

FERMENTATION TEMPERATURE MONITORING

USING AN MCU AND A SINGLE-BOARD COMPUTER

Karl August Forsman

Abstract

In recent years, the number of small scale breweries has skyrocketed in numbers. This has led to an increased 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.

A traditional brewing style, often referred to as *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. 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 and 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 MCU to communicate with a Raspberry Pi (RPi) through serial communication. The signal should not contain any unnecessary noise and can therefor be filtered by the MCU before being transmitted to the server in a time series database.

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.

Contents

1	Introduction	4
1.1	Background	4
1.2	Purpose of the project	4
1.3	Work distribution and planning	4
1.4	Grading criteria	5
2	Working principle	6
2.1	Microcontroller	6
2.2	Single-board computer	6
2.3	Digital temperature sensor	7
2.4	UART communication	7
2.5	Web application	8
3	Implementation	9
3.1	Overview of the system	9
3.2	Hardware and components	10
3.3	Software and development tools	10
3.3.1	Neovim	10
3.3.2	Cross-compilation	10
3.3.3	Org-mode	11
3.3.4	AVR-GCC Toolchain	11
3.3.5	Git	12
3.3.6	Vimtex	12
3.4	Implementation	12
3.4.1	UART drivers	12
3.4.2	DS18B20 drivers	12
3.4.3	MongoDB	12
3.4.4	Systemd service	13
3.4.5	Plotly Dash web application	13
4	Result and discussions	15
5	Conclusions and futher work	16
5.1	Improvements	16

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
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
Workstation ThinkPad P14s G2 AMD Ryzen 7 PRO 5850U

1 Introduction

1.1 Background

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.

1.3 Work distribution and planning

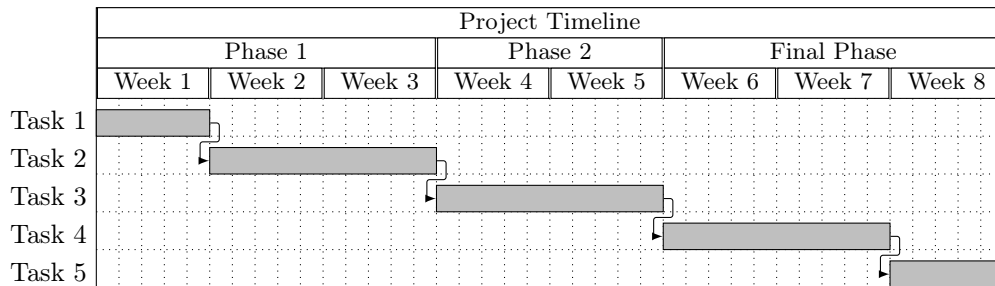


Figure 1: Gantt scheme showing the enumerated tasks during the project timespan

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. 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. 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.

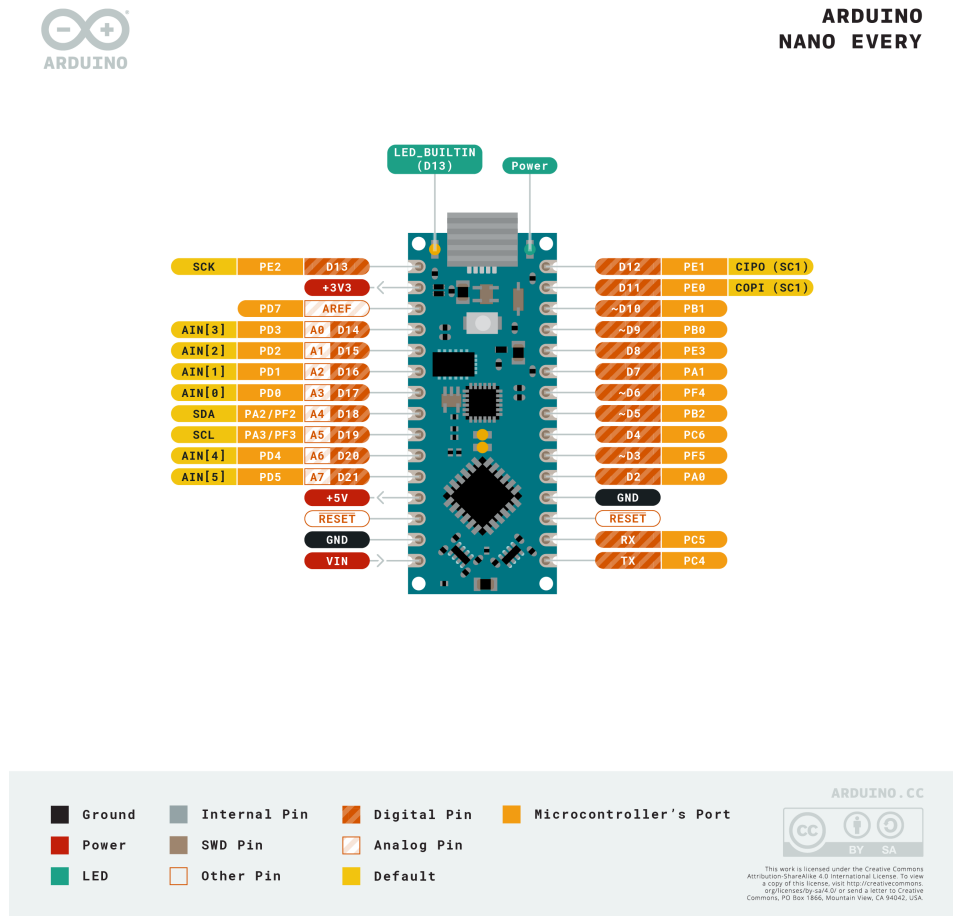


Figure 2: Arduino Nano board layout

According to the Figure 2, 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. 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.

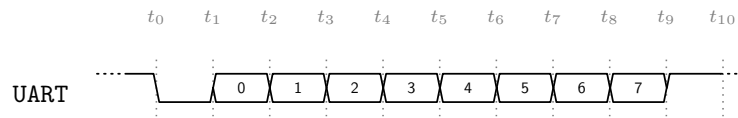


Figure 3: The bitwise UART communication initiates using a low start bit, eight bits of data and a high stop bit.

As seen in 3, the window length is so-and-so.

