DANISH
TECHNOLOGICAL
INSTITUTE

Focus on the control and simulation setup

Appendix B:

Description of the OPSYS test rig software

Title:
Appendix B: Description of the OPSYS test rig software

Prepared for:
EUDP

Prepared by:

Danish Technological Institute
Teknologiparken
Kongsvang Allé 29
8000 Aarhus C
Refrigeration and Heat Pump Technology

October 2016
Author: Torben Green

And

Aalborg University
Section of Automation and Control
Fredrik Bajersvej 7C
9000 Aalborg

February 2017
Author: Kasper Vinther

Table of Contents

1. Introduction

The purpose of this document is to disseminate knowledge regarding the test rig used in the project "*P2003908_Optimering af gulvvarme og vp systemer*", hereafter referred to as OPSYS.

The document is structure in the following way:

- Section 2 gives an in-depth description of the control and simulation setup, including a description of the communication set along with a description of the structure of the Python source code.
- Section 3 presents the proposed test procedures to be followed when working with the test rig.
- Section 4 presents a list of known issues.
- Section 5 presents a short manual for the data processing scripts
- Appendix 1 contains a quick guide on running the OPSYS test rig

2. Control and communication setup

This section provides a description of the control and communication setup for the test rig. Concerning control, communication and data acquisition, the test setup comprises of four different entities.

