, 2022).

[7] "Org Mode." (), [Online]. Available: https://orgmode.org (visited on Mar. 24, 2022).

[8] "GCC Compilers for AVR® and Arm®-Based MCUs and MPUs | Microchip Technology." (), [Online]. Available: https://www.microchip.com/en-us/tools-resources/develop/microchip-studio/gcc-compilers (visited on Mar. 22, 2022).

[9] *Git - fast, scalable, distributed revision control system*, Git, Mar. 22, 2022. [Online]. Available: https://github.com/git/git (visited on Mar. 22, 2022).

[10] K. Y. Lervåg, *VimTeX*, Mar. 22, 2022. [Online]. Available: https://github.com/lervag/vimtex (visited on Mar. 22, 2022).

[11] Atmel Corporation, *ATmega328p datasheet*. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P%3Csub%3ED%3C/sub%3Eatasheet.pdf (visited on Feb. 6, 2022).

[12] "DS18B20 - Programmable Resolution 1-Wire Digital Thermometer," p. 20,

[13] "PyMongo 4.0.2 Documentation — PyMongo 4.0.2 documentation." (), [Online]. Available: https://pymongo.readthedocs.io/en/stable/index.html (visited on Mar. 22, 2022).

[14] "Systemd - ArchWiki." (), [Online]. Available: https://wiki.archlinux.org/title/Systemd (visited on Mar. 24, 2022).

Appendix

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "onewire.h"
#include "uart.h"

#define MSG_LEN_Y 17
#define MSG_LEN_N 14

void main(void)
{
  sei();
  uart_init();
  ow_reset();
  uint8_t data;
  uint16_t len = 8;
  char buffer[len];

  for(;;)
  {
    /* Polling for 'R' */
    if (uart_read_count() > 0)
    {
      data = uart_read();
      if (data == 'R') {
        ow_reset();
        ow_temp_rd(buffer);
        uart_send_arr(buffer, len);
        uart_send_byte('\r');
        uart_send_byte('\n');
      }
    }
  }
}
```

Listing 5.1: ~/Project/DS18B20_UART/src/main.c

```c
#include "uart.h"

/*Global volatile variables for this file*/
volatile static uint8_t uart_tx_active = 1;
volatile static uint8_t rx_buf[RX_BUF_SZ] = {0};
volatile static uint16_t rx_count = 0;

/* Recv interrupt */
ISR(USART_RX_vect)
{
  volatile static uint16_t rx_write_pos = 0;

  rx_buf[rx_write_pos] = UDR0;
```

```
  rx_count++;
  if (rx_write_pos >= RX_BUF_SZ)
  {
    rx_write_pos = 0;
  }

}
/* Transf interrupt */
ISR(USART_TX_vect)
{
  uart_tx_active = 1;

}

void uart_init(void)
{
#if SPEED2X
    UCSR0A |= 1 << U2X0;
#else
    UCSR0A &= ~(1 << U2X0);
#endif
 /* Baud rate helpers (set baud.h) */
  UBRR0H = UBRRH_VALUE;
  UBRR0L = UBRRL_VALUE;

  UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0) | (1 << TXCIE0)
    ;

}

void uart_send_byte(uint8_t c)
{
  while (!uart_tx_active);
  uart_tx_active = 0;
  UDR0 = c;
}

void uart_send_arr(char *c, uint16_t len)
{
  for (uint16_t i = 0; i < len; i++)
  {
    uart_send_byte(c[i]);
  }
}

void uart_send_str(uint8_t *c)
{
  uint16_t i = 0;
  do
  {
    uart_send_byte(c[i]);
    i++;
```

```c
  } while (c[i] != '\0');
}

uint16_t uart_read_count(void)
{
  return rx_count;
}

uint8_t uart_read(void)
{
  static uint16_t rx_read_pos = 0;
  uint8_t data = 0;
  data = rx_buf[rx_read_pos];
  rx_read_pos++;
  rx_count--;
  if (rx_read_pos)
  {
    rx_read_pos = 0;
  }
  return data;
}
```

Listing 5.2: ~/Project/DS18B20_UART/src/uart.c

```c
/*
 2/2022 - Modified for use in a temperature measuring project
 conducted by August Forsman (auan(at)mailbox.org).