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. An example of an open-source project that handles cross-platforms well is Bootstrap, maintained by Twitter. Bootstrap is used both by major companies and hobbyists.

When managing the back-end of a web application, it is important to choose a suitable database that is both compatible with the eventual APIs as well as the type of data it is containing.

3 Implementation

3.1 Overview of the system

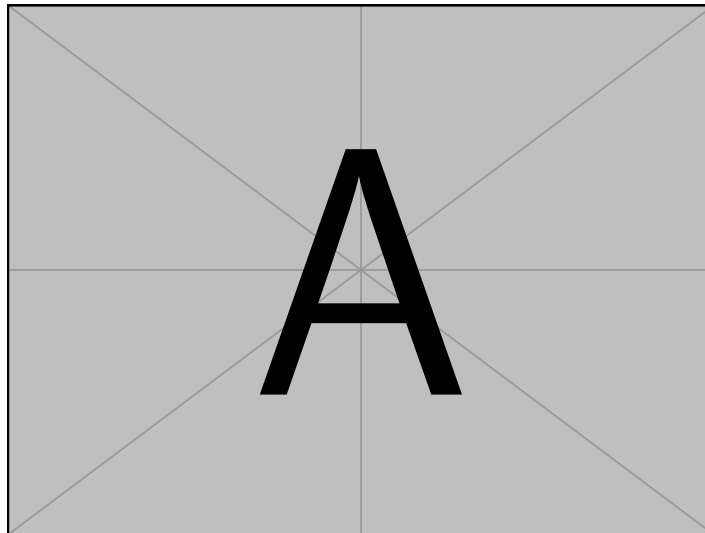


Figure 4: The system layout

The system overview can be seen in Figure 4.

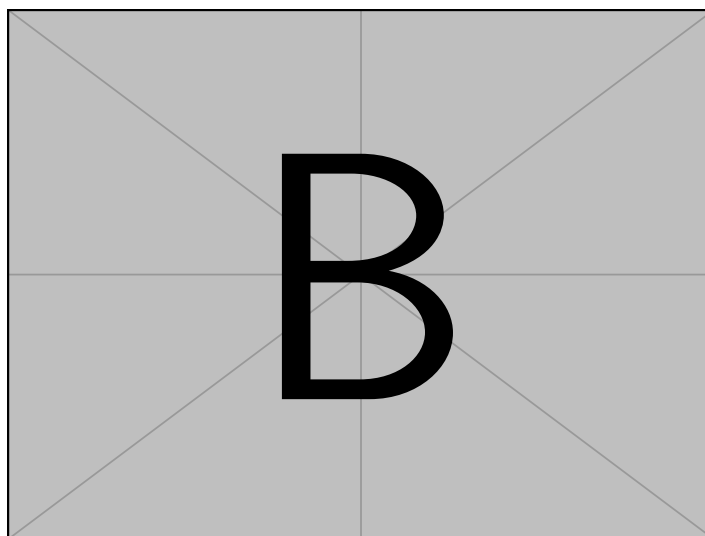


Figure 5: The Arduino and DS18B20 connected through a breadboard

3.2 Hardware and components

The software is written and compiled on a Lenovo ThinkPad P14s Gen 2 with an AMD Ryzen 7 PRO 5850U, referenced to as the *workstation*.

The database and web application is hosted on a Raspberry Pi 3B which has a 64-bit ARM Cortex v7 processor running Arch Linux ARM. The RPi is connected to a router and accessed locally through the SSH protocol from the workstation.

The MCU is an Arduino Nano board based on 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 4.7k Ω pull-up resistor.

3.3 Software and development tools

3.3.1 Neovim

Since Neovim is completely terminal based, 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.

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.

```
1 clang
2 -target
3 avr
4 -mmcu=atmega328p
5 -DF_CPU=16000000UL
6 -DBAUD=57600
7 -nostdinc
8 -ffreestanding
9 -isystem/usr/avr/include
10 -I./include
11 -I../inc
12 -I./inc
13 %h -x
14 %h c-header
```

Listing 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.

```
1 # Makefile for ATmega328p written by August Forsman
2
3 FILENAME      = main
```

```

4 HEADER1      = onewire
5 HEADER2      = uart
6 F_CPU        = 16000000UL
7 PORT         = /dev/ttyUSB0
8 DEVICE       = atmega328p
9 PROGRAMMER   = avrisp
10 BAUD        = 57600
11 CFLAGS       = -Wall -Os -ffreestanding
12 COMPILE      = avr-gcc $(CFLAGS) -mmcu=$(DEVICE) -DBAUD=$(BAUD) -DF_CPU
    =$(F_CPU) -I$(INCLUDE)
13 INCLUDE     = ./inc
14 OBJECTS      = obj/$(FILENAME).o obj/$(HEADER1).o obj/$(HEADER2).o
15
16
17 default: main.elf upload clean
18
19 $(FILENAME).elf: $(OBJECTS)
20     $(COMPILE) -o $(FILENAME).elf $(OBJECTS)
21     avr-objcopy -j .text -j .data -O ihex $(FILENAME).elf $(FILENAME).
    hex
22     avr-size --format=avr --mcu=$(DEVICE) $(FILENAME).elf
23
24 obj/$(FILENAME).o: src/$(FILENAME).c
25     $(COMPILE) -c src/$(FILENAME).c -o obj/$(FILENAME).o
26 obj/$(HEADER1).o: src/$(HEADER1).c inc/$(HEADER1).h
27     $(COMPILE) -c src/$(HEADER1).c -o obj/$(HEADER1).o
28 obj/$(HEADER2).o: src/$(HEADER2).c inc/$(HEADER2).h
29     $(COMPILE) -c src/$(HEADER2).c -o obj/$(HEADER2).o
30
31 upload:
32     avrdude -v -p $(DEVICE) -c $(PROGRAMMER) -P $(PORT) -b $(BAUD) -U
    flash:w:$(FILENAME).hex:i
33
34 .PHONY clean:
35     rm obj/*.o
36     rm $(FILENAME).elf
37     rm $(FILENAME).hex

```

Listing 2: GNU Makefile for compiling and flashing the program to the MCU

In Listing 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. 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 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 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 Raspberry Pi and the workstation.

