PHASE 2 PRESENTATION

PROJECT IN EMBEDDED SYSTEMS — 1TE721

UPPSALA
UNIVERSITET

SUMMARY OF PHASE 2

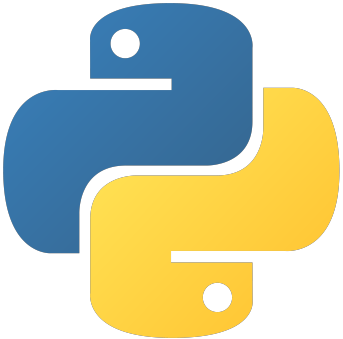
The work is finished according to the project plan.

1. Tuning the UART communication between the ATmega328p MCU and Raspberry Pi
2. Choosing and configuring a suitable database and entry formatting
3. Fail safe data collection
4. Testing the web application visualization with live data

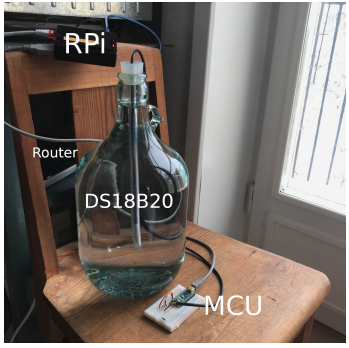
DATA COLLECTION AND VISUALIZATION

A selection of used Python modules...
Down the dependency hole, we go!

1. PySerial – serial communication through UART
2. pyMongo – setup and interact with MongoDB
3. Plotly and Dash – graphs and web app
4. Testing the web application visualization with live data



TESTING SETUP



5 LITRE CARBOY WITH WATER BY THE WINDOW

FETCHING THE DATA

```
1  void main(void)
2  {
3    /* Variables, external interrupts, UART init etc */
4    for(;;)
5    {
6      /* Polling for input */
7      if (uart_read_count() > 0)
8      {
9        data = uart_read();
10       if (data == 'R') {
11         ow_reset();
12         ow_temp_rd(buffer);
13         uart_send_arr(buffer, len);
14         uart_send_byte('\r');
15         uart_send_byte('\n');
16       }
17     }
18   }
19 }
```

Listing 1: MCU program loop

DAEMONIZING DATA COLLECTION

```
[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 2: The systemd service that manages data collection

ACCESSING THE JOURNAL

```
~ % journalctl --user-unit=serial-temp-to-db.service
...
    -- Boot 492caa4c348b44e599a8070b1f5740d6 --
Feb 21 11:49:12 alarm systemd[361]: Starting Reads serially from /dev/ttyUSB
     * and puts in MongoDB...
Feb 21 11:49:59 alarm python[368]: Running tests on port: /dev/ttyUSB0
Feb 21 11:49:59 alarm python[368]: Baud rate = 9600
Feb 21 11:50:00 alarm python[368]: Faulty reading: 85.0000
Feb 21 11:50:00 alarm python[368]: Faulty reading: 85.0000
Feb 21 11:50:01 alarm python[368]: Correct reading: 19.1875
Feb 21 11:50:01 alarm python[368]: Correct reading: 19.1875
Feb 21 11:50:02 alarm python[368]: Correct reading: 19.1875
Feb 21 11:50:02 alarm python[368]: Passed init tests on 3 correct readings
      and 2 faulty readings
Feb 21 11:50:02 alarm systemd[361]: Started Reads serially from /dev/ttyUSB*
       and puts in MongoDB.
```

Listing 3: Init tests logged in the journal

SERIAL DATA TO DB

```
client = MongoClient('localhost', IP_ADR) # MongoDB
...
ser = serial.Serial(PORT, BAUD)           # Open serial conn to MCU
...
def get_temperature(temp):
    ser.write(bytes(READ_CMD))            # Write read command
    line = ser.readline().decode()        # Read returning data
    if no errors and < MAX_FERM_TEMP:     # Error checking
        temp = (float(line.strip('\x00\n\r')))
    return temp

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

Listing 4: The Python script running from systemd

DB TO GRAPH

```
1  client = MongoClient('localhost', IP_ADR) # MongoDB
2  db = client.beertemp
3
4  ...
5
6  df = pd.DataFrame(list(db.entries.find().limit(int(LIVE_RES)).sort([('
       $natural',-1)]))) # Data in span from past to present
7      trace = go.Scatter( # Scatter plot
8          x=df['time'],
9          y=df['temperature'])
10 ...
11
12 return fig
```

Listing 5: Getting data from DB and plotting using Plotly

LIVE DEMO

CONCLUSION

1. IT WORKS!
2. Timing issues and trash input solved
3. I had to read a lot in order to decide *what* to choose
4. The workflow is very streamlined due to (magic) Python module integration

FURTHER WORK AND THE FINAL PHASE

1. Research different methods of deploying the web app
2. Design the application layout. More user interaction
3. Working with the higher grade specifications
4. Write the final report