

Vesa-Matti Yli-Heikkilä

Home surveillance with Raspberry Pi

Bachelor thesis

Spring 2015

School of Technology

Automation Engineering



SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: SeAMK Tekniikka

Tutkinto-ohjelma: Automaatiotekniikka

Suuntautumisvaihtoehto: Sähköautomaatio

Tekijä: Vesa-Matti Yli-Heikkilä

Työn nimi: Kodin valvonta Raspberry Pi:llä

Ohjaaja: Alpo Anttonen

Vuosi: 2015

Sivumäärä: 50

Liitteiden lukumäärä: 7

Opinnäytetyössä tutkittiin Raspberry Pi:tä ja sen mahdollisuuksia toimia kotitalouden tiedonkeruu- ja prosessointiyksikkönä. Raspberry Pi suorittaa kodin valvontaa keräämällä siihen liitetyiltä 1-Wire DS18S20-antureilta lämpötilatietoja SQLite-tietokantaan.

Kodin valvontaan sisältyvät myös kuvakaappaukset Pi NoIR-kameramoduulilla. Kerätty tietokanta on esitetty havainnollisessa ja helposti ymmärrettävässä kaaviomuodossa, jota ylläpidetään Apache2 HTTP-palvelinohjelmistolla.

Avainsanat: Raspberry Pi, 1-Wire, tiedonkeruu, SQLite, Apache2.

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Automation Engineering

Specialisation: Electric Automation

Author: Vesa-Matti Yli-Heikkilä

Title of thesis: Home Surveillance with Raspberry Pi

Supervisor: Alpo Anttonen

Year: 2015

Number of pages: 50

Number of appendices: 7

This bachelor's thesis is a research on Raspberry Pi and its possibilities to work as a data processing and logging unit in a household use. Raspberry Pi performs home surveillance and gathers temperature information data from the attached 1-Wire DS18S20 sensors to an SQLite3 database.

The picture capturing with the Pi NoIR camera module is also a part of the home surveillance. The gathered temperature data is presented in an easily readable line chart form which is hosted by the Apache2 HTTP server.

Keywords: Raspberry Pi, 1-Wire, data logging, SQLite, Apache2.

CONTENTS

Opinnäytetyön tiivistelmä	2
Thesis abstract.....	3
CONTENTS.....	4
TABLES AND FIGURES.....	6
ABBREVIATIONS.....	7
1 INTRODUCTION.....	8
2 THEORETICAL OVERVIEW	9
2.1 Linux	9
2.1.1 History of Linux	9
2.1.2 Linux compared to Windows.....	10
2.1.3 Raspbian Wheezy	10
2.2 Introduction to Raspberry Pi.....	11
2.2.1 Available versions of Raspberry Pi.....	12
2.2.2 Programming languages	13
2.2.3 General Purpose Input Output (GPIO)	13
2.2.4 Raspberry Pi NoIR Camera module	16
2.3 1-Wire technology and DS18S20 temperature sensors	17
2.4 Python programming language	18
2.5 SQLite3.....	19
2.6 Apache2.....	19
3 IMPLEMENTATION OF A HOME SURVEILLANCE SYSTEM	20
3.1 Setting up Raspberry Pi.....	20
3.2 Creating the 1-Wire network	24
3.3 Measuring the temperatures and creating the database	26
3.3.1 Measuring the temperatures	27
3.3.2 Setting up the database for measured data	28
3.3.3 Creating a Python script to measure and store the data	30
3.4 Installing the Apache2 HTTP server	33
3.5 Controlling the GPIO pins with Python	35
3.5.1 Controlling the LED with the GPIO	35

3.5.2 Controlling the heating element	37
3.6 Taking advantage of Raspberry Pi's camera module	38
3.6.1 Installing the Pi NoIR camera module	38
3.6.2 Taking the first pictures and videos with the Pi NoIR camera	39
3.6.3 Creating a Python script for taking pictures.....	40
3.7 Creating scheduled tasks with Cron.....	41
3.8 Creating a website interface.....	42
3.9 Optimization	46
4 SUMMARY.....	47
REFERENCES.....	48
APPENDICES	50

TABLES AND FIGURES

Figure 1. Raspberry Pi model B+	11
Figure 2. Pi NoIR camera module.....	16
Figure 3. Raspberry Pi in protective enclosure	20
Figure 4. Raspi-config	21
Figure 5. The keyboard configuration, setting the layout to the Finnish “fi”	23
Figure 6. Testing the DS18S20 temperature sensors through the GPIO	25
Figure 7. External power supply wiring diagram.....	26
Figure 8. Checking the DS18S20 temperature sensors serial numbers	27
Figure 9. Reading the temperature	28
Figure 10. Finished database structure	29
Figure 11. LED wiring diagram	36
Figure 12. Installing the Pi NoIR camera module	39
Figure 13. Line chart code generated with criteria.....	44
Figure 14. Activity diagram of the website	45
Table 1. Different versions of Raspberry Pi (Element14 [Ref. 22.2.2015].).....	12
Table 2. GPIO connector's pin layout on Raspberry Pi Model B+	14
Table 3. List of special purpose GPIO pins.....	15
Table 4. Technical specifications of the DS18S20 sensor.....	18
Table 5. Example of the crontab command (The Geek Stuff [Ref. 16.3.2015].) ...	42

ABBREVIATIONS

RasPi	Raspberry Pi, a single chip computer.
Kernel	Kernel is the heart of the operating system which handles the communication between the user and hardware.
Root	The administrator user account.
USB	Universal Serial Bus.
MicroSD	Micro Secure Digital.
GPIO	General Purpose Input/Output. A serial communication port on Raspberry Pi.
SQL	Structured Query Language.
HTTP	The Hypertext Transfer Protocol.
HTML	Hypertext Markup Language.
CSS	Cascading Style Sheets.
LCD	Liquid-crystal display.
LED	Light Emitting Diode.
IP	Internet Protocol.
URL	Uniform Resource Locator.

1 INTRODUCTION

Small single chip computers are becoming more and more popular because of their low purchasing prices and their potential in the performance sector. That is what makes them interesting devices for many practical purposes. (Monk 2013, Introduction.)

One of these purposes is a data logging. Data logging means collecting information from sensors and saving it for the later analyzing. After the data has been analyzed, it can be used to control a device or it can be displayed for observation purposes.

The purpose of this thesis work is to find out possibilities for using the Raspberry Pi single chip computer in a household surveillance. The thesis's goals are to create and design an automated surveillance system which is data logging the temperature data from DS18S20 1-Wire temperature sensors and capturing the pictures which are taken with the Pi NoIR camera module. The gathered data is supposed to be easily accessible via internet browser for observing purposes. The graphical user interface aims to clear visualization and real time data updating.

The bachelor's thesis consists of two main parts: theoretical and implementation parts. The theoretical part contains short introductions to the devices and the software which are used in this bachelor's thesis. The implementation part of the thesis covers the device and software installations. It also includes configurations for the installed software and the actual Python programs. The programs which are made in this thesis can be found in the appendices section.

2 THEORETICAL OVERVIEW

This chapter introduces the devices and software which are used in this bachelor's thesis. The chapter also contains short introduction to the Linux operating system which is used in this thesis.

2.1 Linux

Linux is a free open source operating system and it belongs to the Unix operating systems. Actually Linux means the kernel itself which is the heart of the operating system and handles the communication between the user and hardware. Normally Linux is used to refer to the whole Linux distribution. (Upton, E. & Halfacree, G. 2012, 28.)

Linux distribution is a collection of software based on the Linux Kernel. It consists of the GNU-project's components and applications. Because Linux is an open source project, anyone can modify and distribute it. That is the reason why there are many variations of Linux distributions. Most popular distributions are Ubuntu, Red Hat Linux, Debian GNU/Linux and SuSe Linux. (Kuutti, W. & Rantala, A. 2007, 2.)

2.1.1 History of Linux

Linux is a Unix compatible operating system where the operating system's kernel has been reprogrammed. Because of the compatibility most of the free applications programmed for Unix are also available for Linux. In 1973 Unix was reprogrammed in C programming language instead of the assembly code. At that point Unix reached its current outfit. Unix supported multiple users and it was also easy to transfer to new digital machines. Later Unix was given to Universities for further development. (Kuutti, W. & Rantala, A. 2007, 5-6.)

Linux got started in the early 1990s when Linus Torvalds got tired of MS-DOS operating system and decided to create a new operating system for the Intel's cheap

x86 processors. At the time there was already available Minix operating system for microcomputers. However, mainly for teaching purposes created Minix was not good enough for Torvalds. (Kuutti, W. & Rantala, A. 2007, 5-6.)

In October 1991 Linus Torvalds released the first unofficial Linux and the first official Linux version was released in March 1994. Nowadays Linus Torvalds is still partly developing and supporting the kernel's further development. (Kuutti, W. & Rantala, A. 2007, 5-6.)

2.1.2 Linux compared to Windows

When comparing Linux and Windows as operating systems, one of the major differences are that Linux is an open-source project and Windows is a closed-source project. In the closed-source project the users sees only the finished product but do not know how it has been done. In open-source projects everything is made fully visible to the public. (Upton, E. & Halfacree, G. 2012, 13-14.)