3.3.6 Vimtex

The report is written in \LaTeX and compiled using the Neovim Vimtex plug-in.

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`. The byte-wise transfer and receive registers are also activated according to the datasheet.

```

1 UBRROH = UBRRH_VALUE ;
2 UBRROL = UBRUBRROL = UBRRL_VALUE ;
3
4 UCSROB = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0) | (1 << TXCIE0);

```

where

$$UBRR = \frac{f_{osc}}{16 \cdot BAUD} - 1. \quad (1)$$

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

Table 1: Baud and error rate		
$f_{osc} = 16000000 \text{ Hz}$		
Baud	UBRRn	Error
9600	103	0.2%
14400	68	0.6%
57600	18	2.1%

In Table 1, the error rate for a Baud rate of 9600 produces is low and suffices since the data is not sent in high frequency.

3.4.2 DS18B20 drivers

The DS18B20 digital temperature sensor is manufactured by Maxim Integrated and

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

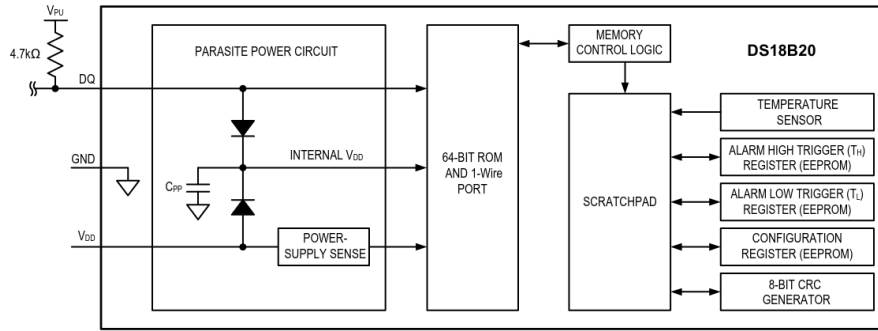


Figure 6: Schematic of the DS18B20 sensor

server is powered on. Using the Python module PyMongo, it is possible to insert entries using JSON formatting.

3.4.4 Systemd service

Many Linux distributions use the Systemd init daemon. 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 *journal*. The latter simplifies tracking how often the service behaves unexpectedly during long timespans.

```

1 [Unit]
2 # Human readable unit name
3 Description=Reads serially from '/dev/ttyUSB*' and puts in MongoDB
4
5 [Service]
6 # Command that executes script
7 ExecStart=/usr/bin/python /home/alarm/Project/serial_temp_to_db.py
8 # Redirect print() to the Linux journal
9 Environment=PYTHONUNBUFFERED=1
10 # Able to notify that the service is ready
11 Type=notify
12 Restart=always
13
14 [Install]
15 # Start service at boot
16 WantedBy=default.target

```

Listing 3: The systemd service managing data collection

3.4.5 Plotly Dash web application

The web application consists of

1. A live graph showing the n most recent data points. It is updated by a timed callback function
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.

Each website object can be decorated with a callback function that either updates due to a time interval or through user interaction.

In order to utilize the power of Pandas, the data is loaded into a dataframe which allows many types of manipulation in areas such as statistics and signal processing.

Since the data ordered by a timestamp, it is sorted in natural order and the start and stop date can be chosen by the user.

4 Result and discussions

The resulting product is an open-source temperature monitoring system suitable for beer fermentation. Users of the web application can filter through time-series data by filtering the start and end date using callback functions communicating 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.

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

5 Conclusions and futher work

The system provides a user-friendly front end where the data can be visualized, accessed and monitored through custom settings.

5.1 Improvements

For a three year time-series sampled every hour it results in 26280 data points. When using a large sampling window, the `savgol` filter is quite intensive to compute for the RPi.

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.
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.

```

1 #include <avr/io.h>
2 #include <util/delay.h>
3 #include <avr/interrupt.h>
4 #include "onewire.h"
5 #include "uart.h"
6
7 #define MSG_LEN_Y 17
8 #define MSG_LEN_N 14
9
10 void main(void)
11 {
12     sei();
13     uart_init();
14     ow_reset();
15     uint8_t data;
16     uint16_t len = 8;
17     char buffer[len];
18
19     for(;;)
20     {
21         /* Polling for 'R' */
22         if (uart_read_count() > 0)
23         {
24             data = uart_read();
25             if (data == 'R') {
26                 ow_reset();
27                 ow_temp_rd(buffer);
28                 uart_send_arr(buffer, len);
29                 uart_send_byte('\r');
30                 uart_send_byte('\n');
31             }
32         }
33     }
34 }

```

/Project/DS18B20_UART/src/main.c

```

1 #include "uart.h"
2
3 /*Global volatile variables for this file*/
4 volatile static uint8_t uart_tx_active = 1;
5 volatile static uint8_t rx_buf[RX_BUF_SZ] = {0};
6 volatile static uint16_t rx_count = 0;
7
8 /* Recv interrupt */
9 ISR(USART_RX_vect)
10 {
11     volatile static uint16_t rx_write_pos = 0;
12
13     rx_buf[rx_write_pos] = UDR0;
14     rx_count++;
15     if (rx_write_pos >= RX_BUF_SZ)

```

```

16     {
17         rx_write_pos = 0;
18     }
19
20 }
21 /* Transf interrupt */
22 ISR(USART_TX_vect)
23 {
24     uart_tx_active = 1;
25
26 }
27
28 void uart_init(void)
29 {
30     #if SPEED2X
31         UCSRA |= 1 << U2X0;
32     #else
33         UCSRA &= ~(1 << U2X0);
34     #endif
35     /* Baud rate helpers (set baud.h) */
36     UBRROH = UBRRH_VALUE;
37     UBRROL = UBRRL_VALUE;
38
39     UCSRB = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0) | (1 << TXCIE0)
40         ;
41 }
42
43 void uart_send_byte(uint8_t c)
44 {
45     while (!uart_tx_active);
46     uart_tx_active = 0;
47     UDR0 = c;
48 }
49
50 void uart_send_arr(char *c, uint16_t len)
51 {
52     for (uint16_t i = 0; i < len; i++)
53     {
54         uart_send_byte(c[i]);
55     }
56 }
57
58 void uart_send_str(uint8_t *c)
59 {
60     uint16_t i = 0;
61     do
62     {
63         uart_send_byte(c[i]);
64         i++;
65     } while (c[i] != '\0');
66 }