For the full library, see the contributions made by

Access Dallas 1-Wire Devices with ATMEL AVRs
Author of the initial code: Peter Dannegger (danni(at)specs.de)
modified by Martin Thomas (mthomas(at)rhrk.uni-kl.de)
 9/2004 - use of delay.h, optional bus configuration at runtime
10/2009 - additional delay in ow_bit_io for recovery
 5/2010 - timing modifcations, additonal config-values and comments,
          use of atomic.h macros, internal pull-up support
 7/2010 - added method to skip recovery time after last bit transfered
          via ow_command_skip_last_recovery
*/


#include <avr/io.h>
#include <util/delay.h>
#include <util/atomic.h>
#include <stdio.h>

#include "onewire.h"
#include "uart.h"

#define MAX_FLOAT 127.9375

```

```
#ifdef OW_ONE_BUS

#define OW_GET_IN()   ( OW_IN & (1<<OW_PIN))
#define OW_OUT_LOW()  ( OW_OUT &= (~(1 << OW_PIN)) )
#define OW_OUT_HIGH() ( OW_OUT |= (1 << OW_PIN) )
#define OW_DIR_IN()   ( OW_DDR &= (~(1 << OW_PIN )) )
#define OW_DIR_OUT()  ( OW_DDR |= (1 << OW_PIN) )

#else

/* set bus-config with ow_set_bus() */
uint8_t OW_PIN_MASK;
volatile uint8_t* OW_IN;
volatile uint8_t* OW_OUT;
volatile uint8_t* OW_DDR;

#define OW_GET_IN()   ( *OW_IN & OW_PIN_MASK )
#define OW_OUT_LOW()  ( *OW_OUT &= (uint8_t) ~OW_PIN_MASK )
#define OW_OUT_HIGH() ( *OW_OUT |= (uint8_t)  OW_PIN_MASK )
#define OW_DIR_IN()   ( *OW_DDR &= (uint8_t) ~OW_PIN_MASK )
#define OW_DIR_OUT()  ( *OW_DDR |= (uint8_t)  OW_PIN_MASK )

void ow_set_bus(volatile uint8_t* in,
  volatile uint8_t* out,
  volatile uint8_t* ddr,
  uint8_t pin)
{
  OW_DDR=ddr;
  OW_OUT=out;
  OW_IN=in;
  OW_PIN_MASK = (1 << pin);
  ow_reset();
}

#endif

uint8_t ow_input_pin_state()
{
  return OW_GET_IN();
}

void ow_parasite_enable(void)
{
  OW_OUT_HIGH();
  OW_DIR_OUT();
}

void ow_parasite_disable(void)
{
  OW_DIR_IN();
#if (!OW_USE_INTERNAL_PULLUP)
  OW_OUT_LOW();
```

```
#endif
}


uint8_t ow_reset(void)
{
  uint8_t err;

  OW_OUT_LOW();
  OW_DIR_OUT();            // pull OW-Pin low for 480us
  _delay_us(480);

  ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
    // set Pin as input - wait for clients to pull low
    OW_DIR_IN(); // input
#if OW_USE_INTERNAL_PULLUP
    OW_OUT_HIGH();
#endif

    _delay_us(64);       // was 66
    err = OW_GET_IN();   // no presence detect
                         // if err!=0: nobody pulled to low, still high
  }

  // after a delay the clients should release the line
  // and input-pin gets back to high by pull-up-resistor
  _delay_us(480 - 64);       // was 480-66
  if( OW_GET_IN() == 0 ) {
    err = 1;             // short circuit, expected low but got high
  }