In practice this can be seen in Linux's easy customization for different platforms. This process is called porting. There are several distributions ported to the Raspberry Pi's BCM2835 chip. One of the distributions is called Raspbian Wheezy. (Upton, E. & Halfacree, G. 2012, 14.)

2.1.3 Raspbian Wheezy

Raspbian Wheezy is a free operating system based on Debian distribution. It is created by a small team of developers who are fans of Raspberry Pi. Raspbian is optimized for the Raspberry Pi's hardware and it comes with over 35 000 packages and pre-compiled software. Raspbian is still under active development and it aims to improve the stability and performance of the Debian packages. (Raspbian [Ref. 15.2.2015])

Raspbian is officially recommended for beginners and it includes the graphical desktop environment called LXDE. Raspbian Wheezy is one of the fastest ways to setup and get the RasPi running. (McManus, S. & Cook, M. 2013, 20.)

2.2 Introduction to Raspberry Pi

Raspberry Pi is a credit-card sized, fully featured computer which runs the Linux operating system (Figure 1). Raspberry Pi has all the necessary connection ports where user can plug peripheral devices. A monitor can be plugged through an HDMI, a mouse and a keyboard to the USB ports and for the speakers Raspberry provides a 3.5mm audio jack. In the model B+ there is also an Ethernet socket for the internet connection. (Monk 2013, 1.)

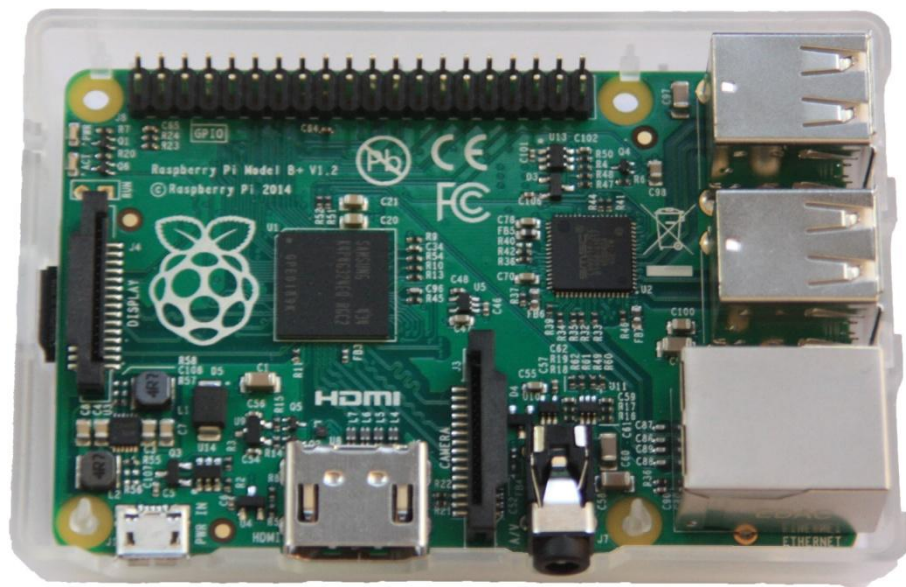


Figure 1. Raspberry Pi model B+

All in all it is a capable little computer which can be used for the same tasks which can be done with a normal desktop computer. For instance, reading emails, surfing on the websites, word-processing or watching high-definition videos. Also it is very popular in different kind of electronic projects and as a tool for learning the programming. (Raspberry Pi Foundation [Ref. 1.2.2015].)

Raspberry Pi is relatively cheap and the prices are starting from 25 \$. Because of its cheap starting price some components are not included and they need to be bought separately. For example, it does not have any kind of protective enclosure and the micro-USB power supply is not included either. (Monk 2013, 1.)

The idea of Raspberry Pi came from the computer laboratory of Cambridge University where the designer Eben Upton noticed in 2006 that the students who applied to study computer sciences started to have less programming experience than the students of the past years. Six years later the first Raspberry Pi was released in February 2012. (McManus, S. & Cook, M. 2013, 10.)

2.2.1 Available versions of Raspberry Pi

At the moment there are several different versions available of Raspberry Pi. The newest version is the Raspberry Pi 2 Model B. It is the second generation Raspberry Pi and it was published in February 2015. (Raspberry Pi Foundation [Ref. 22.2.2015]) The different versions and their technical specifications are shown in the Table 1.

Table 1. Different versions of Raspberry Pi (Element14 [Ref. 22.2.2015].)

Models & specifications			
Model	Raspberry Pi 2 B	Raspberry Pi B+	Raspberry Pi A+
Processor	BCM2836 ARMv7	BCM2835 ARMv6	BCM2835 ARMv6
CPU frequency	900 MHz	700 MHz	700 MHz
RAM	1 GB SDRAM	512 MB SDRAM	256 MB SDRAM
Storage	MicroSD	MicroSD	MicroSD
Power Draw & Voltage	600mA - 1.8A @5V	600mA - 1.8A @5V	600mA - 1.8A @5V
GPIO	40 pin	40 pin	40 pin
HDMI	Yes	Yes	Yes
USB 2.0	4 Ports	4 Ports	1 Port
Ethernet Port	Yes	Yes	No
Audio	3.5mm audio jack and composite video	3.5mm audio jack and composite video	3.5mm audio jack and composite video

2.2.2 Programming languages

There are considerable numbers of programming languages which have been adapted for Raspberry Pi. Python programming language is recommended by The Raspberry Pi foundation especially for the beginners. Basically any programming language which can be compiled for ARMv6 can run on the Raspberry Pi. Therefore the users are not restricted to use only the Python. On the Raspberry Pi there are preinstalled several languages for example C, C++, Java, Scratch and Ruby. (Raspberry Pi Foundation [Ref. 5.2.2015].)

2.2.3 General Purpose Input Output (GPIO)

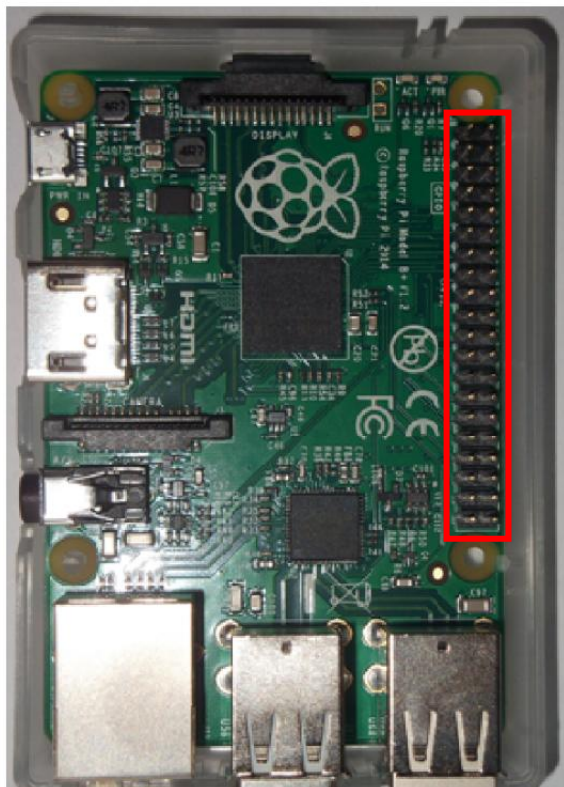
Raspberry Pi has two rows of pins on one side of it. These pins are called GPIO connector. The GPIO connector allows attachment of electronic hardware to the Raspberry Pi. It is an alternative option for a USB port. (Monk 2013, 115.)

The pins which are labeled as GPIO can all be used as general purpose input/output pins. It means that they can be defined to be either an input or an output pin. (Monk 2013, 115.)

The GPIO connector varies little bit in different Raspberry Pi models. In the earlier models B and A the GPIO connector consisted of 26 pins. In the model B+ the GPIO consists of 40-pin connector where the first 26 pins are same as in the earlier versions. (Adafruit 2015.) Table 2 shows the GPIO connector's pins on the Raspberry Pi model B+.

Table 2. GPIO connector's pin layout on Raspberry Pi Model B+

GPIO pin layout Model B+			
3V3	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	Ground
GPIO4	7	8	GPIO14
Ground	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	Ground
GPIO22	15	16	GPIO23
3V3	17	18	GPIO24
GPIO10	19	20	Ground
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
Ground	25	26	GPIO7
ID_SD	27	28	ID_SC
GPIO5	29	30	Ground
GPIO6	31	32	GPIO12
GPIO13	33	34	Ground
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
Ground	39	40	GPIO21



The pins are listed from top left corner, so that odd numbers are at the left side and even numbers are at the right side. Some of the GPIO connector's pins have extra labels after the pin name. They are markings for special features. For instance, GPIO2 and 3, have the labels of SDA and SCL. These pins are data and clock lines for a serial bus type. This serial bus type is called I2C and it is popular for communicating with peripherals such as temperature sensors and LC displays (LCD). (Monk 2013, 116.) Table 3 shows the GPIO pins with the special purposes.