```

```

67
68 uint16_t uart_read_count(void)
69 {
70     return rx_count;
71 }
72
73 uint8_t uart_read(void)
74 {
75     static uint16_t rx_read_pos = 0;
76     uint8_t data = 0;
77     data = rx_buf[rx_read_pos];
78     rx_read_pos++;
79     rx_count--;
80     if (rx_read_pos)
81     {
82         rx_read_pos = 0;
83     }
84     return data;
85 }

```

/Project/DS18B20_UART/src/uart.c

```

1  /*
2   2/2022 - Modified for use in a temperature measuring project
3   conducted by August Forsman (auan(at)mailbox.org).
4
5   For the full library, see the contributions made by
6
7   Access Dallas 1-Wire Devices with ATMEL AVR's
8   Author of the initial code: Peter Dannegger (dannu(at)specs.de)
9   modified by Martin Thomas (mthomas(at)rhrk.uni-kl.de)
10  9/2004 - use of delay.h, optional bus configuration at runtime
11  10/2009 - additional delay in ow_bit_io for recovery
12  5/2010 - timing modifications, additional config-values and comments,
13           use of atomic.h macros, internal pull-up support
14  7/2010 - added method to skip recovery time after last bit transfered
15           via ow_command_skip_last_recovery
16  */
17
18
19 #include <avr/io.h>
20 #include <util/delay.h>
21 #include <util/atomic.h>
22 #include <stdio.h>
23
24 #include "onewire.h"
25 #include "uart.h"
26
27 #define MAX_FLOAT 127.9375
28
29 #ifdef OW_ONE_BUS
30

```

```

31 #define OW_GET_IN()    ( OW_IN & (1<<OW_PIN))
32 #define OW_OUT_LOW()  ( OW_OUT &= (~(1 << OW_PIN)) )
33 #define OW_OUT_HIGH() ( OW_OUT |= (1 << OW_PIN) )
34 #define OW_DIR_IN()   ( OW_DDR &= (~(1 << OW_PIN)) )
35 #define OW_DIR_OUT()  ( OW_DDR |= (1 << OW_PIN) )
36
37 #else
38
39 /* set bus-config with ow_set_bus() */
40 uint8_t OW_PIN_MASK;
41 volatile uint8_t* OW_IN;
42 volatile uint8_t* OW_OUT;
43 volatile uint8_t* OW_DDR;
44
45 #define OW_GET_IN()    ( *OW_IN & OW_PIN_MASK )
46 #define OW_OUT_LOW()  ( *OW_OUT &= (uint8_t) ~OW_PIN_MASK )
47 #define OW_OUT_HIGH() ( *OW_OUT |= (uint8_t)  OW_PIN_MASK )
48 #define OW_DIR_IN()   ( *OW_DDR &= (uint8_t) ~OW_PIN_MASK )
49 #define OW_DIR_OUT()  ( *OW_DDR |= (uint8_t)  OW_PIN_MASK )
50
51 void ow_set_bus(volatile uint8_t* in,
52                volatile uint8_t* out,
53                volatile uint8_t* ddr,
54                uint8_t pin)
55 {
56     OW_DDR=ddr;
57     OW_OUT=out;
58     OW_IN=in;
59     OW_PIN_MASK = (1 << pin);
60     ow_reset();
61 }
62
63 #endif
64
65 uint8_t ow_input_pin_state()
66 {
67     return OW_GET_IN();
68 }
69
70 void ow_parasite_enable(void)
71 {
72     OW_OUT_HIGH();
73     OW_DIR_OUT();
74 }
75
76 void ow_parasite_disable(void)
77 {
78     OW_DIR_IN();
79 #if (!OW_USE_INTERNAL_PULLUP)
80     OW_OUT_LOW();
81 #endif
82 }

```

```

83
84
85 uint8_t ow_reset(void)
86 {
87     uint8_t err;
88
89     OW_OUT_LOW();
90     OW_DIR_OUT();           // pull OW-Pin low for 480us
91     _delay_us(480);
92
93     ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
94         // set Pin as input - wait for clients to pull low
95         OW_DIR_IN(); // input
96 #if OW_USE_INTERNAL_PULLUP
97         OW_OUT_HIGH();
98 #endif
99
100     _delay_us(64);          // was 66
101     err = OW_GET_IN();      // no presence detect
102                             // if err!=0: nobody pulled to low, still
103     high
104 }
105
106 // after a delay the clients should release the line
107 // and input-pin gets back to high by pull-up-resistor
108 _delay_us(480 - 64);        // was 480-66
109 if( OW_GET_IN() == 0 ) {
110     err = 1;                // short circuit, expected low but got high
111 }
112
113 // TODO check if this works
114 /*return (OW_GET_IN() == 0) ? err = 1:0;*/
115 return err;
116 }
117
118 /* Timing issue when using runtime-bus-selection (!OW_ONE_BUS):
119 The master should sample at the end of the 15-slot after initiating
120 the read-time-slot. The variable bus-settings need more
121 cycles than the constant ones so the delays had to be shortened
122 to achieve a 15uS overall delay
123 Setting/clearing a bit in I/O Register needs 1 cycle in OW_ONE_BUS
124 but around 14 cycles in configurable bus (us-Delay is 4 cycles per
125 uS) */
126 static uint8_t ow_bit_io_intern( uint8_t b, uint8_t
127     with_parasite_enable )
128 {
129     ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
130 #if OW_USE_INTERNAL_PULLUP
131         OW_OUT_LOW();
132 #endif
133     OW_DIR_OUT();           // drive bus low