  // TODO check if this works
  /*return (OW_GET_IN() == 0) ? err = 1:0;*/
  return err;
}


/* Timing issue when using runtime-bus-selection (!OW_ONE_BUS):
   The master should sample at the end of the 15-slot after initiating
   the read-time-slot. The variable bus-settings need more
   cycles than the constant ones so the delays had to be shortened
   to achieve a 15uS overall delay
   Setting/clearing a bit in I/O Register needs 1 cycle in OW_ONE_BUS
   but around 14 cycles in configurable bus (us-Delay is 4 cycles per uS) */
static uint8_t ow_bit_io_intern( uint8_t b, uint8_t with_parasite_enable )
{
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
#if OW_USE_INTERNAL_PULLUP
    OW_OUT_LOW();
```

```
#endif
    OW_DIR_OUT();     // drive bus low
    _delay_us(2);     // T_INT > 1usec accoding to timing-diagramm
    if ( b ) {
      OW_DIR_IN(); // to write "1" release bus, resistor pulls high
#if OW_USE_INTERNAL_PULLUP
      OW_OUT_HIGH();
#endif
    }

    // "Output data from the DS18B20 is valid for 15usec after the falling
    // edge that initiated the read time slot. Therefore, the master must
    // release the bus and then sample the bus state within 15ussec from
    // the start of the slot."
    _delay_us(15-2-OW_CONF_DELAYOFFSET);

    if( OW_GET_IN() == 0 ) {
      b = 0;  // sample at end of read-timeslot
    }

    _delay_us(60-15-2+OW_CONF_DELAYOFFSET);
#if OW_USE_INTERNAL_PULLUP
    OW_OUT_HIGH();
#endif
    OW_DIR_IN();

    if ( with_parasite_enable ) {
      ow_parasite_enable();
    }

  } /* ATOMIC_BLOCK */

  _delay_us(OW_RECOVERY_TIME); // may be increased for longer wires

  return b;
}

uint8_t ow_bit_io( uint8_t b )
{
  return ow_bit_io_intern( b & 1, 0 );
}

uint8_t ow_byte_wr( uint8_t b )
{
  uint8_t i = 8, j;

  do {
    j = ow_bit_io( b & 1 );
    b >>= 1;
```

```
    if( j ) {
      b |= 0x80;
    }
  } while( --i );

  return b;
}

uint8_t ow_byte_wr_with_parasite_enable( uint8_t b )
{
  uint8_t i = 8, j;

  do {
    if ( i != 1 ) {
      j = ow_bit_io_intern( b & 1, 0 );
    } else {
      j = ow_bit_io_intern( b & 1, 1 );
    }
    b >>= 1;
    if( j ) {
      b |= 0x80;
    }
  } while( --i );

  return b;
}


uint8_t ow_byte_rd( void )
{
  // read by sending only "1"s, so bus gets released
  // after the init low-pulse in every slot
  return ow_byte_wr( 0xFF );
}


uint8_t ow_rom_search( uint8_t diff, uint8_t *id )
{
  uint8_t i, j, next_diff;
  uint8_t b;

  if( ow_reset() ) {
    return OW_PRESENCE_ERR;         // error, no device found <---
   early exit!
  }

  ow_byte_wr( OW_SEARCH_ROM );         // ROM search command
  next_diff = OW_LAST_DEVICE;          // unchanged on last device

  i = OW_ROMCODE_SIZE * 8;             // 8 bytes

  do {
```

```
    j = 8;                          // 8 bits
    do {
      b = ow_bit_io( 1 );         // read bit
      if( ow_bit_io( 1 ) ) {      // read complement bit
        if( b ) {               // 0b11
          return OW_DATA_ERR; // data error <--- early exit!
        }
      }
      else {
        if( !b ) {              // 0b00 = 2 devices
          if( diff > i || ((*id & 1) && diff != i) ) {
            b = 1;          // now 1
            next_diff = i;  // next pass 0
          }
        }
      }
      ow_bit_io( b );             // write bit
      *id >>= 1;
      if( b ) {
        *id |= 0x80;            // store bit
      }

      i--;