Table 3. List of special purpose GPIO pins

Special purpose GPIO pins		
Pin #	Label	Explanation
GPIO2	SDA	Data line
GPIO3	SCL	Clock
GPIO9	MISO	SPI, Serial peripheral interface
GPIO10	MOSI	SPI
GPIO11	SCLK	SPI
GPIO14	Tx	Transmit pin for the serial port
GPIO15	Rx	Receive pin for the serial port
ID_SD		Reserved for Pi plates
ID_SC		Reserved for Pi plates

When using the GPIO as an input there are two states, it can be either "1" or "0". These states are described as voltage levels. The voltages which are above the 1.7V gives the first state "1" and the voltages below 1.7V gives the second state "0". For instance, if the GPIO gets the voltage of 1.65V it's input state would be "0". (Monk 2013, 116.)

Instead if the GPIO pin is defined to be an output there are also two states. These states are logical 1 and 0. When the GPIO pin is at the logical state 1, it means that the voltage level is then 3.3V. The logical state 0 is describing the voltage level of 0V. All the GPIO pins are 3.3V pins and connecting them to higher voltage could damage Raspberry Pi. (Monk 2013, 116.)

The maximum current from any of the GPIO pin is 16mA. This means that the pins can be used for controlling only small devices or lights which consumes low current. For instance, normal LED's (light-emitting diode) maximum current is approximately 10mA and it can be attached to the GPIO when there is a resistor connected to serial. (McManus, S. & Cook, M. 2013, 327.)

2.2.4 Raspberry Pi NoIR Camera module

As mentioned earlier there are existing additional components for Raspberry Pi which can be bought separately. One of these additional components is a Pi NoIR camera module. (Figure 2)

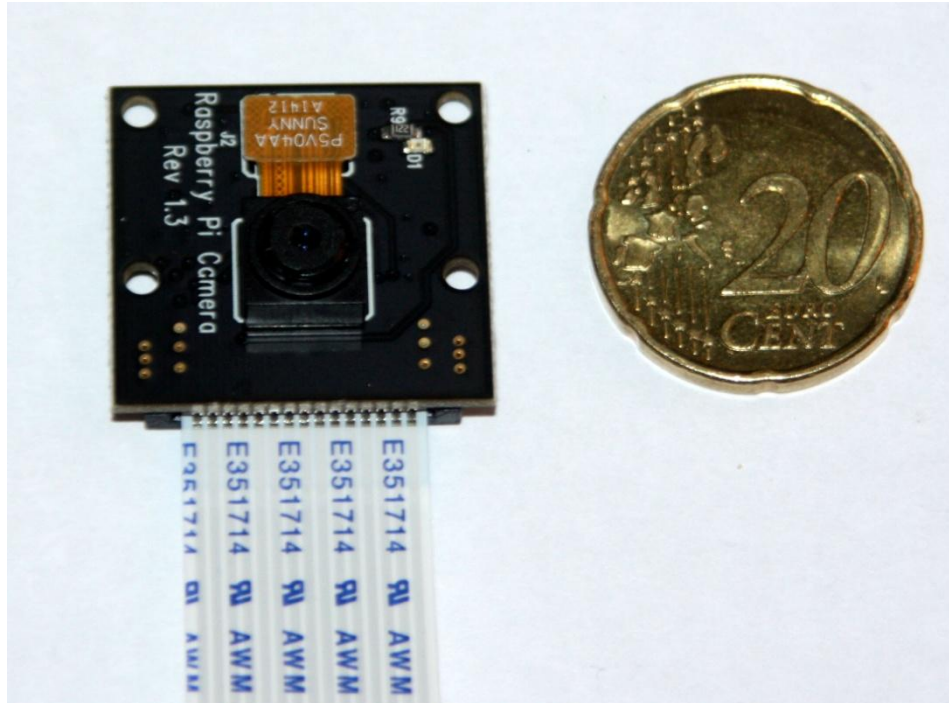


Figure 2. Pi NoIR camera module

NoIR comes from the words "No Infra-red". It means that Pi NoIR camera module does not have an infra-red filter. Because of that the pictures which are taken in daylight looks strange but it gives the ability to see in the dark with an infrared lightning. (Raspberry Pi Foundation [Ref. 25.3.2015].)

The camera module itself is a small sized, it can be compared to a 20 Cent coin as shown in the Figure 2. The camera has a maximum resolution of 5 megapixels (2592×1944 pixels). It uses an Omnivision 5647 sensor and has a fixed focus lens on board. The camera module is also capable of recording full high definition videos. (Raspberry Pi NoIR Camera Board [Ref. 25.3.2015].)

The camera module is a compatible with all the Raspberry Pi models and it can be accessed through MMAL and V4L APIs. At the moment there exists numerous

third-party libraries built for the camera module. One of these libraries is a Picamera Python library. (Raspberry Pi Foundation [Ref. 25.3.2015].)

2.3 1-Wire technology and DS18S20 temperature sensors


Single wire (1-Wire) technology is a serial protocol which uses a single data line and a ground line for the communication. The communication inside the 1-Wire bus is controlled by the 1-Wire master device which controls the slave devices such like sensors attached to the 1-Wire bus. Each 1-Wire device has a unique identification number on it which is also called device address. Mostly the 1-Wire devices does not have a pin for a power supply and they will get their power from the 1-Wire bus. This connection method is known as parasite powering. (Linke, B. 2008.)

Typical 1-Wire network consists of master, and one or more slave devices connected to the 1-Wire bus. The 1-Wire protocol supports two different communication speeds. In a standard mode the communication speed is 15,4 kbps and in overdrive mode it is 125 kbps. (Maxim Integrated [Ref. 15.3.2015].)

The DS18S20 1-Wire temperature sensors (Table 4) are manufactured by the Maxim Integrated Company. The DS18S20 provides 9-bit Celsius temperature measurements in an operating range from -55 Celsius to +125 Celsius degrees. The sensor can derive power directly from the data line (DQ). This wiring method is known as parasite powering. It eliminates the need for an external power supply. Each of the DS18S20 sensors have the unique 64-bit serial number which allows multiple sensors to work on the same 1-Wire bus. (DS18S20 Datasheet 2015, 5.)

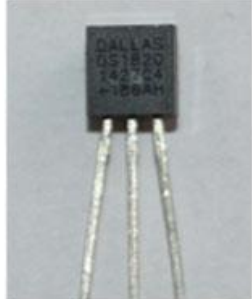
Table 4. Technical specifications of the DS18S20 sensor

Key specifications of the DS18S20 sensor	
Sensor type	Local
Interface	1-Wire
Accuracy (\pm °C)	0.5
Parasite Power Mode	Yes
Temp. Tresh.	Programmable (NV)
Temp. Resolution (bits)	9
Operating Temp. (°C)	-55 to +125



DS18S20

1 2 3



DS18S20 Thermometer		
PIN	Signal name	Description
1	GND	Ground
2	DQ	1-Wire Data
3	Vdd	Optional Vdd

2.4 Python programming language

Python programming language is developed in the late 1980s at the National Research Institute by Guido van Rossum. Python has grown in popularity, and it is widely used commercially. (Upton, E. & Halfacree, G. 2012, 152.)

Python is a flexible and powerful programming language but still it is easy to learn and follow. The clear syntax of Python makes it a valuable tool for users who wants to learn programming. This is one of the reasons why it is recommended by the Raspberry Pi Foundation. Python is published under an open-source license and it is available for different operating systems. Python runs on Linux, OS X and Windows computer systems. (Upton, E. & Halfacree, G. 2012, 152.)

Cross-platform support guarantees that the programs which are written in Python are also compatible in other platforms. There are few exceptions where the programs are not compatible. For instance, when the Python is addressed to use the specific hardware such like Raspberry Pi's GPIO. (Upton, E. & Halfacree, G. 2012, 152.)

2.5 SQLite3

SQLite3 is a free SQL (Structured Query Language) database engine software. It can be used for private and commercial purposes. That is one of the reasons why it is widely used in different kinds of applications and projects. SQLite3 does not have a separate server process like many other SQL databases. It reads and writes to ordinary disk files, and the complete SQL database is contained in a single disk file. (SQLite [Ref. 18.2.2015].)

For its small size SQLite3 is very capable and compact database engine, with all features enabled the library size can be less than 500 KB. It is very carefully tested before new releases are made. That is a part reason why SQLite3 has a reputation for being a very reliable tool. SQLite3 is supported by international developers who are working on SQLite3 full time expanding the capabilities and improving its reliability and performance. (SQLite [Ref. 18.2.2015].)

2.6 Apache2

Apache2 is an open-source HTTP server project which is highly configurable and extensible with third-party modules. The server can be customized by writing modules and using the Apache module API. It is the project of the Apache Software Foundation and it runs on all modern operating systems. Apache was launched in 1995, and since 1996 it has been the most popular web server on the Internet. (The Apache2 project [Ref. 19.2.2015].)

Apache is in under active development and encourages to user feedback. Many frequently asked features have been implemented. The Apache project's goals are to provide a secure, efficient and extensible server that provides HTTP services in sync with the current standards. (The Apache2 project [Ref. 19.2.2015].)

3 IMPLEMENTATION OF A HOME SURVEILLANCE SYSTEM

This chapter contains the device and software installations, creating a 1-Wire network for measuring temperatures and creating a graphical user interface. The graphical user interface is based on a website which is hosted by Apache2 HTTP server.

3.1 Setting up Raspberry Pi

As said earlier Raspberry Pi comes without any peripheral devices. The first thing to do is to unpack RasPi and protect it with an enclosure (Figure 3). Raspberry Pi can be installed to the protective enclosure without using any tools. The enclosure has plastic clips which are holding the Raspberry Pi in its place.