```

```

132     _delay_us(2);      // T_INT > 1usec accoding to timing-diagramm
133     if ( b ) {
134         OW_DIR_IN(); // to write "1" release bus, resistor pulls high
135 #if OW_USE_INTERNAL_PULLUP
136         OW_OUT_HIGH();
137 #endif
138     }
139
140     // "Output data from the DS18B20 is valid for 15usec after the
141     // edge that initiated the read time slot. Therefore, the master
142     // must
143     // release the bus and then sample the bus state within 15ussec
144     // from
145     // the start of the slot."
146     _delay_us(15-2-OW_CONF_DELAYOFFSET);
147
148     if( OW_GET_IN() == 0 ) {
149         b = 0; // sample at end of read-timeslot
150     }
151
152     _delay_us(60-15-2+OW_CONF_DELAYOFFSET);
153 #if OW_USE_INTERNAL_PULLUP
154     OW_OUT_HIGH();
155 #endif
156     OW_DIR_IN();
157
158     if ( with_parasite_enable ) {
159         ow_parasite_enable();
160     }
161 } /* ATOMIC_BLOCK */
162
163 _delay_us(OW_RECOVERY_TIME); // may be increased for longer wires
164
165 return b;
166 }
167
168 uint8_t ow_bit_io( uint8_t b )
169 {
170     return ow_bit_io_intern( b & 1, 0 );
171 }
172
173 uint8_t ow_byte_wr( uint8_t b )
174 {
175     uint8_t i = 8, j;
176
177     do {
178         j = ow_bit_io( b & 1 );
179         b >>= 1;
180         if( j ) {
181             b |= 0x80;

```

```

181     }
182 } while( --i );
183
184 return b;
185 }
186
187 uint8_t ow_byte_wr_with_parasite_enable( uint8_t b )
188 {
189     uint8_t i = 8, j;
190
191     do {
192         if ( i != 1 ) {
193             j = ow_bit_io_intern( b & 1, 0 );
194         } else {
195             j = ow_bit_io_intern( b & 1, 1 );
196         }
197         b >>= 1;
198         if( j ) {
199             b /= 0x80;
200         }
201     } while( --i );
202
203     return b;
204 }
205
206
207 uint8_t ow_byte_rd( void )
208 {
209     // read by sending only "1"s, so bus gets released
210     // after the init low-pulse in every slot
211     return ow_byte_wr( 0xFF );
212 }
213
214
215 uint8_t ow_rom_search( uint8_t diff, uint8_t *id )
216 {
217     uint8_t i, j, next_diff;
218     uint8_t b;
219
220     if( ow_reset() ) {
221         return OW_PRESENCE_ERR;           // error, no device found <---
222         early exit!
223     }
224
225     ow_byte_wr( OW_SEARCH_ROM );         // ROM search command
226     next_diff = OW_LAST_DEVICE;          // unchanged on last device
227
228     i = OW_ROMCODE_SIZE * 8;             // 8 bytes
229
230     do {
231         j = 8;                           // 8 bits
232         do {

```

```

232     b = ow_bit_io( 1 );           // read bit
233     if( ow_bit_io( 1 ) ) {       // read complement bit
234         if( b ) {                 // 0b11
235             return OW_DATA_ERR; // data error <--- early exit!
236         }
237     }
238     else {
239         if( !b ) {                 // 0b00 = 2 devices
240             if( diff > i || ((*id & 1) && diff != i) ) {
241                 b = 1;             // now 1
242                 next_diff = i;     // next pass 0
243             }
244         }
245     }
246     ow_bit_io( b );               // write bit
247     *id >>= 1;
248     if( b ) {
249         *id |= 0x80;               // store bit
250     }
251
252     i--;
253
254     } while( --j );
255
256     id++;                          // next byte
257
258     } while( i );
259
260     return next_diff;              // to continue search
261 }
262
263
264 static void ow_command_intern( uint8_t command, uint8_t *id, uint8_t
    with_parasite_enable )
265 {
266     uint8_t i;
267
268     ow_reset();
269
270     if( id ) {
271         ow_byte_wr( OW_MATCH_ROM ); // to a single device
272         i = OW_ROMCODE_SIZE;
273         do {
274             ow_byte_wr( *id );
275             id++;
276         } while( --i );
277     }
278     else {
279         ow_byte_wr( OW_SKIP_ROM ); // to all devices
280     }
281
282     if ( with_parasite_enable ) {

```



```

283     ow_byte_wr_with_parasite_enable( command );
284 } else {
285     ow_byte_wr( command );
286 }
287 }
288
289 void ow_command( uint8_t command, uint8_t *id )
290 {
291     ow_command_intern( command, id, 0);
292 }
293
294 void ow_command_with_parasite_enable( uint8_t command, uint8_t *id )
295 {
296     ow_command_intern( command, id, 1 );
297 }
298
299 void ow_temp_rd(char *buffer)
300 {
301     // 12bytes long
302     uint8_t temperature[2];
303     int8_t digit;
304     uint16_t decimal;
305
306     // Reset, skip ROM, start conversion
307     ow_reset();
308     ow_byte_wr(OW_SKIP_ROM);
309     ow_byte_wr(OW_CONVERTTEMP);
310
311     ow_reset();
312     ow_byte_wr(OW_SKIP_ROM);
313     ow_byte_wr(OW_RSCRATCHPAD);
314
315     // Read scratchpad
316     temperature[0] = ow_byte_rd();
317     temperature[1] = ow_byte_rd();
318     ow_reset();
319
320
321     // Store temperature int and dec digits
322     digit = temperature[0] >> 4;
323     digit |= (temperature[1] & 0b00000111) << 4;
324     if (temperature[1] > 0b01111111) {
325         digit = digit - MAX_FLOAT;
326     }
327
328     // Store dec digits
329     decimal = temperature[0] & 0xf;
330     decimal *= OW_DEC_STEP_12BIT;
331
332     sprintf(buffer, "%d.%04u", digit, decimal);
333 }

