Pump Gradual Adjustment Room Temperature for Thermia PGART_T

A small system that will try to reduce the electric energy consumption for a Thermia Atlas heat pump by adjusting the desired indoor temperature.

2023 PG Andersson

See the configuration file `pgart_control_heating.conf_en` for verbose explanations.

The programs are written in Python. Tested in Oracle VM with Debian and in a Raspberry Pi.

Prerequisites

The system loads `hourly_rate_yyyymmdd.txt` by running one of the programs shown in the configuration file. Some of them scrape day-ahead prices from webpages. If the used webpages change functionality, the scraping programs must be updated. The system can also run external programs to create the file.

By default the system scrapes windchill forecasts from SMHI. Seems to work in the Nordic and the Baltic states. It can also run an external program to create the forecasts file.

TCP-Modbus will likely never change but the API to online-genesis might change when a bug or new security methods are needed.

Security alert

Once TCP is configured on the pump it is wide open for access on the LAN.
If you access a pump over internet you must thus use online-genesis unless you have an application FW that secures the pump.

Installation

Untar/unzip `pgart.tar.gz` in `your-user/`.

```
pgart
├── bin
│   ├── pgart_calc_hourly_rate_adj_func.py
│   ├── pgart_control_heating.py
│   ├── pgart_env_func.py
│   ├── pgart_experimental_calc_costs.py
│   ├── pgart_experimental_conv_eon_files_to_generic_func.py
│   ├── pgart_get_hourly_rates_elprisetjustnu_se.py
│   ├── pgart_get_hourly_rates_entsoe_eu.py
│   ├── pgart_get_hourly_rates_herrforsnat_fi.py
│   ├── pgart_get_hourly_rates_minspotpris_no.py
│   ├── pgart_get_indoor_temp_raspberry_pi.py
│   ├── pgart_get_monthly_rates_elbruk_se.py
│   ├── pgart_get_smhi_forecasts_func.py
│   ├── pgart_get_smhi_forecasts.py
│   ├── pgart_lang_func.py
│   ├── pgart_misc_func.py
│   ├── pgart_read_control_params_func.py
│   ├── pgart_thermia_modbus_func.py
│   └── pgart_thermia_online_genesis_func.py
├── etc
│   ├── pgart_control_heating.conf_en
│   ├── pgart_control_heating.conf_se
│   ├── pgart_language.conf
│   └── pgart_mail_params.conf
├── var
│   ├── local
│   │   └── explanation_summary_run_log.txt
│   ├── log
│   └── stat
└── xtra_if_systemd
    ├── help_install_systemd.txt
    ├── pgart_control_heating.service
    └── pgart_control_heating.timer
```

Production

If you have a gmail account with an "app password" messages can be sent to it.

The following three configuration files should be initially updated :

- `pgart_language.conf.`
- `pgart_control_heating.conf_se/_en.`
- `pgart_mail_params.conf.`

The program must run once an hour otherwise the logic fails. Via crontab or systemd.

`Crontab:`
Let the program run 1-2 minutes past the hour to minimize the risk of request timeouts.
Add the next line:

```
2 * * * /usr/bin/python3 /home/your-user/pgart/pgart_control_heating.py >>
/home/your-user/pgart/var/log/hourly_run.log
```

`systemd:`
See `xtra_if_systemd` for instruktions.

An experimental part of this system creates a table with the costs per hour based on hourly rates, monthly rates and the el-consumption. The system reads consumption data from files that have to be be created externally. When EON is the provider you can use a program here (but it requires some manual work.) See more under “Experimental Data Files”.

This command creates a file with a cost compilation.

```
Python3 pgart_experimental_calc_costs.py --fromymd <yyyymmdd> --
toymd <yyyymmdd>
```

Ordinary Data Files

`your-user/pgart/var/local/hourly_rate_yyyymmdd.txt`
The hourly-rate processing needs a file with the day-ahead prices. The system runs the program named in the parameter “`pgm_create_hourly_rates`” to create such a file but it can also run a user provided program.

Extracts from a standardised hourly-rate file.
Hour,price (in “öre, cent with two decimals”)/kWh

```
00: 305.16
06: 255.30
17: 360.55
23: 202.71
```

`your-user/pgart/var/smhi_forecast.txt`
A json-file with verbose forecast for the next 24 hours. Scraped from SMHI. An intermediate file which is only used when the system creates the `forecast_short.txt` file.