After Raspberry Pi has been installed to enclosure and well protected, all the necessary peripherals can be attached to it. Just like any other computer, Raspberry Pi needs some basic devices such as display which is connected via the HDMI cable, the mouse and the keyboard, and the internet connection cable.



Figure 3. Raspberry Pi in protective enclosure

Before plugging the power cable, MicroSD-card should be checked if it is flashed and prepared with an operating system. Also it is recommendable to create a backup folder of the MicroSD-card just in case of complications.

The MicroSD-card can be checked with a card-reader. The card-reader can be found from most of the laptops and desktop computers. Insert the MicroSD-card into the card-reader and check that there is something stored in the MicroSD-card. If everything looks good, take the MicroSD-card and plug it into the Raspberry Pi. Now the power cable can be connected.

Raspberry Pi does not have any kind of power switch so it will start up immediately when the power cable is connected to it. At the start up text starts to flow on the monitor and shortly after that there appears a configuration menu. The configuration menu is called Raspi-config (Figure 4). In Raspi-config it is possible to change some of the settings on Raspberry.



Figure 4. Raspi-config

The most important settings that should be checked in Raspi-config are:

- Expand Filesystem, where it is necessary to check that RasPi can use the whole memory capacity of the MicroSD-card. Otherwise the memory can run out fast.
- Internationalisation Options, where it is possible to choose between different languages and the time zones.
- Advanced Options, if the internet cable has been plugged in, it is possible to update RasPi to the latest version available. (McManus, S. & Cook, M. 2013, 38.)

It is recommendable that users who do not have so much experience with Linux operating systems should choose the English language because then help and advice can be found more easily from the internet.

It is possible to get back to the Raspi-config and change the settings also after the first setup by typing the following command into the terminal:

```
sudo raspi-config
```

After making the changes on the Raspberry Pi's settings, the settings can be accepted by choosing the Finish option. Now the terminal view should appear and it might be asking for the username and the password. The username in Raspian Wheezy is by default **pi** and the password should be **raspberry**. Notice that these are written in small letters. The Linux is letter case sensitive and it will recognize the difference between small and capital letters.

The next step is logging in to Raspberry and instead of the graphical environment there will be a command console flashing. However, the graphical environment, or so called desktop view, can be started by entering the command:

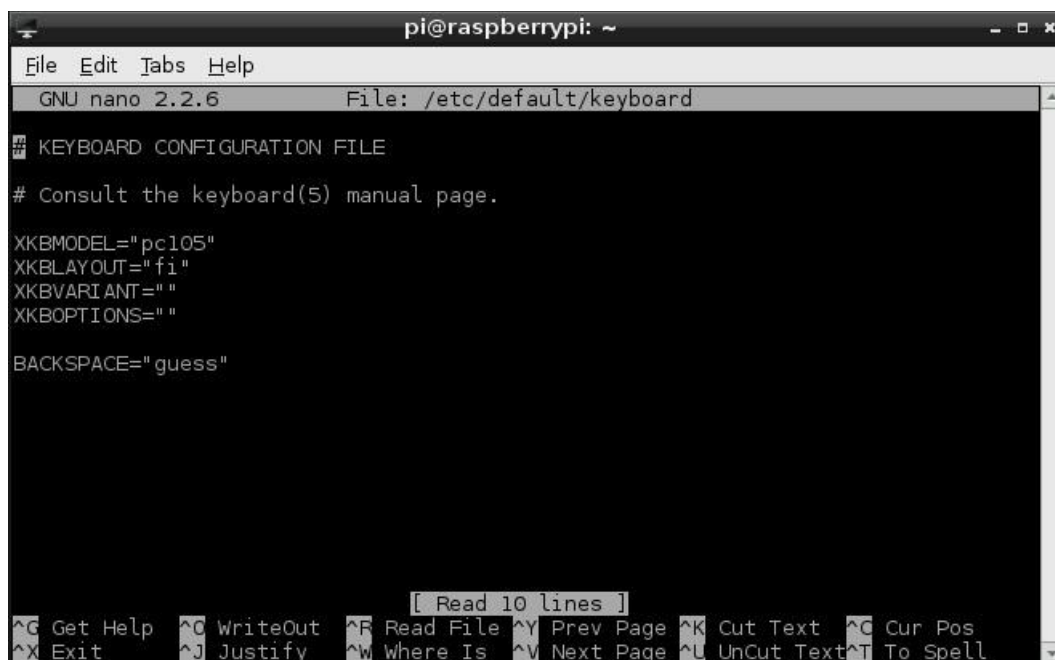
```
startx
```

Now Raspberry will be loading for a while and a few seconds later there will appear a more user friendly desktop view. It is recommended to learn how to use the command console as it makes some of the actions faster than doing them in the desktop view.

So far the basic configurations are made for the Raspberry. There might still be some things that are not working correctly. For instance, the keyboard layout might be defined to be in UK style which is the default keyboard layout setting on Raspberry Pi. This can be frustrating and annoying. The layout can be changed easily by opening the LXTerminal which opens the command console. Open the keyboard file in the command console with the nano text editor by typing the following command:

```
sudo nano /etc/default/keyboard
```

The keyboard configuration file (Figure 5) will appear and it can be modified. The keyboard layout can be changed by replacing the XKBLAYOUT value as shown in Figure 5. After the file is edited it can be saved by pressing CTRL + O key combination.

A screenshot of a terminal window on a Raspberry Pi. The window title is 'pi@raspberrypi: ~'. The terminal shows the nano text editor editing the file '/etc/default/keyboard'. The editor's status bar at the top indicates 'GNU nano 2.2.6' and 'File: /etc/default/keyboard'. The main content of the file is as follows:
KEYBOARD CONFIGURATION FILE
Consult the keyboard(5) manual page.

XKBMODEL="pc105"
XKBLAYOUT="fi"
XKBVARIANT=""
XKBOPTIONS=""

BACKSPACE="guess"
The bottom of the screen shows a status bar with various keyboard shortcuts: [Read 10 lines], ^G Get Help, ^O WriteOut, ^R Read File, ^Y Prev Page, ^K Cut Text, ^C Cur Pos, ^X Exit, ^J Justify, ^W Where Is, ^V Next Page, ^L UnCut Text, ^T To Spell.

Figure 5. The keyboard configuration, setting the layout to the Finnish “fi”

3.2 Creating the 1-Wire network

This chapter describes how to setup a 1-Wire network and how to attach the DS18S20 temperature sensors to it. The 1-Wire bus is created by using the GPIO connector on Raspberry Pi.

When creating 1-Wire bus for the first time it is recommendable to create the 1-Wire bus in a test environment. It lowers the risk of complications and makes it easier to test different programs and wirings. It also gives a better overview of how everything is working.

There are two different wiring possibilities for the DS18S20 temperature sensor. The sensor can use an external power supply, where the VDD pin is in use or it can be connected with the method of a parasitic power supply, where the sensor takes the power from the DQ pin.

According to the DS18S20 datasheet the 1-Wire bus requires a pull-up resistor. The impedance for the pull-up resistor should be approximately 5k Ohms. (DS18S20 Datasheet 2015, 10.)

The pieces of equipment and the tools used to create the test environment are:

- breadboard
- jump wires
- ribbon cable 40 pin
- pull-up resistor (4700 ohm)
- datasheet for DS18S20 sensor.

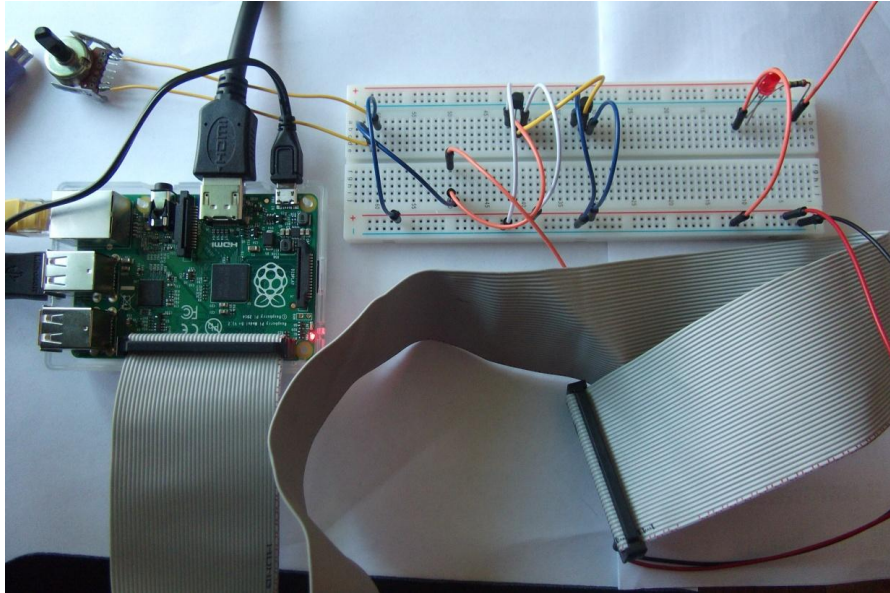


Figure 6. Testing the DS18S20 temperature sensors through the GPIO

In Figure 6 the temperature sensors are placed to a breadboard. The benefit of the breadboard is that it does not require any soldering. It is a fast and an easy way to modify the wirings. The breadboard itself has been connected to the RasPi with the 40 pin ribbon cable. The Figure 7 presents the wiring diagram with the external power supply method.