```

```
1 #ifndef UART_H_
2 #define UART_H_
3 /* TODO get rid of this redefinition */
4 #define BAUD 9600
5 #include <avr/io.h>
6 #include <util/delay.h>
7 #include <util/setbaud.h>
8 #include <avr/interrupt.h>
9
10 #define RX_BUF_SZ 12
11
12 void uart_init(void);
13 void uart_send_byte(uint8_t c);
14 void uart_send_str(uint8_t *c);
15 void uart_send_arr(char *c, uint16_t len);
16
17 uint16_t uart_read_count(void);
18 uint16_t uart_read_count(void);
19 uint8_t uart_read(void);
20
21 #endif /* ifndef UART_H_ */
```

```
1 #ifndef ONEWIRE_H_
2 #define ONEWIRE_H_
3
4 #ifdef __cplusplus
5 extern "C" {
6 #endif
7
8 #include <stdint.h>
9
10 /* ***** */
11 /* Hardware connection */
12 /* ***** */
13
14 /* Define OW_ONE_BUS if only one 1-Wire-Bus is used
15    in the application -> shorter code.
16    If not defined make sure to call ow_set_bus() before using
17    a bus. Runtime bus-select increases code size by around 300
18    bytes so use OW_ONE_BUS if possible */
19 #define OW_ONE_BUS
20
21 #ifdef OW_ONE_BUS
22
23 #define OW_PIN 0
24 #define OW_IN PINC
25 #define OW_OUT PORTC
```

```

26 #define OW_DDR DDRC
27 #define OW_CONF_DELAYOFFSET 0
28
29 #else
30 #if ( F_CPU < 1843200 )
31 #warning | Experimental multi-bus-mode is not tested for
32 #warning | frequencies below 1,84MHz. Use OW_ONE_WIRE or
33 #warning | faster clock-source (i.e. internal 2MHz R/C-Osc.).
34 #endif
35 #define OW_CONF_CYCLES PERACCESS 13
36 #define OW_CONF_DELAYOFFSET ( (uint16_t)( ((OW_CONF_CYCLES PERACCESS) *
      1000000L) / F_CPU ) )
37 #endif
38
39 // Recovery time (T_Rec) minimum 1usec - increase for long lines
40 // 5 usecs is a value give in some Maxim AppNotes
41 // 30u secs seem to be reliable for longer lines
42 // #define OW_RECOVERY_TIME 5 /* usec */
43 // #define OW_RECOVERY_TIME 300 /* usec */
44 #define OW_RECOVERY_TIME 10 /* usec */
45
46 // Use AVR's internal pull-up resistor instead of external 4,7k
   resistor.
47 // Based on information from Sascha Schade. Experimental but worked in
   tests
48 // with one DS18B20 and one DS18S20 on a rather short bus (60cm),
   where both
49 // sensores have been parasite-powered.
50 #define OW_USE_INTERNAL_PULLUP 0 /* 0=external, 1=internal */
51
52 /*****/
53
54
55 #define OW_MATCH_ROM 0x55
56 #define OW_SKIP_ROM 0xCC
57 #define OW_SEARCH_ROM 0xF0
58 #define OW_RSCRATCHPAD 0xBE
59 #define OW_CONVERTTEMP 0x44
60
61 #define OW_SEARCH_FIRST 0xFF // start new search
62 #define OW_PRESENCE_ERR 0xFF
63 #define OW_DATA_ERR 0xFE
64 #define OW_LAST_DEVICE 0x00 // last device found
65
66 #define OW_DEC_STEP_12BIT 625
67
68 // rom-code size including CRC
69 #define OW_ROMCODE_SIZE 8
70
71 extern uint8_t ow_reset(void);
72
73 extern uint8_t ow_bit_io( uint8_t b );

```

```

74 extern uint8_t ow_byte_wr( uint8_t b );
75 extern uint8_t ow_byte_rd( void );
76 extern void ow_temp_rd(char *buffer);
77
78 extern uint8_t ow_rom_search( uint8_t diff, uint8_t *id );
79
80 extern void ow_command( uint8_t command, uint8_t *id );
81 extern void ow_command_with_parasite_enable( uint8_t command, uint8_t
      *id );
82
83 extern void ow_parasite_enable( void );
84 extern void ow_parasite_disable( void );
85 extern uint8_t ow_input_pin_state( void );
86
87 #ifndef OW_ONE_BUS
88 extern void ow_set_bus( volatile uint8_t* in,
89     volatile uint8_t* out,
90     volatile uint8_t* ddr,
91     uint8_t pin );
92 #endif
93
94 #ifdef __cplusplus
95 }
96 #endif
97
98 #endif

```

/Project/DS18B20_UART/inc/onewire.h

```

1  #!/bin/python
2  import dash
3  from dash import dcc
4  from dash import html
5  from dash.dependencies import Input, Output, State
6  import serial
7  from pymongo import MongoClient
8  from datetime import datetime as dt
9  from datetime import date
10 import numpy as np
11 import time
12 import pprint
13 import pandas as pd
14 import plotly
15 from random import random
16 import plotly.graph_objs as go
17 import re
18 from scipy.signal import savgol_filter
19
20 ''' ***** '''
21
22 # TODO add constants
23 BAUD = 9600

```

```

24 IP_ADR = 27017
25 HOUR_MS = 60*600000
26 PADDING = 150
27 BG_COLOR = '#FFFFFF'
28
29 ''' MongoDB '''
30 client = MongoClient('localhost', IP_ADR)
31 db = client.beertemp
32 settings = db.settings
33 entry = db.entries
34
35 ''' ***** '''
36
37 external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
38
39 app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
40
41 app.layout = html.Div(style={'padding': PADDING, 'background' :
    BG_COLOR}, children =[
42     html.H1('Beer Temperature Logging'),
43     html.H3('This is a live feed!'),
44     html.Div([
45         dcc.Graph(id='live-update-graph-scatter', animate=False),
46         html.Hr(),
47         dcc.Interval(
48             id='interval-component',
49             interval=1*HOUR_MS
50         )
51     ]),
52     html.Div([
53         html.H3('Set temperature alarm value'),
54         # dcc.Input(
55             # id="input_range_min", type="number", debounce=True,
56             placeholder="Set min temp",
57             # min=-5, max=40, step=1, value=int(settings.find_one({"
58             _id": "settings"})['min'])
59             # ),
60         # dcc.Input(
61             # id="input_range_max", type="number", debounce=True,
62             placeholder="Set max temp",
63             # min=-5, max=40, step=1, value=int(settings.find_one({"
64             _id": "settings"})['max'])
65             # ),
66         dcc.Input(
67             id="input_range_drop", type="number", debounce=True,
68             placeholder="Temp drop tolerance",
69             min=1, max=40, step=1, value=int(settings.find_one({"_id":
70             "settings"})['drop'])
71         ),
72         # html.Div(id="min-max-out"),
73         html.Div(id="drop-out"),
74         html.H3('Set start and end date'),