1. The Simulation pc
2. The SIP
3. The Trend controllers
4. The Agilent logger

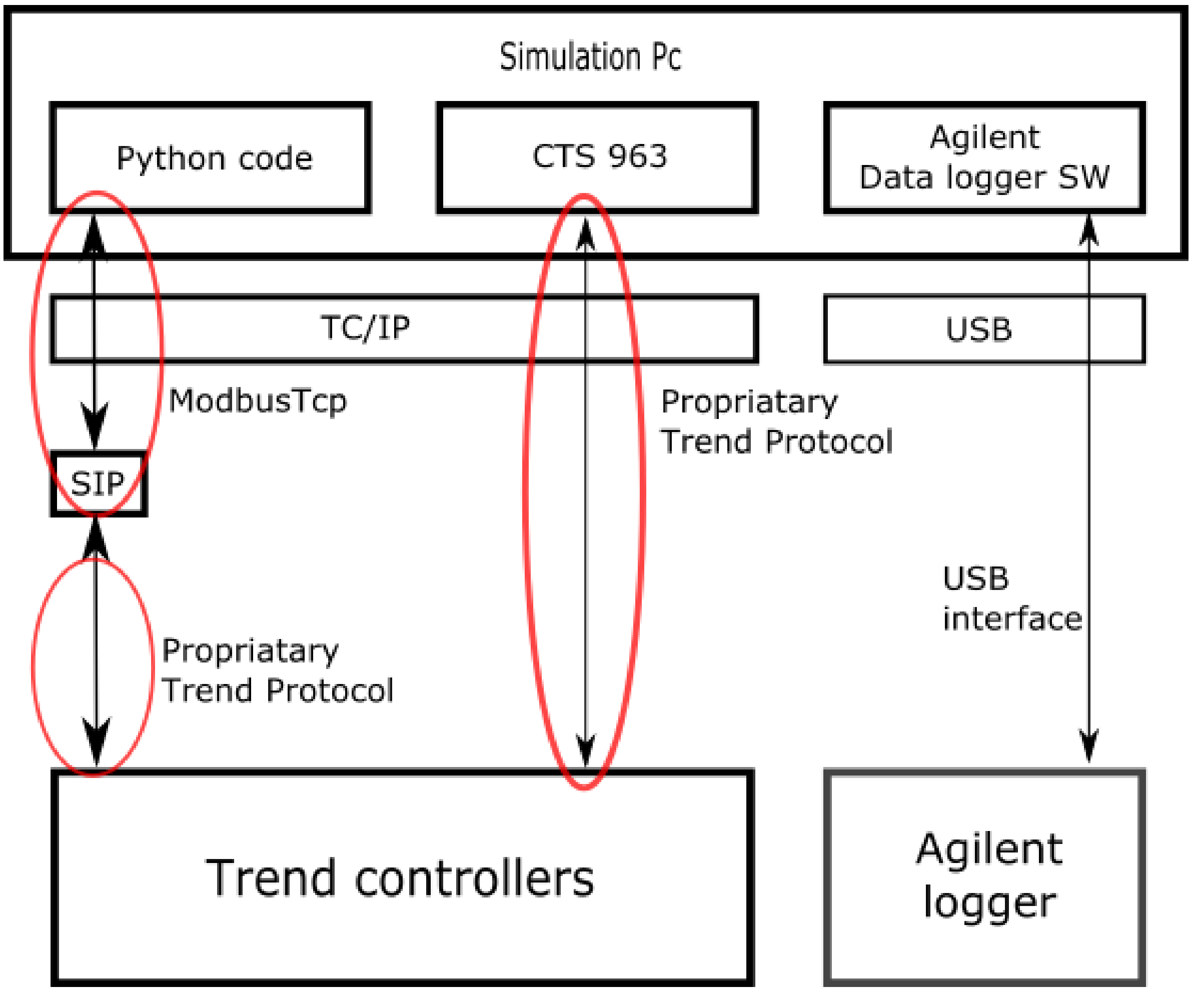


Figure 1: Communication and data acquisition setup

The four entities are all depicted in Figure 1, which illustrates the communication and data acquisition setup.

2.1. The simulation pc

Figure 1 shows that the **Simulation pc** is running three different tasks, which are:

1. Executing the python code which handles the following tasks:
    a. Running the simulation of the house
    b. Running the high-level control algorithms
    c. Running the Modbus server to enable communication with the Trend controllers through the SIP Modbus interface box
    d. Data logging in a csv-file where the file name is based on the data and time when the Python script pymodbus_server.py was started. File name structure: "simRlog<YYYYmmDD>_<HHMM>_ID<#>.csv"
2. Running the CTS 963 software, which handles the following tasks
    a. Providing GUI for the Trend controllers and thereby an interface to the test rig
    b. Communicating with Trend controllers through proprietary protocol over TCP/IP
    c. Data logging the Trend controllers in an internally used database
    d. Data logging each measurement point in a separate csv file. Files are saved in "C:\Log data". Files are only updated when CTS963 is running.
3. Agilent Data logger SW, which are handling the following tasks
    a. Logging of the Agilent data logger through the USB interface
    b. Providing online plots of the logged signals
    c. Exporting logged data into a csv file where the name is based on the date and time when the log was started. Locating of the data file is "C:\Log data Agilent". File name structure:
    "opsysAgilent <mm>_<DD>_<YYYY> <HH>_<MM>_<SS> <#>.csv"

2.1.1. Data logging and data sources

Because the Simulation pc is running three different and separate tasks, it is also generating three different data sets from each test. None of the three data sets have the same structure, and they, therefore, have to be processed in different ways.

The Python scripts utf16ToUtf8.py and csvTst.py can be used to process the data. The script Utf16ToUtf8.py converts the data files from the CTS 963 software and the data file from the Agilent logger from UTF16 to UTF8 encoding. The script csvTst.py provides different export and plotting features for the data. Both scripts provide the user with a command line interface, e.g. by typing the following in the anaconda prompt:"python ut16ToUtf8.py <path to folder with data>". In addition, both scripts have an optional help option that can be accessed in the following way: "python csvTst.py –h".

2.1.2. Data of interest

Because the test run creates three different data sets and therefore covers three different, but overlapping, time periods, it has been decided that the simulation time defines the data of interest. Hence, the time period covered in the data file from the python code i.e. "simRlog<YYYYmmDD>_<HHMM>_ID<#>.csv" should represent a sub-set of the time period covered in the data files written by the CTS 963 software and the Agilent data logger software. This enables slicing of the time series in the CTS 963 data files and the Agilent data file based on the start and end time in the data file from the simulation.

2.1.3. Python files

The python code is split up into five files:

1. **pymodbus_server.py**
    a. Main file used when making tests on the test rig hardware. To start all the python codes see Section 3.
    b. Uses class/function descriptions located in the other python files.
    c. The following configuration parameters can be changed in this file:
        i. SERVER_IP = '172.17.32.91' #ip address of the host computer
        ii. SERVER_MODBUS_PORT = 502
        iii. FMU_DIR_RELPATH = ' '
        iv. FMU_NAME = 'OpSys_0Experiments_Test_0setup_0models_BR1970_0house_0simulation_0with_0data.fmu'
        v. SIM_START = 0.0
        vi. SIM_END = 60*60*24*31 #continue sim for 31 days
        vii. SIM_STEP = 15.0 #seconds between sim iterations
        viii. CTRL_STEP = 600.0 #seconds between ctrl iterations
    d. The file makes sure that a simulation_runner thread, which propagates the house model every SIM_STEP seconds and records data in a csv file, is started.
    e. The file also makes sure that a ctrl_alg thread, which runs the control algorithms that needs to be tested, is started.
2. **datastore.py**
    a. Python modbus data context used to share data between the SIP, the simulation thread, and the control thread.
    b. Read/write operations are blocking to make it interrupt/thread safe.
3. **simulation_runner.py**
    a. Reads data from the shared data store context.
    b. Generates input vectors for the simulation model and propagates it SIM_STEP seconds.
    c. Stores results in csv file and sends room temperatures and water return temperatures to the Trend controllers through the SIP.
    d. The following configuration parameters can be changed in this file:
        i. self._model.set("filter_cutoff_hz",1) #Fast filter that makes sure that input data is smooth and continuous for more robust simulation.
        ii. self._model.set("T_forward_off",273.15) #convert T_forward to Kelvin
        iii. self._model.set("start_time_sim",-24*60*60*30) #start sim 30. January
    e. The fmu file, e.g., named **"OpSys_0Experiments_Test_0setup_0models_BR1970_0house_0simulation_0with_0data.fmu"** contain the model description for a 1970's house which uses the weather and load data profiles for a year located in **"all_data_1970.txt"** or

"all_data_noisy_1970.txt" (values are added +/- 10% normally distributed random noise).

   f. Note that the fmu should be generated as an fmu 2.0 using co-simulation with Dymola solvers in order to function properly (standard CVODE solver sometimes crashes and should be avoided – Dymola Radau-II solver is better).

4. **fileTools.py**
   a. Contains a function that can generate a unique filename for the logged data.
5. **ctrl_alg.py**
   a. Runs control algorithms, e.g., room temperature on/off valve control.
   b. Modify the function "run_ctrl()" inside the class CtrlAlg() with your own control code.
   c. If a neural network is in use then the file **"weather_jan_to_mar.mat"** contains predictions/forecasts of the weather in the first three months of the year, and the file **"NN_pars_nX_kY.mat"** contains the pre-trained neural network parameters (X represents the number of neurons and Y represents the training run that provided the best performing network).
   d. Currently, this file also contains help functions for running neural network prediction and genetic algorithm optimization.

In order to be able to run the python code on another computer, the following installations must be made:

1. Anaconda 4.3.0 or newer (python 2.7, 32 bit).
2. Pyfmi 2.3.1 (just run the MS windows installer for python 2.7 from https://pypi.python.org/pypi/PyFMI)
3. pymodbus version 1.3.0 from github is required (not pypi standard version).
4. If a neural network is in use then open an anaconda prompt and type "pip install neurolab".

2.2. The SIP

The task of the SIP is to create a Modbus interface between the Python code running in the Simulation pc and the Trend controllers. The SIP acts as a ModbusTcp client, and it ensures that data signals from the Trend controllers are written into the data store maintained by the Modbus server running in Python on the Simulation pc. In addition, the SIP also ensures that data is transferred from the datastore of the Modbus server to the Trend controllers. The mapping between the Modbus addresses and the Trend controller input and output is configured using the web based configuration tool on the SIP.

- IP: http://172.17.32.93/
- User name: Admin
- Password:password

(The back up file "SIP_BackupSImPCAsServer" of the working configuration can be found on the DTI server on: Y:\Projects\P2003908_Optimering af gulvvarme og vp systemer\Fagligt\Sip\.)

2.3. The Trend controllers

The task of the Trend controllers is to provide the low-level controls of all the electronic valves on the test rig along with the logging of all the relevant measurements and set point for the various control loops. In addition, the Trend controllers provides an interface to the heat pump by presenting five inputs and five outputs that can be used for future improvements of the test rig setup. One of the outputs from the Trend controllers is currently being used to set the ambient temperature for the heat pump and thereby indirectly controlling the forward temperature of the heat pump.

2.4. The Agilent logger

The task of the Agilent data logger is to log the sensors and signals connected to the Agilent logger that are not logged through the CTS 963 or the Python code. Hence, the Agilent logger is only used for logging signals and is, therefore, not necessary in connection with the running of a test. In the current setup, it is, therefore, not possible to include sensors or signals connected to the Agilent logger in the control setup of the test rig. Signals and sensors that should be used for control by the Trend controllers or the high-level control in the Python should be connected through the Trend controllers.

3. Test procedure

The section provides a description of a test procedure that ensures that data from all three data sources can be used and will cover the time period of the data of interest as discussed in section 2.1.1. A start up procedure and a post-test procedure will be described.