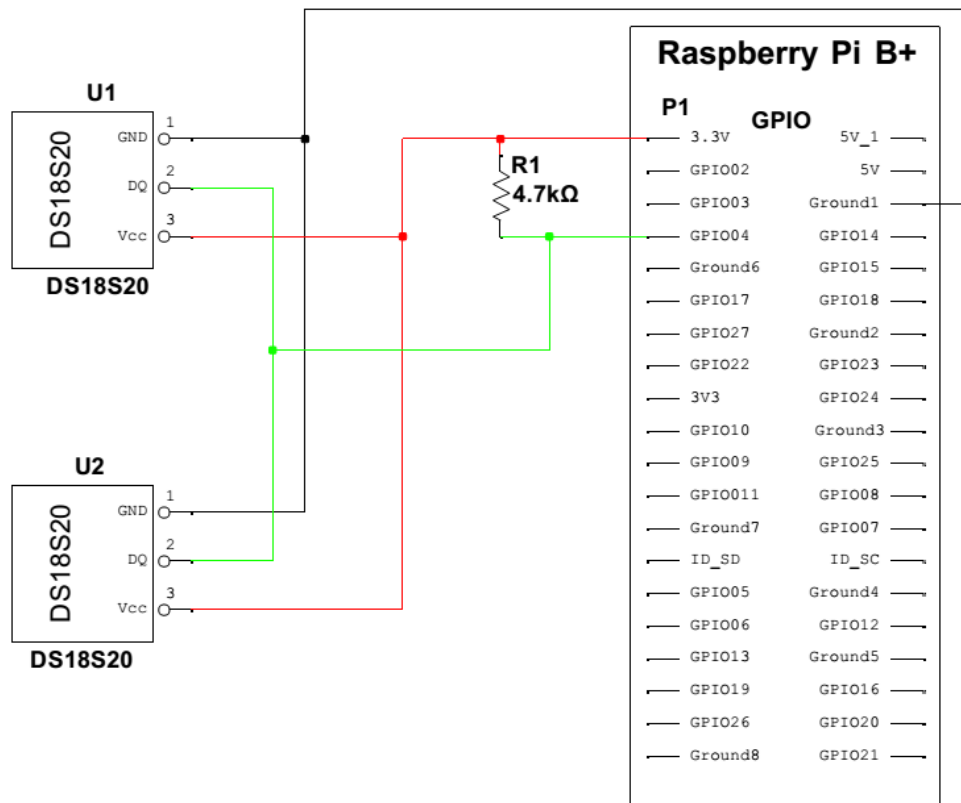


Figure 7. External power supply wiring diagram

3.3 Measuring the temperatures and creating the database

This chapter demonstrates how to read and measure the temperatures from the DS18S20 1-Wire temperature sensors. The temperature data is then stored into the SQLite3 database for later use. SQLite3 is being used in this bachelor's thesis as a database engine.

3.3.1 Measuring the temperatures

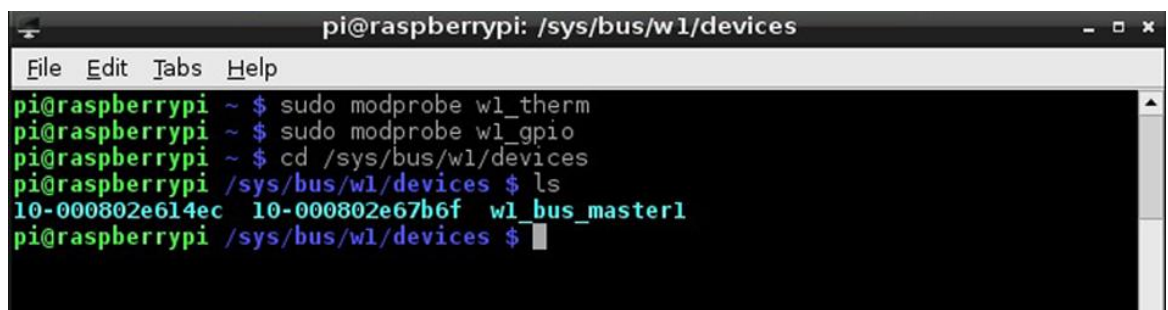
Before starting to measure the temperatures, it is necessary to create an external power supply connection which is shown in Figure 7. In the next step two additional kernel modules need to be loaded. These modules are called `w1_therm` and `w1_gpio`. The modules are preinstalled in the Raspbian distribution and they can be manually loaded by entering the following commands to the terminal:

```
sudo modprobe w1_gpio
sudo modprobe w1_therm
```

Where the `w1_gpio` module is a 1-wire bus master driver and the `w1_therm` module is the temperature conversion module for the Maxim DS18*20 and DS1825 based temperature sensors. (The Linux Kernel Archives, 2013.) After loading these modules the 1-Wire bus can be explored by entering the command:

```
cd /sys/bus/w1/devices/      //Opens the W1 devices folder
ls                          //Lists the files in current folder
```

Now there appear some directories (Figure 8) which are named with sensors serial numbers. These serial numbers will be used later in a Python script so it is useful to write them down. In case that there will not appear any directories, check the wirings and repeat the steps mentioned above.



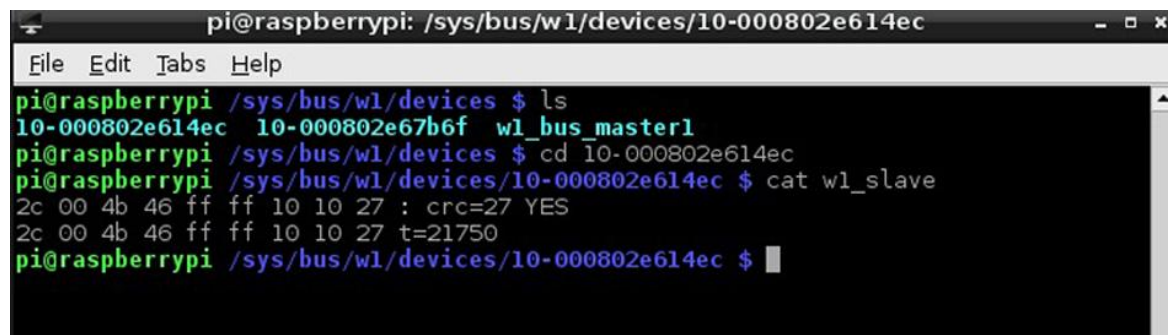
```
pi@raspberrypi: /sys/bus/w1/devices
File Edit Tabs Help
pi@raspberrypi ~ $ sudo modprobe w1_therm
pi@raspberrypi ~ $ sudo modprobe w1_gpio
pi@raspberrypi ~ $ cd /sys/bus/w1/devices
pi@raspberrypi /sys/bus/w1/devices $ ls
10-000802e614ec 10-000802e67b6f w1_bus_master1
pi@raspberrypi /sys/bus/w1/devices $
```

Figure 8. Checking the DS18S20 temperature sensors serial numbers

The current temperature in a room can be read by opening one of the directories shown in `/sys/bus/w1/devices` and opening the `w1_slave` file. This is done with following commands:

```
cd 10-000802e614ec
cat w1_slave
```

The current temperature will be printed on a terminal where the value of "t=" is presenting the temperature in degrees Celsius. However, the value of "t=" can seem a bit strange at first because it does not have a comma after the first two numbers. In Figure 9 the temperature is presented as "t=21750" which would be 21,75 degrees Celsius. The next chapter will present how to create the SQLite3 database where the measured data from the temperature sensors can be stored.



```
pi@raspberrypi: /sys/bus/w1/devices/10-000802e614ec
File Edit Tabs Help
pi@raspberrypi /sys/bus/w1/devices $ ls
10-000802e614ec 10-000802e67b6f w1_bus_master1
pi@raspberrypi /sys/bus/w1/devices $ cd 10-000802e614ec
pi@raspberrypi /sys/bus/w1/devices/10-000802e614ec $ cat w1_slave
2c 00 4b 46 ff ff 10 10 27 : crc=27 YES
2c 00 4b 46 ff ff 10 10 27 t=21750
pi@raspberrypi /sys/bus/w1/devices/10-000802e614ec $
```

Figure 9. Reading the temperature

3.3.2 Setting up the database for measured data

In this bachelor's thesis SQLite3 is used to store the measured data from the temperature sensors because of its compactness and reliability. Also it does not require any kind of configuration.

The SQLite3 database program needs to be installed to the Raspberry and it can be done by typing the following command in the terminal:

```
sudo apt-get install SQLite3 //Install the SQLite3
```

After the SQLite3 installation, there are two possibilities to create the database. The first possibility is to use the SQLite3's command shell or the second possibility is to install an external application for creating and browsing the databases.

Using the external application turned out to be easier for handling the databases and for observing the stored data. In this bachelor's thesis the external application called SQLite database browser (Figure 10) is used to create the database for temperature sensors. SQLite database browser can be installed like any other Linux program by using the apt-get install command. After installing the SQLite database browser it can be started from the Programming section.