```

```

69     dcc.DatePickerRange(
70         id='date-picker-range',
71         min_date_allowed = entry.find_one()['time'].date(),
72         max_date_allowed = list(entry.find().sort('$natural',-1).
limit(1))[0]['time'].date(),
73
74         # TODO timedelta based fallback
75         start_date = date(2022,2,28),
76         end_date = date(2022,3,4)
77     ),
78     dcc.Graph(id='history-graph-scatter', animate=False),
79     html.Div(id='output-container-date-picker-range')
80 ]),
81 ])
82
83
84 @app.callback(Output('live-update-graph-scatter', 'figure'),
85               Input('interval-component', 'n_intervals'))
86 def update_graph_scatter(graph_update):
87     LIVE_RES = 200
88
89     try:
90         df = pd.DataFrame(list(db.entries.find().limit(int(LIVE_RES)).
sort([('natural',-1)])))
91         trace = go.Scatter(
92             x=df['time'],
93             y=df['temperature'],
94             name='Beer temperature',
95             mode= 'lines+markers',
96             marker = {'color': 'Blue',
97                       'size': 4}
98         )
99     except Exception as e:
100         print(str(e))
101
102     layout = go.Layout(
103         paper_bgcolor='rgba(0,0,0,0)',
104         plot_bgcolor='rgba(0,0,0,0)',
105         margin=dict(l=20, r=20, t=50, b=50),
106         yaxis= {'autorange': True}
107     )
108
109     fig = go.Figure(data=[trace], layout=layout)
110
111     fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='Grey')
112     fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='Grey')
113
114     return fig
115
116 # TODO callback to apply Savgol filtering
117 @app.callback(Output('history-graph-scatter', 'figure'),
118               [Input(component_id='date-picker-range', component_property='

```

```

119         start_date'),
        Input(component_id='date-picker-range', component_property='
end_date'))]
120     )
121 # @app.callback(Output('live-update-graph-scatter', 'figure'))
122 def update_history(start_date, end_date):
123     start_date = pd.to_datetime(start_date)
124     end_date = pd.to_datetime(end_date)
125     df = pd.DataFrame(list(db.entries.find().sort([('_id', -1)])))
126
127     filtered_df = df[df['time'].between(
128         dt.strptime(start_date, "%Y-%m-%d"),
129         dt.strptime(end_date, "%Y-%m-%d")
130     )]
131
132     SAVGOL_WIN_LEN = 201
133
134     try:
135         # trace1 = go.Scatter(
136             # x=df['time'],
137             # y=df['temperature'],
138             # name='Beer temperature',
139             # mode= 'markers',
140             # marker = {'color': 'Blue',
141                 # 'size': 3}
142             # )
143
144         trace2 = go.Scatter(
145             x=filtered_df['time'],
146             # y=savgol_filter(filtered_df['temperature'],
SAVGOL_WIN_LEN, 2),
147             y=filtered_df['temperature'],
148             name='Sensor signal [C]',
149             # name='Filtered signal',
150             mode= 'lines+markers',
151             marker = {'color': 'Red',
152                 'size': 2}
153             )
154     except Exception as e:
155         print(str(e))
156
157     layout = go.Layout(
158         paper_bgcolor='rgba(0,0,0,0)',
159         plot_bgcolor='rgba(0,0,0,0)',
160         margin=dict(l=20, r=20, t=50, b=50),
161         yaxis= {'autorange': True}
162     )
163
164     # fig = go.Figure(data=[trace1, trace2], layout=layout)
165     fig = go.Figure(data=trace2, layout=layout)
166     min_temp = filtered_df['temperature'].min()
167     max_temp = filtered_df['temperature'].max()

```

```

168 mean_temp = filtered_df['temperature'].mean()
169
170 fig.add_annotation(text="Min = " + str(min_temp) + " C",
171 xref="paper", yref="paper",
172 x=0.1, y=1, showarrow=False,
173 font=dict(
174     family="Courier New, monospace",
175     size=16,
176     color="#ffffff"
177 ),
178 bordercolor="#c7c7c7",
179 borderwidth=2,
180 borderpad=4,
181 bgcolor="red",
182 opacity=0.8
183 )
184
185 fig.add_annotation(text="Max = " + str(max_temp) + " C",
186 xref="paper", yref="paper",
187 x=0.9, y=1, showarrow=False,
188 font=dict(
189     family="Courier New, monospace",
190     size=16,
191     color="#ffffff"
192 ),
193 bordercolor="#c7c7c7",
194 borderwidth=2,
195 borderpad=4,
196 bgcolor="red",
197 opacity=0.8
198 )
199
200 fig.add_annotation(text="Mean = " + str(round(mean_temp, 3)) + " C",
201 xref="paper", yref="paper",
202 x=0.5, y=1, showarrow=False,
203 font=dict(
204     family="Courier New, monospace",
205     size=16,
206     color="#ffffff"
207 ),
208 bordercolor="#c7c7c7",
209 borderwidth=2,
210 borderpad=4,
211 bgcolor="red",
212 opacity=0.8
213 )
214 fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='Grey')
215 fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='Grey')
216
217 return fig
218

```



```

219 # @app.callback(
220     # Output("min-max-out", "children"),
221     # Input("input_range_min", "value"),
222     # Input("input_range_max", "value"),
223 # )
224 # def set_temp_min_max(rangemin, rangemax):
225     # # if rangemin or rangemax == None:
226     # #     return "Temperature(s) out of range"
227     # if rangemin >= rangemax:
228     #     return "Min has to be lower than max"
229     # else:
230     #     settings.update_one({"_id": "settings"}, {"$set":{"min":
rangemin, "max": rangemax}})
231     #     return "Max temp: {}C | Min temp: {}C".format(rangemin,
rangemax)
232
233 @app.callback(
234     Output("drop-out", "children"),
235     Input("input_range_drop", "value"),
236 )
237 def set_temp_drop(rangedrop):
238     if rangedrop == None:
239         return "Temp drop out of range"
240     else:
241         settings.update_one({"_id": "settings"}, {"$set":{"drop":
rangedrop}})
242         return "Warning if temperature drops >= {} C".format(rangedrop)
243
244 if __name__ == '__main__':
245     app.run_server(debug=True, host="0.0.0.0")

```

/Project/livedb.py

```

1 import serial
2 from pymongo import MongoClient
3 from datetime import datetime as dt
4 import time
5 import systemd.daemon
6 import os.path
7 import base64
8 from email.mime.text import MIMEText
9 from google.auth.transport.requests import Request
10 from google.oauth2.credentials import Credentials
11 from google_auth_oauthlib.flow import InstalledAppFlow
12 from googleapiclient.discovery import build
13 from googleapiclient.errors import HttpError
14
15 # If modifying these scopes, delete the file token.json.
16 SCOPES = ['https://www.googleapis.com/auth/gmail.send']
17
18 # Initilization temperature. 0C looks better than a 'trash output'.