`your-user/pgart/var/forecast_short.txt`

A standardised file with forecasts. It is required for the windchill processing. The system creates it by extracting data from `smhi_forecast.txt.` The system can also run a user provided program to create the file

Layout: One line per hour:
day_hour : forecast-temperature, forecast-wind-m/s

```
2023-01-17_20: 0.8, 6.8
```

`your-user/pgart/var/windchill_calculations.txt`
The program will create this file with extended windchill information for the records in `forecast_short.txt.`

Layout: One line per hour: (Here spaced for better visibility.)
Day_hour : temperature wind m/s : calculated windchill diff real- and windchill temp

```
2023-01-17_20 : 0.8    6.8  : -4.8          -5.6
```

`your-user/pgart/var/log/windchill_stats.log`
This file keeps the values used in the calculation of the indoor temperature according to the windchill effect.
A long line. Breaken up for visibility.

```
2023-02-05_14:02
h: 14  last_indoor_t: 20
forec_t: 1.1  forec_wind: 5.8
factor: 0.8
windchill_t: -4.0  diff:- 5.1
wanted_inc: 2.0  got_inc: 2.0 (max 3)
```
The temperature increase is acceptable.

`your-user/pgart/var/log/action.log`
The system logs some processing steps here.

`your-user/pgart/var/log/summary_run.log`
One line per run. (If the pump is without a room sensor the temperature value will be 200!)

The header in the real logfile has explanations.
```
2023-03-13_14:02 000010 No_schema    out:7   room:200 sensor:20.3
pump:22 new:22 hr_rate:0 off      windchill:1 set_hour

2023-03-13_15:02 001000 No_schema    out:8   room:200 sensor:20.6
pump:22 new:21 hr_rate:0 off      windchill:1 reset_hour

2023-03-13_16:02 000001 Set_manually out:8   room:200 sensor:20.2
pump:20 new:20 hr_rate:0 off      windchill:0 off
```

`your-user/pgart/var/settings_status.txt`
The system stores internal runtime data here.

Experimental Data Files

The program that creates the table with the rates for the el-consumption requires that files with the hourly-el-consumption are stored on the system as `hourly_comsumption_yyyymmdd.txt.` See below for file layout.

If you use EON you can manually download the csv-files with the hourly el-comsumption.
EON has not any free of charge API.

`your-user/pgart/var/local/export-2022.csv`
A file downloaded from EON with your hourly el comsumption. The files will get names like export-2023.csv, export-2023 (1).csv There is not any timestamp in a downloaded file.

An extract:
Förbrukning per timme för 2022
"Klockslag";"Förbrukning för 2022(kwh)"
```
"00:00";2.127
"01:00";1.248
```

You must rename the downloaded files to `eon_yyyymmdd.csv` so the system knows the dates
`your-user/pgart/var/local/eon_yyyymmdd.csv`

`your-user/pgart/var/local/hourly_comsumption_yyyymmdd.txt`
A standardised file with the hourly el comsumption. The system converts EON's csv-file to this format.

An extract:
Hour consumption (kwh)
```
00: 2.127
06: 4.933
17: 2.436
23: 0.902
```

`your-user/pgart/var/local/monthly_rate.txt`
A file with the monthly rates is also required for the function to work. The system scrapes those from elbruk.se.

An extract:
year-month : price (öre/kWh)
```
2023-02: 52.03
2023-01: 70.38
```

`your-user/pgart/var/stat/el_charges_fromyyyymmdd_toyyyymmdd.txt.`
The file with the cost compilation. It will be created by `pgart_calc_costs.py`

File layout (In Swedish because this will probably only work in Sweden.):

2022-12-17
```
Kl    kWh     tim   medel  timpris medelpris    diff
             pris    pris  kostnad   kostnad
00  2.127    3.05    2.71     6.49      5.77    0.72
```

```
06  4.933    2.55    2.71    12.59      13.39   -0.80
17  2.436    3.61    2.71     8.78       6.61    2.17
23  0.902    2.03    2.71     1.83       2.45   -0.62
```

Dygnets förbrukning: 57.28 kWh
Kostnad för hela dygnet med timpriset: 166.10 kr, med månadspriset: 155.48 kr, skillnad: -6.40%

Pump help

To activate Modbus TCP:

1. Operation mode off.
2. Authorisation code 607080.
3. Activate BMS.
4. TCP, keep port 502.

Resource usage

The control program will run once an hour. A new temperature will only be sent to the pump if it differs from the current temperature.