The next step is to create a new database called temperatures. The program propose to create a table and requests name for the table. The table can be named as "temperatures". Then the table needs new fields called Date, Time, Room and Temperature and their field types are DATE, TIME, TEXT and NUMERIC. Remember finally to save all the changes to the database.

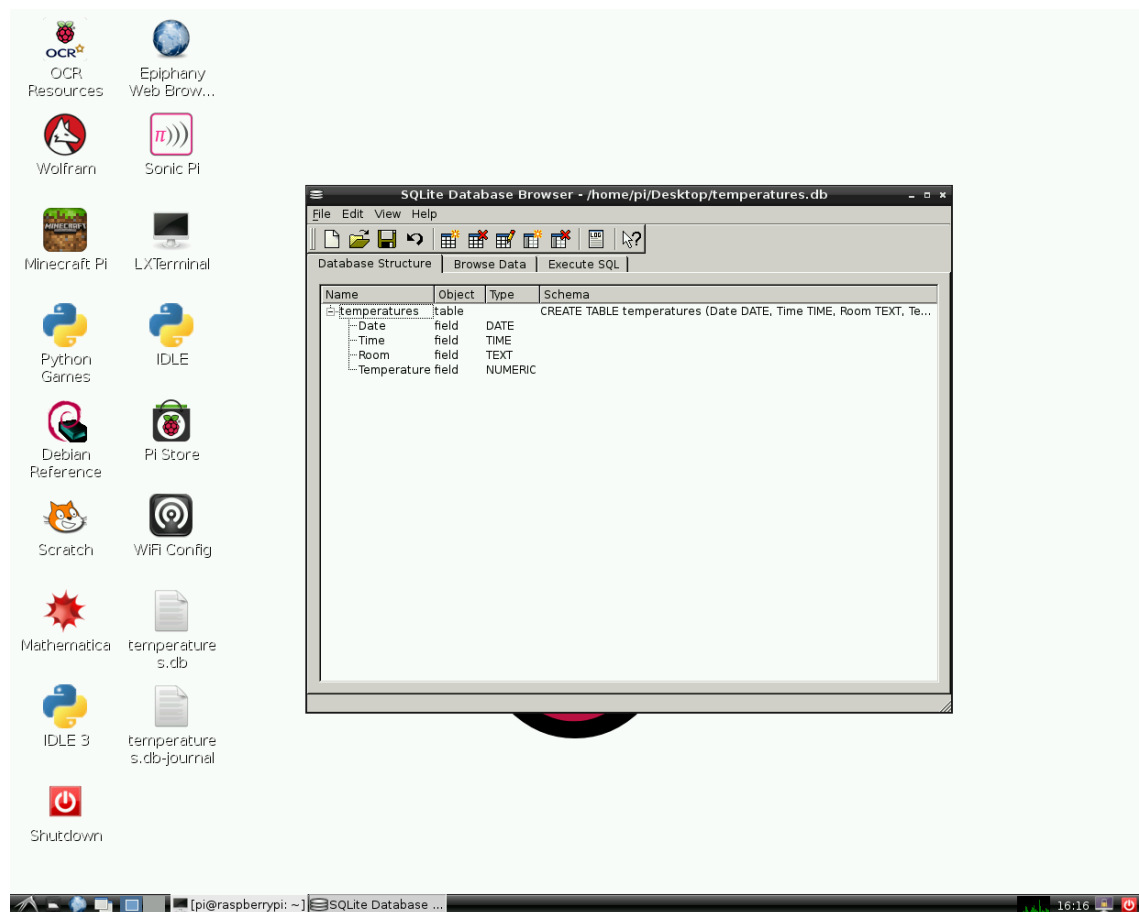


Figure 10. Finished database structure

3.3.3 Creating a Python script to measure and store the data

So far the temperatures have been measured manually by using the terminal and the temperatures database is created where the measured data will be stored. The next step is to create a Python script which reads the temperature data from the sensors and stores it to the database.

It is possible to write Python scripts with any of the preinstalled text editors such like LeafPad. However, when it comes to the more complicated scripts it is recommended to use a Python editor to avoid unnecessary syntax errors. The PyDev program has been used in this bachelor's thesis as the Python editor.

The first step is to create a new script file called monitor.py. The first line of the script is a "shebang" line. This line indicates how the script should be executed.

```
#!/usr/bin/env python
```

After inserting the shebang line the SQLite3 library needs to be imported to the script. This can be done with following Python command:

```
import sqlite3
```

Importing the SQLite3 library makes it possible to use SQLite3 commands in the Python script. After importing the library global variables can be defined. The global variables in this case are: room names, the sensor id-numbers and the database location.

```
#Rooms where the sensors are:
sensor1_room="Kitchen"
sensor2_room="Garage"

#Sensors ID-numbers:
sensor1_ID="/sys/bus/w1/devices/10-000802e614ec/w1_slave"
sensor2_ID="/sys/bus/w1/devices/10-000802e67b6f/w1_slave"

#Database name
dbname='/var/www/temperatures.db'
```

These global variables are sent to the `getTemperature`-function, which opens the `w1_slave` file from the `sensor_ID` location. The data in `w1_slave` file is then stored into the text variable.

```
def getTemperature(ID, sensor_room):
    tfile=open(ID)
    text=tfile.read()
    tfile.close()
```

At this point the text variable also includes unnecessary details which are not essential. The text variable is parsed in the next few lines of Python code.

At the first, the text variable is split with the new lines and the second line is selected. Then the second line is split into the words by referring to the spaces. The temperature is located at the 10th word and it is selected. The line and word numbering starts from zero and that is the reason why the second line is called with `[1]` and the 10th word is selected by calling `[9]`. (Kirk, M. 2012.)

```
#Split the text with new lines and select the second line
secondline= text.split("\n")[1]

#Split the line into words and select the 10th word
temperaturedata= secondline.split(" ")[9]
```

Now the 10th word is selected and it still includes the marking `"t="`, so the first two characters of the 10th word need to be deleted. The string variable also needs to be converted to a number by using the **`float()`** command. The decimal point should be also placed on its place. (Kirk, M. 2012.)

```
temperature=float(temperaturedata[2:])
sensor1_data = temperature/1000
```

At this point the sensor's temperature data is stored into variable called `sensor1_data`. This measured temperature data can be now inserted into the database. The connection to the database can be now opened and the cursor can be created with the following commands:

```
#Connect to the temperature database
db=sqlite3.connect(dbname)
cur = db.cursor()
```

After the connection to the database has been created, the `sensor1_data` can be inserted to the database. The data is inserted into the table called `temperatures`, where four columns exists: Date, Time, Room and the room's Temperature.

```
cur.execute("INSERT INTO temperatures
VALUES(date('now','localtime'), time('now','localtime'), (?),
(?));", (sensor_room, sensor_data,))
```

It is important to remember to commit all the changes which have been done in database and afterwards to close the database connection.

```
db.commit()          #Commit changes in database
db.close()           #Close connection
```

Now the `getTemperature`-function can be called as many times as there are sensors connected to the 1-Wire bus. In this case there are two sensors and the function is called twice in a main-function.

```
def main():
    getTemperature(sensor1_ID, sensor1_room)
    getTemperature(sensor2_ID, sensor2_room)
```

When calling the main-function, it sends the sensor-ID and the sensor's room variables to the `getTemperature`-function. The `getTemperature`-function stores the current temperature data to the database at the moment when it is called. Finally remember to save the script into `/home/pi` location. After saving the script it can be ran through the terminal by calling it:

```
python monitor.py
```


After running the script several times, check the database if it is receiving the data which is sent through the script. The database can be browsed with the SQLite Database browser program. Open the temperatures.db database with the SQLite Database browser and then click the Browse Data tap. The stored data should appear into the screen.

3.4 Installing the Apache2 HTTP server

The Apache2 HTTP server is used to publish the temperature data on a website. The website is supposed to show the measured data in a form which can be easily observed over an internet browser.

The Apache2 server needs to be installed to Raspberry and it can be done in the terminal by typing:

```
sudo apt-get install apache2
```

After the installation, the server needs to be configured before it can be used to run Python scripts. This can be done by editing the Apache's configuration files. In the terminal, open the configuration file called 000-default by typing the following command:

```
sudo nano /etc/apache2/sites-enabled/000-default
```

In the 000-default configuration file search for the code <Directory "/usr/lib/cgi-bin"> and add a cgi-script handler to below this section of the code.

```
AddHandler cgi-script .py
```

The configuration file needs to be saved after the changes are made. After editing the configuration file the Apache2 service needs to be restarted in order for changes to take the effect. The service can be restarted by entering the following command in the terminal:

```
sudo service apache2 reload
```

At this point the Apache2 server is able to execute the Python scripts. However, there is still some configuration to do. The Apache2 uses by default a user called www-data which belongs to a group www-data. This can cause some error messages while starting the Apache2 service.