3.1. Start-up procedure

The following procedure must be followed to ensure that the data generated by the Trend controller system during testing is stored correctly. The Trend controller stores 1000 samples from each sensor, and the CTS 963 will, therefore, have to collect data from each of the controllers before they reach the limit of 1000 samples. The collection of sensor data from the Trend controller is automated in the CTS 963 software. Before each test it is recommended to do a clean-up of both the directory, where CTS 963 stores csv files and the internal database of the CTS963.

3.1.1. Pre-test procedure

1. Ensure that the directory for the csv-files is empty. The directory is: 'C:\Log data'. Files should be moved to another directory or deleted.
2. Start CTS 963
   - Data will be logged and saved in both the database and the csv-files as soon as CTS 963 is started.
3. Empty database using the “Tøm database” (Empty database) button in the CTS 963 interface, and deleted newly created files in the csv directory 'C:\Log data'.
4. **Optional step:** Change relevant settings in the CTS 963, e.g. toggle relevant switches to ensure that input from SimPc is used as input to the Trend controllers.

5. Run system until system is ready for the actual test
    - To ensure proper operation of the test rig, it is necessary to start the simulation by running the python script "pymodbus_server.py. This ensures that the test rig receives relevant feedback and can operate correctly.
6. Run test
    - Stop the Python script pymodbus_server.py.
    - Redo step 1 and 3.
    - Start Agilent logger.
    - Start the Python script pymodbus_server.py.
7. **IMPORTANT DO NOT** stop CTS 963 immediately after test. Wait at least 20 minutes to ensure that all relevant data is stored in both the database and the csv files.

3.2. Post-test procedure

The following steps should be followed when a test run is over and data are to be recovered. The order of the steps is important to ensure that data from all of the three data sources can be used together.

1. Stop the Python script pymodbus_server.py.
2. Create a directory to store all data from the test.
3. Copy the simRlog[data_time_ID#] csv to the directory created in step 2.
4. Stop Agilent logger.
5. Copy Agilent data file to directory created in step 2.
6. Wait at least 20 minutes after the test has finalized before continuing with this procedure.
    a. Waiting 20 minutes ensures that all test data is collected from the Trend controllers, stored in the SQL database, and written to the csv-files. Hence, if this procedure is continued prematurely, test data might get corrupted.
7. Move all the csv-files from 'C:\Log data' to the directory created in step 2 for storage.
    a. Store files together with data files from the Python script pymodbus_server.py, i.e. the simRlog[data_time_ID#] csv and the data files created by the Agilent data logger. This ensures that the data processing scripts can be used to process the data.
8. **Optional step:** Empty the internal CTS 963 database.
9. Repeat test or close the system.

4. List of known issues

This section presents a list of known technical issues concerning the test rig and the communication and control setup.

1. CTS 963 alarms.
    a. The CTS 963 creates a large number of alarms during operation. Some of these indicate that data is/might be missing. Some few short periods – less than half an hour – are missing for some of the 963 values. The reason for this has not yet be found, but is dealt with in a small Fortran program.

2. CTS 963 data files contain redundant data.
    a. The data files are updated by appending the entire buffer for a given measurement point from a Trend controller to the corresponding csv file. Hence, there is a significant redundancy of data in the files. Cleaned files can be created by using the following command: ”python csvTst.py export <path to folder>
3. CTS 963 database limit.
    a. The database used internally by the CTS 963 software has a limit of 10GB and it is, therefore, recommended that the database should be emptied using the "Tøm database" (Empty database) button in the CTS 963 interface.
4. Data blow up when emptying the database.
    a. When emptying the database file, *.mdf, the log file, *.ldf files blows up to approximately 10-12 times the size of the database file. Hence, the *.mdf files should not be allowed to grow above 8 % of the remaining capacity on the hard drive.