    } while( --j );

    id++;                           // next byte

  } while( i );

  return next_diff;                   // to continue search
}


static void ow_command_intern( uint8_t command, uint8_t *id, uint8_t
    with_parasite_enable )
{
  uint8_t i;

  ow_reset();

  if( id ) {
    ow_byte_wr( OW_MATCH_ROM );     // to a single device
    i = OW_ROMCODE_SIZE;
    do {
      ow_byte_wr( *id );
      id++;
    } while( --i );
  }
  else {
    ow_byte_wr( OW_SKIP_ROM );      // to all devices
  }
```

```
  if ( with_parasite_enable  ) {
    ow_byte_wr_with_parasite_enable( command );
  } else {
    ow_byte_wr( command );
  }
}

void ow_command( uint8_t command, uint8_t *id )
{
  ow_command_intern( command, id, 0);
}

void ow_command_with_parasite_enable( uint8_t command, uint8_t *id )
{
  ow_command_intern( command, id, 1 );
}

void ow_temp_rd(char *buffer)
{
  // 12bytes long
  uint8_t temperature[2];
  int8_t digit;
  uint16_t decimal;

  // Reset, skip ROM, start conversion
  ow_reset();
  ow_byte_wr(OW_SKIP_ROM);
  ow_byte_wr(OW_CONVERTTEMP);

  ow_reset();
  ow_byte_wr(OW_SKIP_ROM);
  ow_byte_wr(OW_RSCRATCHPAD);

  // Read scratchpad
  temperature[0] = ow_byte_rd();
  temperature[1] = ow_byte_rd();
  ow_reset();


  // Store temperature int and dec digits
  digit = temperature[0] >> 4;
  digit |= (temperature[1] & 0b00000111) << 4;
  if (temperature[1] > 0b01111111) {
    digit = digit - MAX_FLOAT;
  }

  // Store dec digits
  decimal = temperature[0] & 0xf;
  decimal *= OW_DEC_STEP_12BIT;

  sprintf(buffer, "%d.%04u", digit, decimal);
```

```
}
```

Listing 5.3: ~/Project/DS18B20_UART/src/onewire.c

```
#ifndef UART_H_
#define UART_H_
/* TODO get rid of this redefenition */
#define BAUD 9600
#include <avr/io.h>
#include <util/delay.h>
#include <util/setbaud.h>
#include <avr/interrupt.h>

#define RX_BUF_SZ 12

void uart_init(void);
void uart_send_byte(uint8_t c);
void uart_send_str(uint8_t *c);
void uart_send_arr(char *c, uint16_t len);

uint16_t uart_read_count(void);
uint16_t uart_read_count(void);
uint8_t uart_read(void);

#endif /* ifndef UART_H_ */
```

Listing 5.4: ~/Project/DS18B20_UART/inc/uart.h

```
#ifndef ONEWIRE_H_
#define ONEWIRE_H_

#ifdef __cplusplus
extern "C" {
#endif

#include <stdint.h>

/*******************************************/
/* Hardware connection                     */
/*******************************************/

/* Define OW_ONE_BUS if only one 1-Wire-Bus is used
   in the application -> shorter code.
   If not defined make sure to call ow_set_bus() before using
   a bus. Runtime bus-select increases code size by around 300
   bytes so use OW_ONE_BUS if possible */
#define OW_ONE_BUS