There are two possibilities to fix this problem. The first possibility is to create the new user called "www-data" and the second possibility is to change the Apache2's user. In this bachelor's thesis the Apache2's user is changed to "Pi". It can be changed by opening the Apache2's configuration file called envvars with the nano editor.

```
cd /etc/apache2/ // Opens the Apache2 directory
sudo nano envvars // Opens the configuration file envvars
```

In the envvars configuration file search for the lines:

```
export APACHE_RUN_USER=www-data
export APACHE_RUN_GROUP=www-data
```

Replace the text "www-data" in both lines to the text "pi" and save the envvars configuration file. Then the Apache2 service needs to be restarted again. Now configuration for the server is finished. The locations for the website files and the Python scripts are located at:

```
/var/www/ // Location for the html files
/usr/lib/cgi-bin/ // Location for the executable scripts
```

The Apache2 HTTP server can be accessed from the web-browser by using the Raspberry Pi's IP-address or alternative option is to use a localhost. The IP address is possible to check from the terminal with the following command:

```
ifconfig
```

Notice that in this point the Apache2 server might not be able to take the connections which are coming outside of the home network. This can occur if a router is not properly configured. The router might block the Apache2's communication if a port number 80 is not assigned to be open. In this case check your router's model and search the router specific instructions for port forwarding and follow the instructions.

3.5 Controlling the GPIO pins with Python

This chapter discovers the GPIO connector and how it can be used in controlling. The first experiments with the GPIO were to light up a LED (light-emitting diode) through the Python Shell.

The second experiment is little bit more complex and it demonstrates the controlling loop of an heating element. The heating element will start to heat the room when room's temperature is getting below the pre-defined lower limit and stops heating when the temperature in the room reaches the second pre-defined upper limit.

3.5.1 Controlling the LED with the GPIO

This is the first experiment with the GPIO connector and it demonstrates how to use it in controlling. This experiment requires a LED and a resistor. The resistor's resistance can be calculated from the Ohm's law which is shown in Formula 1.

Defining the resistance from the Ohm's law:

$$U = R * I \quad (1)$$

$$R = \frac{U}{I} = \frac{3.3V_{GPIO \text{ voltage}} - 1.18V_{LED \text{ voltage drop}}}{10mA_{LED \text{ current}}} = 212 \Omega \quad (2)$$

Where

- U is voltage
- R is resistance
- I is current

The resistors above 212Ω are suitable and can be used for lighting the LED directly from the GPIO. The wiring for the LED and resistor is shown in Figure 11.

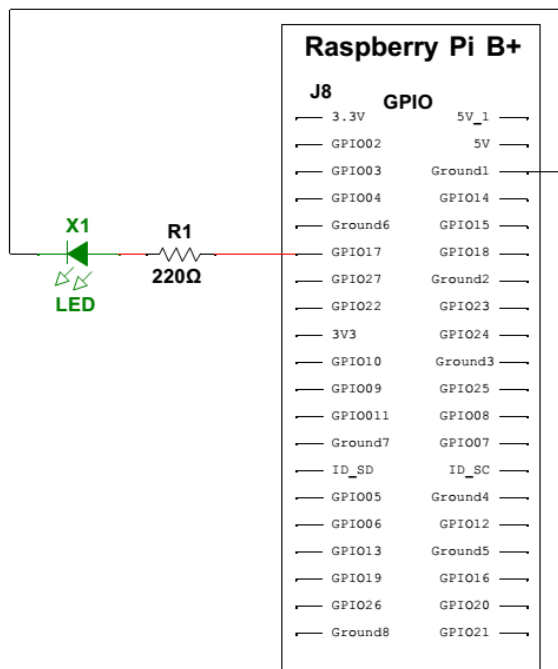


Figure 11. LED wiring diagram

After the wirings are done the Python library called python-rpi.gpio needs to be installed. This library allows controlling the GPIO pins. It can be installed with the following command:

```
sudo apt-get install python-rpi.gpio
```

When the installation is finished, open up a Python Shell from the terminal as root user and import the RPi.GPIO library.

```
import RPi.GPIO as GPIO
```

Next thing to do is to set the mode to use the pin numbers from the ribbon cable board and define one of the GPIO pins to be an output. For instance the GPIO 17:

```
GPIO.setmode(GPIO.BCM)      # Ribbon cable board
GPIO.setup(17, GPIO.OUT)    # Defines the GPIO17 to be output
```

Now it is possible to control the GPIO17 pin to high and low. The LED will light up when the pin 17 is set to high and when it is set to low the LED will turn off.

```
GPIO.output(17, GPIO.HIGH)   # Turns the GPIO17 to high
GPIO.output(17, GPIO.LOW)    # Turns the GPIO17 to low
```

3.5.2 Controlling the heating element

In this experiment two LEDs are controlled according to the current temperature in a room. The two LEDs are presenting the states of the heating element. The green LED is lightened up when the heating is activated in the room and the red LED is indicating that the heating is turned off.

This experiment needs two temperature limits before its working principle is reasonable. The first low limit value defines the lowest acceptable point for the room's temperature when it is time to start the heating element. The second limit is the upper limit which defines when it is time to shut down the heater.

A Python script starts as usual by loading modules which are necessary. This time RPi.GPIO and time modules need to be loaded at the beginning of the script. The GPIO pins 17 and 18 are used and they must be set as output pins. The pins 17 and 18 are controlling the red and green LEDs. The room's current temperature is received in a getTemperature-function. The function's working principle is similar than in the monitor.py script, which is discovered in the chapter 3.3.3.

In the beginning of a main-function the starting temperature is defined by calling the `getTemperature`-function, and it is then compared to the heating limits. In case that the starting temperature is lower than the heating on limit the green LED will light up.

```
starting_temp=getTemperature(sensor_ID)
starting_temp=float(starting_temp)

if starting_temp <= temp_limit_on:
    GPIO.output(17, GPIO.LOW)           #Turn off the red LED
    GPIO.output(18, GPIO.HIGH)          #Light up the green LED
```

The green LED is turned on as long as the room's temperature reaches the heating off limit. The room's temperature is checked in certain intervals.

```
while(temp < temp_limit_off):
    time.sleep(10)                       #Waiting time in seconds
    temp=getTemperature(sensor_ID)       #check room temp
    print ("Heating on, temperature currently: %s" % temp)
```

3.6 Taking advantage of Raspberry Pi's camera module

This chapter is about the Pi NoIR camera module's installation to the Raspberry Pi, and observing the built in functions which are made for it. At the end of this chapter a Python script is created to take resized pictures. The pictures are named with current timestamp and saved to an own directory.

3.6.1 Installing the Pi NoIR camera module

The Raspberry Pi's NoIR camera module board comes in anti-static plastic bag. It is fast and easy to install. The camera module can be mounted to the protective case's cover, where is reserved slot for the camera. (Figure 12) It is screwed with two small screws, and the ribbon cable is connected to the Raspberry Pi's camera connection port. The connection port is located between the 3.5mm audio jack and the HDMI socket. The connection port's clip has to be pulled up before plugging the camera module's ribbon cable on its place.

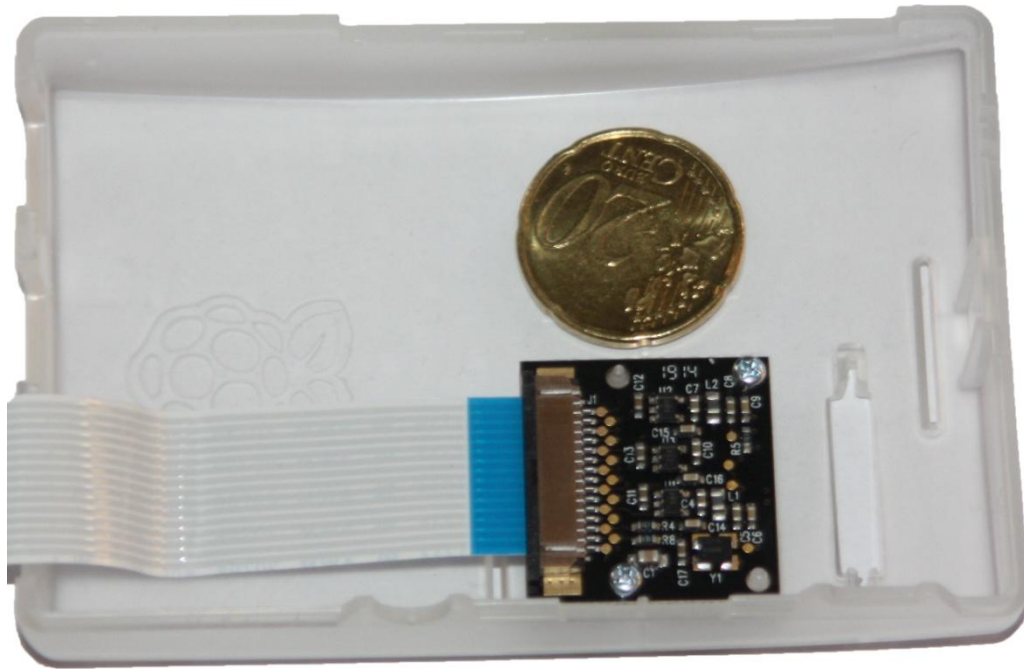


Figure 12. Installing the Pi NoIR camera module

After mounting the camera module, it is required to enable the camera module from the Raspi-config configuration tool and then Raspberry Pi has to be rebooted so that the changes will take effect.