5. Data processing scripts

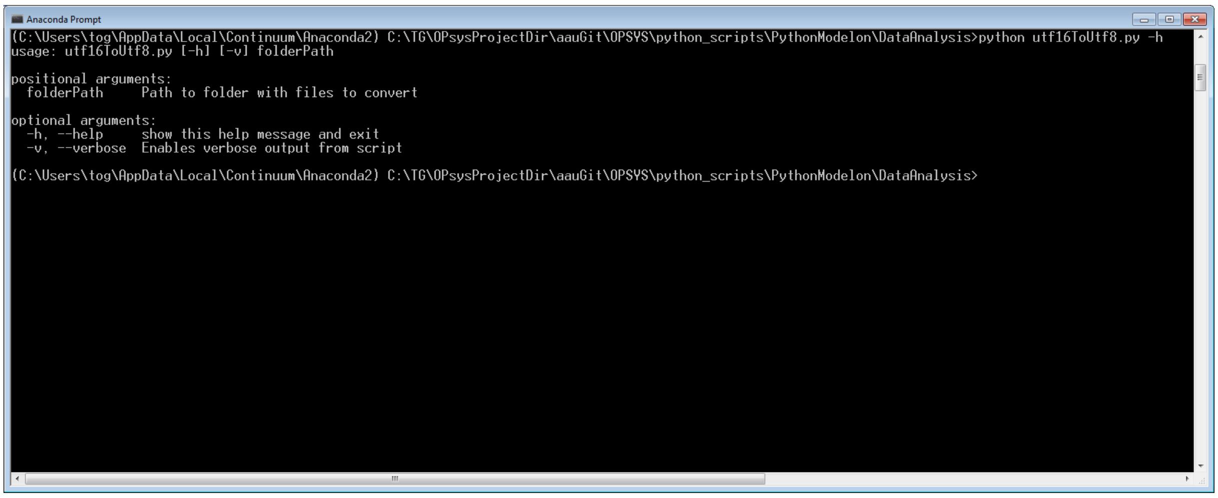
This section provides a short manual for the data processing scripts. There are two main scripts that should be used for processing the data, and they are called utf16ToUtf8.py and csvTst.py. The use of the scripts will be described using a description of a suggested workflow.

The suggested workflow is as follows:

1. Convert to files from utf16 to uft8 encoding
    a. python utf16ToUtf8.py <folder Path>
2. Inspect the data set by looking at which signals exists in the given data set.
    a. python csvTst.py folder –p <folder Path>
3. Export the CTS 963 files
    a. python csvTst.py export <folder Path>
4. Choose which signals to plot
    a. Edit the list of signals to plot in csvTst.py. The list can be found in the beginning of the function called "combfolder_main"
5. Plot the signals
    a. python csvTst.py folder <folder Path>
    b. Optional arguments to consider:
        i. ”–v” for a more verbose output
        ii. "-R" for real-time time stamps. Default is simulation time based time stamps
        iii. "-n" for no show of the plots. Pdf files of the plot are still saved.

5.1. Optional arguments

Both of the python scripts have optional arguments which always start with a dash”-“. The most important optional arguments is the help argument, which is invoked in the following way:

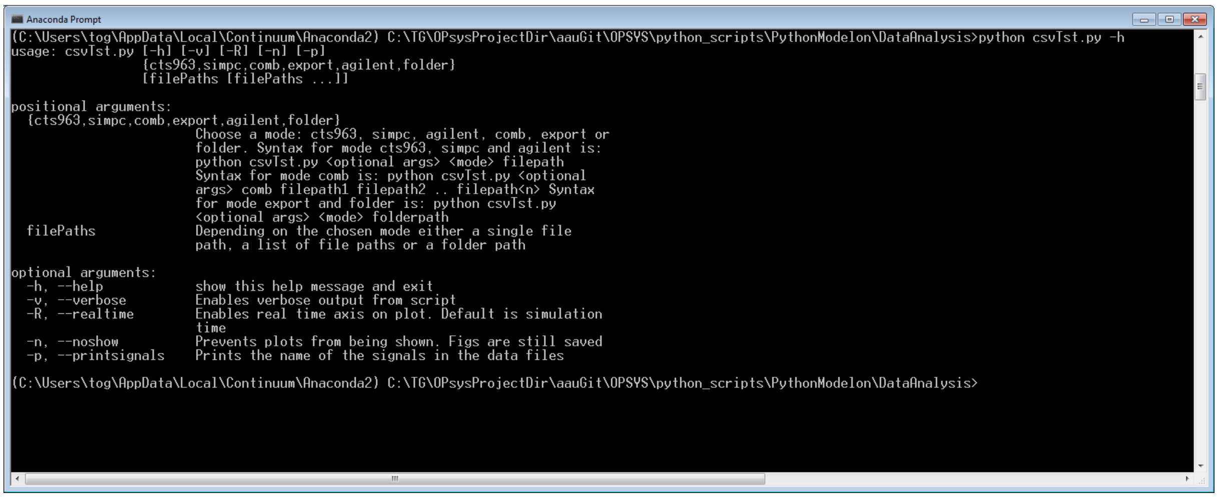


```
Anaconda Prompt
(C:\Users\tog\AppData\Local\Continuum\Anaconda2) C:\TG\OPsysProjectDir\aauGit\OPSYS\python_scripts\PythonModelon\DataAnalysis>python utf16ToUtf8.py -h
usage: utf16ToUtf8.py [-h] [-v] folderPath

positional arguments:
  folderPath     Path to folder with files to convert

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  Enables verbose output from script

(C:\Users\tog\AppData\Local\Continuum\Anaconda2) C:\TG\OPsysProjectDir\aauGit\OPSYS\python_scripts\PythonModelon\DataAnalysis>
```



```
Anaconda Prompt
(C:\Users\tog\AppData\Local\Continuum\Anaconda2) C:\TG\OPsysProjectDir\aauGit\OPSYS\python_scripts\PythonModelon\DataAnalysis>python csvTst.py -h
usage: csvTst.py [-h] [-v] [-R] [-n] [-p]
                 {cts963,simpc,comb,export,agilent,folder}
                 [filePaths [filePaths ...]]

positional arguments:
  {cts963,simpc,comb,export,agilent,folder}
                        Choose a mode: cts963, simpc, agilent, comb, export or
                        folder. Syntax for mode cts963, simpc and agilent is:
                        python csvTst.py <optional args> <mode> filepath
                        Syntax for mode comb is: python csvTst.py <optional
                        args> comb filepath1 filepath2 .. filepath<n> Syntax
                        for mode export and folder is: python csvTst.py
                        <optional args> <mode> folderpath
  filePaths             Depending on the chosen mode either a single file
                        path, a list of file paths or a folder path

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Enables verbose output from script
  -R, --realtime        Enables real time axis on plot. Default is simulation
                        time
  -n, --noshow          Prevents plots from being shown. Figs are still saved
  -p, --printsignals    Prints the name of the signals in the data files

(C:\Users\tog\AppData\Local\Continuum\Anaconda2) C:\TG\OPsysProjectDir\aauGit\OPSYS\python_scripts\PythonModelon\DataAnalysis>
```

The “-h” or “—help” option provides the user with a help output that describes both the usages of the script and the positional arguments as well as the optional arguments.

Appendix 1

Quick guide on how to operate the OPSYS test rig

The following is a small guide describing how to stop a running test on the OPSYS test rig and how to start a new test:

%%
In order to stop the test:
%%
1) stop any python scripts currently running (press Ctrl+c)
2) stop the agilent on the square black Start/Stop button
3) wait 20 minutes (so that we are sure that all CTS data has been logged)
4) cut the newest:
  - simRlog csv file from ..\python_scripts\testrig_with_opt_ctrl,
  - agilent csv file from C:\Log data Agilent,
  - and all csv files from C:\Log data.

Put the files into a folder on the external drive or on your own computer (to make sure we don’t use all the hard disk space on the C drive).

5) Open the CTS963 program and navigate to the "Oversigt Anlæg" menu (vertical menu on the left side of the screen).
  Press the "Tøm database" button.
  Open the folder path C:\Program Files (x86)\Trend Control Systems\963\database and wait until the file i96X_log.ldf is about 1 mb large again (it will grow for a while and then jump back to 1 mb after it has reached a size of about 11 times the size of i96X_data.mdf).
  Note that there might be cases when the size of i96X_log.ldf reaches certain value and in is not 1mb if so then press "Tøm database" button again.
%%
In order to start a test:
%%

1) First apply your changes to the python scripts.
2) Push the triangular green start button in the agilent program.
3) wait a couple of minutes to get all logs up an running.
4) From the windows menu start open the Anaconda Prompt.
5) Using 'cd path_of_the_file' command navigate to ..\OPSYS\python_scripts\testrig_with_opt_ctrl.
We do not give specific folder name because it might change together with the user, if in doubt where the file is search it using windows.
6) run the python script pymodbus_server.py by typing "python pymodbus_server.py".

Plot of the test data in matlab:
1) Open a windows command prompt and navigate to OPSYS git respository ..\OPSYS\python_scripts\DataAnalysis.
2) Type "python utf16ToUtf8.py your_path_to_log_data".
3) Type "python csvTst.py export your_path_to_log_data".
4) Run the matlab script "plot_data" found in ..\OPSYS\matlab_scripts\data_analysis.