#ifdef OW_ONE_BUS

#define OW_PIN  0
#define OW_IN   PINC
```

```
#define OW_OUT  PORTC
#define OW_DDR  DDRC
#define OW_CONF_DELAYOFFSET 0

#else
#if ( F_CPU < 1843200 )
#warning | Experimental multi-bus-mode is not tested for
#warning | frequencies below 1,84MHz. Use OW_ONE_WIRE or
#warning | faster clock-source (i.e. internal 2MHz R/C-Osc.).
#endif
#define OW_CONF_CYCLESPERACCESS 13
#define OW_CONF_DELAYOFFSET ( (uint16_t)( ((OW_CONF_CYCLESPERACCESS) * 1000000L) / F_CPU ) )
#endif

// Recovery time (T_Rec) minimum 1usec - increase for long lines
// 5 usecs is a value give in some Maxim AppNotes
// 30u secs seem to be reliable for longer lines
//#define OW_RECOVERY_TIME        5  /* usec */
//#define OW_RECOVERY_TIME      300 /* usec */
#define OW_RECOVERY_TIME         10 /* usec */

// Use AVR's internal pull-up resistor instead of external 4,7k resistor.
// Based on information from Sascha Schade. Experimental but worked in tests
// with one DS18B20 and one DS18S20 on a rather short bus (60cm), where both
// sensores have been parasite-powered.
#define OW_USE_INTERNAL_PULLUP     0  /* 0=external, 1=internal */

/*******************************************/


#define OW_MATCH_ROM    0x55
#define OW_SKIP_ROM     0xCC
#define OW_SEARCH_ROM   0xF0
#define OW_RSCRATCHPAD  0xBE
#define OW_CONVERTTEMP  0x44

#define OW_SEARCH_FIRST 0xFF        // start new search
#define OW_PRESENCE_ERR 0xFF
#define OW_DATA_ERR     0xFE
#define OW_LAST_DEVICE  0x00        // last device found

#define OW_DEC_STEP_12BIT 625

// rom-code size including CRC
#define OW_ROMCODE_SIZE 8

extern uint8_t ow_reset(void);

```

```c
extern uint8_t ow_bit_io( uint8_t b );
extern uint8_t ow_byte_wr( uint8_t b );
extern uint8_t ow_byte_rd( void );
extern void ow_temp_rd(char *buffer);

extern uint8_t ow_rom_search( uint8_t diff, uint8_t *id );

extern void ow_command( uint8_t command, uint8_t *id );
extern void ow_command_with_parasite_enable( uint8_t command, uint8_t
    *id );

extern void ow_parasite_enable( void );
extern void ow_parasite_disable( void );
extern uint8_t ow_input_pin_state( void );

#ifndef OW_ONE_BUS
extern void ow_set_bus( volatile uint8_t* in,
  volatile uint8_t* out,
  volatile uint8_t* ddr,
  uint8_t pin );
#endif

#ifdef __cplusplus
}
#endif

#endif
```

Listing 5.5: ~/Project/DS18B20_UART/inc/onewire.h

```python
#!/bin/python
import dash
from dash import dcc
from dash import html
from dash.dependencies import Input, Output, State
import serial
from pymongo import MongoClient
from datetime import datetime as dt
from datetime import date
import numpy as np
import time
import pprint
import pandas as pd
import plotly
from random import random
import plotly.graph_objs as go
import re
from scipy.signal import savgol_filter

''' **************************************** '''

# TODO add constants
```

```python
BAUD = 9600
IP_ADR = 27017
HOUR_MS = 60*600000
PADDING = 150
BG_COLOR = '#FFFFFF'

''' MongoDB '''
client = MongoClient('localhost', IP_ADR)
db = client.beertemp
settings = db.settings
entry = db.entries

''' ****************************************  '''