```

```

19 temp = 0
20
21 ''' Constants '''
22 MAX_FERM_TEMP = 50
23 BAUD = 9600
24 PORT = "/dev/ttyUSB0"
25 READ_CMD = b'R'
26 MIN_FLOAT_LEN = 9
27 MAX_FLOAT_LEN = 11
28 IP_ADR = 27017
29 CORRECT_TESTS = 3
30
31 ''' MongoDB '''
32 client = MongoClient('localhost', IP_ADR)
33 db = client.beertemp
34 entry = db.entries
35 settings = db.settings
36
37 # Open serial connection to MCU
38 ser = serial.Serial(PORT, BAUD)
39
40 def get_temperature(temp):
41     ser.write(bytes(READ_CMD))
42     line = ser.readline().decode()
43     if len(line) > MIN_FLOAT_LEN \
44         and len(line) < MAX_FLOAT_LEN \
45         and float(line.strip('\x00\n\r')) < MAX_FERM_TEMP:
46         temp = (float(line.strip('\x00\n\r')))
47     return temp
48
49 def temp_to_db(temp):
50     temp_entry = {"temperature": get_temperature(temp),
51                  "time": dt.utcnow()}
52     entry.insert_one(temp_entry)
53
54 def init_script(temp):
55     print('Running tests on port: ' + ser.portstr)
56     print('Baud rate = ' + str(BAUD))
57     correct_count = 0
58     faulty_count = 0
59     while correct_count != CORRECT_TESTS:
60         try:
61             time.sleep(0.5)
62             ser.write(READ_CMD)
63             line = ser.readline().decode()
64             if len(line) > MIN_FLOAT_LEN \
65                 and len(line) < MAX_FLOAT_LEN \
66                 and float(line.strip('\x00\n\r')) < MAX_FERM_TEMP:
67                 print('Correct reading: ' + line.strip('\x00\n\r'))
68                 correct_count+=1
69             else:
70                 print('Faulty reading: ' + line.strip('\x00\n\r'))

```

```

71         faulty_count+=1
72     except Exception as e:
73         raise e
74     print('Passed init tests on ' + str(correct_count) + \
75           ' correct readings and ' + str(faulty_count) + ' faulty
readings')
76     systemd.daemon.notify('READY=1')
77
78 def detect_temp_drop():
79     # Return list of the two latest temperatures using natural
ordering
80     diff_entries = list(entry.find().sort('$natural',-1).limit(2))
81
82     temp_1 = diff_entries[0]['temperature']
83     temp_2 = diff_entries[1]['temperature']
84
85     date_1 = diff_entries[0]['time'].date()
86     date_2 = diff_entries[1]['time'].date()
87
88     # Return temp drop tolerance
89     drop_tol = int(settings.find_one({"_id": "settings"})['drop'])
90
91     # Check temp drop tolerance only if the two entries has same date
92     if date_1 == date_2:
93         temp_drop = temp_2 - temp_1
94         if temp_drop > drop_tol:
95             send_warning(drop_tol, temp_drop, date_1)
96
97 def create_message(sender, to, subject, message_text):
98     message = MIMEText(message_text)
99     message['to'] = to
100    message['from'] = sender
101    message['subject'] = subject
102    return {'raw': base64.urlsafe_b64encode(message.as_string().encode()
).decode() }
103
104 def send_warning(drop_tol, temp_drop, date_1):
105     creds = None
106     # The file token.json stores the user's access and refresh tokens,
and is
107     # created automatically when the authorization flow completes for
the first
108     # time.
109     if os.path.exists('/home/alarm/token.json'):
110         creds = Credentials.from_authorized_user_file('/home/alarm/
token.json', SCOPES)
111     # If there are no (valid) credentials available, let the user log
in.
112     if not creds or not creds.valid:
113         if creds and creds.expired and creds.refresh_token:
114             creds.refresh(Request())
115         else:

```

```

116         flow = InstalledAppFlow.from_client_secrets_file(
117             '/home/alarm/credentials.json', SCOPES)
118         creds = flow.run_local_server(port=0)
119         # Save the credentials for the next run
120         with open('token.json', 'w') as token:
121             token.write(creds.to_json())
122
123     try:
124         # Call the Gmail API
125         service = build('gmail', 'v1', credentials=creds)
126         to = "forsman.august@gmail.com"
127         sender = "beermail2022@gmail.com"
128         user_id = "me"
129         subject = "Beer Temperature Warning"
130         message_text = "WARNING, AUGUST!\nTemperature has dropped: " +
131             str(temp_drop) + "C\nbetween the last two readings."
132         message = create_message(sender, to, subject, message_text)
133         # send_message(service, user_id, message)
134
135     except HttpError as error:
136         print(error)
137
138 def send_message(service, user_id, message):
139     try:
140         message = (service.users().messages().send(userId=user_id, body=
141             message)
142             .execute())
143         print('Warning message sent')
144         print(message['id'])
145         return message
146     except HttpError as error:
147         print(error)
148
149 def main():
150     init_script(temp)
151     while True:
152         time.sleep(10)
153         try:
154             temp_to_db(temp)
155             detect_temp_drop()
156             print("Current temp drop parameter: ")
157             print(int(settings.find_one({"_id": "settings"})['drop']))
158         except Exception as e:
159             print(str(e))
160
161 ''' ***** '''
162 if __name__ == "__main__":
163     ''' Wait for DB and Serial init '''
164     time.sleep(2)
165     main()

```

/Project/serial_temp_to_db.py