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div(style={'padding': PADDING, 'background' :
    BG_COLOR}, children =[
     html.H1('Beer Temperature Logging'),
     html.H3('This is a live feed!'),
     html.Div([
         dcc.Graph(id='live-update-graph-scatter', animate=False),
         html.Hr(),
         dcc.Interval(
             id='interval-component',
             interval=1*HOUR_MS
             )
         ]),
     html.Div([
         html.H3('Set temperature alarm value'),
         # dcc.Input(
             # id="input_range_min", type="number", debounce=True,
    placeholder="Set min temp",
             # min=-5, max=40, step=1, value=int(settings.find_one({"
    _id": "settings"})['min'])
             # ),
         # dcc.Input(
             # id="input_range_max", type="number", debounce=True,
    placeholder="Set max temp",
             # min=-5, max=40, step=1, value=int(settings.find_one({"
    _id": "settings"})['max'])
             # ),
         dcc.Input(
             id="input_range_drop", type="number", debounce=True,
    placeholder="Temp drop tolerance",
             min=1, max=40, step=1, value=int(settings.find_one({"_id":
     "settings"})['drop'])
             ),
         # html.Div(id="min-max-out"),
         html.Div(id="drop-out"),
```

```
        html.H3('Set start and end date'),
        dcc.DatePickerRange(
            id='date-picker-range',
            min_date_allowed = entry.find_one()['time'].date(),
            max_date_allowed = list(entry.find().sort('$natural',-1).
    limit(1))[0]['time'].date(),

            # TODO timedelta based fallback
            start_date = date(2022,2,28),
            end_date = date(2022,3,4)
            ),
        dcc.Graph(id='history-graph-scatter', animate=False),
        html.Div(id='output-container-date-picker-range')
        ]),
    ])


@app.callback(Output('live-update-graph-scatter', 'figure'),
        Input('interval-component', 'n_intervals'))
def update_graph_scatter(graph_update):
    LIVE_RES = 200

    try:
        df = pd.DataFrame(list(db.entries.find().limit(int(LIVE_RES)).
    sort([('$natural',-1)])))
        trace = go.Scatter(
            x=df['time'],
            y=df['temperature'],
            name='Beer temperature',
            mode= 'lines+markers',
            marker = {'color': 'Blue',
                'size': 4}
            )
    except Exception as e:
        print(str(e))

    layout = go.Layout(
            paper_bgcolor='rgba(0,0,0,0)',
            plot_bgcolor='rgba(0,0,0,0)',
            margin=dict(l=20, r=20, t=50, b=50),
            yaxis= {'autorange': True}
            )

    fig = go.Figure(data=[trace], layout=layout)

    fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='Grey')
    fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='Grey')

    return fig

# TODO callback to apply Savgol filtering
@app.callback(Output('history-graph-scatter', 'figure'),
```

```
        [Input(component_id='date-picker-range', component_property='
    start_date'),
        Input(component_id='date-picker-range',component_property='
    end_date')]
        )
# @app.callback(Output('live-update-graph-scatter', 'figure'))
def update_history(start_date, end_date):
    start_date = pd.to_datetime(start_date)
    end_date = pd.to_datetime(end_date)
    df = pd.DataFrame(list(db.entries.find().sort([('_id',-1)])))

    filtered_df = df[df['time'].between(
        dt.strftime(start_date, "%Y-%m-%d"),
        dt.strftime(end_date, "%Y-%m-%d")
    )]

    SAVGOL_WIN_LEN = 201

    try:
        # trace1 = go.Scatter(
            # x=df['time'],
            # y=df['temperature'],
            # name='Beer temperature',
            # mode= 'markers',
            # marker = {'color': 'Blue',
                # 'size': 3}
            # )

        trace2 = go.Scatter(
            x=filtered_df['time'],
            # y=savgol_filter(filtered_df['temperature'],
    SAVGOL_WIN_LEN, 2),
            y=filtered_df['temperature'],
            name='Sensor signal [C]',
            # name='Filtered signal',
            mode= 'lines+markers',
            marker = {'color': 'Red',
                'size': 2}
            )
    except Exception as e:
        print(str(e))

    layout = go.Layout(
            paper_bgcolor='rgba(0,0,0,0)',
            plot_bgcolor='rgba(0,0,0,0)',
            margin=dict(l=20, r=20, t=50, b=50),
            yaxis= {'autorange': True}
            )

    # fig = go.Figure(data=[trace1, trace2], layout=layout)
    fig = go.Figure(data=trace2, layout=layout)
    min_temp = filtered_df['temperature'].min()
```

```python
    max_temp = filtered_df['temperature'].max()
    mean_temp = filtered_df['temperature'].mean()

    fig.add_annotation(text="Min = " + str(min_temp) + " C",
        xref="paper", yref="paper",
        x=0.1, y=1, showarrow=False,
           font=dict(
            family="Courier New, monospace",
            size=16,
            color="#ffffff"
            ),
        bordercolor="#c7c7c7",
        borderwidth=2,
        borderpad=4,
        bgcolor="red",
        opacity=0.8
        )

    fig.add_annotation(text="Max = " + str(max_temp) + " C",
        xref="paper", yref="paper",
        x=0.9, y=1, showarrow=False,
           font=dict(
            family="Courier New, monospace",
            size=16,
            color="#ffffff"
            ),
        bordercolor="#c7c7c7",
        borderwidth=2,
        borderpad=4,
        bgcolor="red",
        opacity=0.8
        )

    fig.add_annotation(text="Mean = " + str(round(mean_temp, 3)) + " C",
        xref="paper", yref="paper",
        x=0.5, y=1, showarrow=False,
           font=dict(
            family="Courier New, monospace",
            size=16,
            color="#ffffff"
            ),
        bordercolor="#c7c7c7",
        borderwidth=2,
        borderpad=4,
        bgcolor="red",
        opacity=0.8
        )
    fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='Grey')
    fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='Grey')

    return fig
```

```python

# @app.callback(
    # Output("min-max-out", "children"),
    # Input("input_range_min", "value"),
    # Input("input_range_max", "value"),
# )
# def set_temp_min_max(rangemin, rangemax):
    # # if rangemin or rangemax == None:
        # # return "Temperature(s) out of range"
    # if rangemin >= rangemax:
        # return "Min has to be lower than max"
    # else:
        # settings.update_one({"_id": "settings"}, {"$set":{"min": rangemin, "max": rangemax}})
        # return "Max temp: {}C | Min temp: {}C".format(rangemin, rangemax)

@app.callback(
    Output("drop-out", "children"),
    Input("input_range_drop", "value"),
)
def set_temp_drop(rangedrop):
    if rangedrop == None:
        return "Temp drop out of range"
    else:
        settings.update_one({"_id": "settings"}, {"$set":{"drop": rangedrop}})
        return "Warning if temperature drops >= {} C".format(rangedrop)

if __name__ == '__main__':
    app.run_server(debug=True, host="0.0.0.0")
```

Listing 5.6: ~/Project/livedb.py

```python
import serial
from pymongo import MongoClient
from datetime import datetime as dt
import time
import systemd.daemon
import os.path
import base64
from email.mime.text import MIMEText
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError

# If modifying these scopes, delete the file token.json.
SCOPES = ['https://www.googleapis.com/auth/gmail.send']

```

```
# Iniatilzation temperature. 0C looks better than a 'trash output'.
temp = 0

''' Constants '''
MAX_FERM_TEMP = 50
BAUD = 9600
PORT = "/dev/ttyUSB0"
READ_CMD = b'R'
MIN_FLOAT_LEN = 9
MAX_FLOAT_LEN = 11
IP_ADR = 27017
CORRECT_TESTS = 3

''' MongoDB '''
client = MongoClient('localhost', IP_ADR)
db = client.beertemp
entry = db.entries
settings = db.settings

# Open serial connection to MCU
ser = serial.Serial(PORT, BAUD)

def get_temperature(temp):
    ser.write(bytes(READ_CMD))
    line = ser.readline().decode()
    if len(line) > MIN_FLOAT_LEN \
            and len(line)< MAX_FLOAT_LEN \
            and float(line.strip('\x00\n\r')) < MAX_FERM_TEMP:
        temp = (float(line.strip('\x00\n\r')))
    return temp

def temp_to_db(temp):
    temp_entry = {"temperature": get_temperature(temp),
            "time": dt.utcnow()}
    entry.insert_one(temp_entry)

def init_script(temp):
    print('Running tests on port: ' + ser.portstr)
    print('Baud rate = ' + str(BAUD))
    correct_count = 0
    faulty_count = 0
    while correct_count != CORRECT_TESTS:
        try:
            time.sleep(0.5)
            ser.write(READ_CMD)
            line = ser.readline().decode()
            if len(line) > MIN_FLOAT_LEN \
                    and len(line)< MAX_FLOAT_LEN \
                    and float(line.strip('\x00\n\r')) < MAX_FERM_TEMP:
                print('Correct reading: ' + line.strip('\x00\n\r'))
                correct_count+=1
            else:
```

```
                print('Faulty reading: ' + line.strip('\x00\n\r'))
                faulty_count+=1
        except Exception as e:
            raise e
    print('Passed init tests on ' + str(correct_count) + \
            ' correct readings and ' + str(faulty_count) + ' faulty readings')
    systemd.daemon.notify('READY=1')

def detect_temp_drop():
    # Return list of the two latest temperatures using natural ordering
    diff_entries = list(entry.find().sort('$natural',-1).limit(2))

    temp_1 = diff_entries[0]['temperature']
    temp_2 = diff_entries[1]['temperature']

    date_1 = diff_entries[0]['time'].date()
    date_2 = diff_entries[1]['time'].date()

    # Return temp drop tolerance
    drop_tol = int(settings.find_one({"_id": "settings"})['drop'])

    # Check temp drop tolerance only if the two entries has same date
    if date_1 == date_2:
        temp_drop = temp_2 - temp_1
        if temp_drop > drop_tol:
            send_warning(drop_tol, temp_drop, date_1)

def create_message(sender, to, subject, message_text):
  message = MIMEText(message_text)
  message['to'] = to
  message['from'] = sender
  message['subject'] = subject
  return {'raw': base64.urlsafe_b64encode(message.as_string().encode()).decode()}

def send_warning(drop_tol, temp_drop, date_1):
    creds = None
    # The file token.json stores the user's access and refresh tokens, and is
    # created automatically when the authorization flow completes for the first
    # time.
    if os.path.exists('/home/alarm/token.json'):
        creds = Credentials.from_authorized_user_file('/home/alarm/token.json', SCOPES)
    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
```

```
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                '/home/alarm/credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.json', 'w') as token:
            token.write(creds.to_json())

    try:
        # Call the Gmail API
        service = build('gmail', 'v1', credentials=creds)
        to = "forsman.august@gmail.com"
        sender = "beermail2022@gmail.com"
        user_id = "me"
        subject = "Beer Temperature Warning"
        message_text = "WARNING, AUGUST!\nTemperature has dropped: " + str(temp_drop) + "C\nbetween the last two readings."
        message = create_message(sender, to, subject, message_text)
        # send_message(service, user_id, message)

    except HttpError as error:
        print(error)

def send_message(service, user_id, message):
  try:
    message = (service.users().messages().send(userId=user_id, body=message)
               .execute())
    print('Warning message sent')
    print(message['id'])
    return message
  except HttpError as error:
    print(error)

def main():
    init_script(temp)
    while True:
        time.sleep(10)
        try:
            temp_to_db(temp)
            detect_temp_drop()
            print("Current temp drop parameter: ")
            print(int(settings.find_one({"_id": "settings"})['drop']))
        except Exception as e:
            print(str(e))

''' ****************************************  '''

if __name__ == "__main__":
    ''' Wait for DB and Serial init '''
    time.sleep(2)
    main()
```

Listing 5.7: ~/Project/serial_temp_to_db.py