3.6.2 Taking the first pictures and videos with the Pi NoIR camera

In Raspbian there are built in functions for the camera module. With these built in functions it is possible to take pictures and record videos, just to try out proper function of the camera module. One of these built in functions or commands is called “raspistill”.

```
raspistill -v -o first_image.jpg
```

After typing the command above into terminal a preview window is started up. The preview window is running for 5 seconds, and then Raspberry takes the picture, and saves it to the file called first_image.jpg. Parameters -v stands for verbose information during the run and with the -o parameter it is possible to give filename for the output file.

Other simple and useful parameters which can be added into raspistill command are:

- image width **-w**
- image height **-h**
- image quality **-q**
- flip the image vertically **-vf**
- flip the image horizontally **-hf**
- image rotation **-rot**

A complete parameter list can be found from the RaspiCam documentation. (RaspiCam Documentation. 2013. 5-18.)

3.6.3 Creating a Python script for taking pictures

First things to consider before creating the script which takes the picture and stores it automatically are: where the picture is stored, finding the right parameters for the picture so that the image quality and size does not suffer too much.

After a while, some limitations for the pictures are found. The size and quality are reduced to minimize the picture size on the hard drive. The Quality of 75% and the resolution of 1280x720 pixels are sufficient. With these parameters the picture size on the hard drive is around 500KB. That is good starting point, and trade-off between picture quality and available space for picture saving.

All the pictures which are taken by the Python script will be saved to the own folder with current timestamp filename. The folder is located at /var/www/camera/. Apache2 is hosting the folder so that the pictures are available on the website.

Creating the script starts with placing the shebang information and importing the necessary libraries. These libraries are datetime, picamera and time.

```
#!/usr/bin/env/ python

import datetime
import picamera
import time
```


On the second step a function called takePicture should be defined. It does not take any input variables. The function consists of three parts. The first part is the general settings, where the location to the saved pictures and the filename are defined.

```
def takePicture():
    location="/var/www/camera/"           #Location to the files
    date=datetime.datetime.now()         #Get current date
    file_name=date.strftime("%Y-%m-%d %H%M") #Format the string
```

The second part of the function is defining the settings for the picture size and it starts also the preview mode.

```
#configuration for the pictures
camera = picamera.PiCamera()
camera.resolution = (1280,720)
camera.start_preview()
```

In the last part of the function, the preview mode is kept on for a certain time to warm up the camera. After the warm up time, the function captures the picture and saves it to the predefined location. The picture is named with current timestamp. At the end of the script the preview mode is stopped and the camera is closed.

```
time.sleep(2) #Camera warm up time

# Capture the picture and saved it with the current date
camera.capture("%s%s.jpg" % (location,file_name), quality=75)
camera.stop_preview()
camera.close()
```

3.7 Creating scheduled tasks with Cron

Raspbian Wheezy has powerful task scheduler, which can be used to perform tasks automatically. It is known as Cron. Cron uses a cron table, which is also called crontab. It is a list of commands that user wants to be performed. Cron table takes several parameters which defines how often the task will be performed. (Computer Hope [Ref. 16.3.2015].)

Cron table can be displayed and edited by typing the commands below in terminal:

```
crontab -l      // Displays the cron table
crontab -e      // Opens the cron table in editor view
```

An example of the cron table is shown in Table 5. The job script.py is scheduled to be performed on June 10 at 8:30 AM. The first column in Table 5 shows the minute when the script is performed. The second and third columns defines the hour and day of the month when the script is performed. The last two columns defines the day of the week when the script is performed and the path to the script. Inserting the scripts which have been created earlier to the cron table allows temperature monitoring and taking pictures in certain time interval.

Table 5. Example of the crontab command (The Geek Stuff [Ref. 16.3.2015].)

Example of cron table					
Minute	Hour	Day of Month	Month	Day of week	Command
30	8	10	6	*	/home/example/script.py
30th minute	8:00 AM	10th day of month	June	At every day of week	Path to the script

3.8 Creating a website interface

This chapter presents the idea how to implement Python elements and how to create a website interface. Creating the website interface is time consuming and challenging. This bachelor's thesis uses an open source website template which is downloaded from dreamtemplates.com. The reason why the premade template is used in this bachelor's thesis is that it saved time and programming work. The template is little bit modified so that it suits better for the thesis purposes.

The website's front page consists of three main elements: a top menu, a left navigation bar and a main column. On the left navigation bar, lays a feed box where the website visitor sees current date, time when the website is updated, current temperature, minimum and maximum temperature of the day.

The feed box is a Python script displayed in an iframe element. The iframe is an HTML element which shows another content from a different source. In this case it shows the MinMax.py script.

```
#HTML code part from a index.html
<iframe src="/cgi-bin/MinMax.py" name="Temps" marginwidth="0"
marginheight="0" align="middle" frameborder="1" height="250"
scrolling="no" width="186"></iframe>
```

The MinMax script consists of as many functions as there are displayed data in the feed box. The main-function is gathering the data from other functions which retrieves the data from the temperatures database. Each of the functions which are called in the main-function returns one of the displayed feeds. The main-function also includes the HTML code and CSS styling for the feed-box element.

The pictures which are taken with the Pi NoIR camera module are stored into a camera section. The pictures are taken every 15 minutes and named with current timestamp.

In a monitoring section visitor sees a graphical visualization of the rooms temperatures. The rooms temperature data can be observed within certain intervals. The interval for the observed temperatures can be changed from the left navigation bar. The first radio button group determines the time interval and the second radio button group determines the room which is showed in a line chart. The visitor can track the temperature data points from the curve by moving the cursor on the line. (Figure 13)

The time scale can be selected within current date and the past 7 days options. On the room selector the visitor can choose between three options. The first option shows both rooms in the same line chart. This option is set by a default option when the website loads. The second and third options changes between a garage and a kitchen rooms.

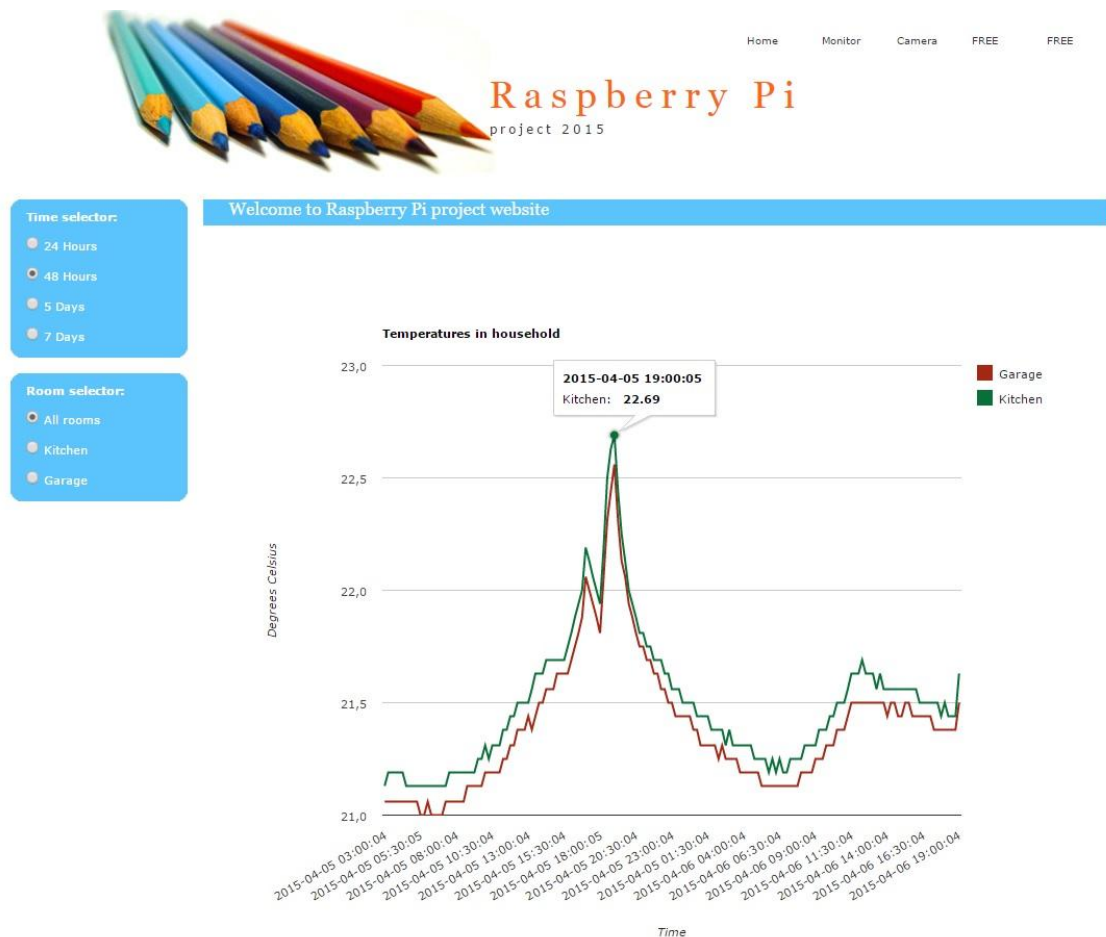


Figure 13. Line chart code generated with criteria

The actual line chart code is based on the JavaScript code and it is embed in the website. The chart code is generated by the Google charts which provides a wide collection of charts and figures which can be used on the websites for data visualization.

Implementing the line chart code with the Google charts is simple. All it requires is loading Google chart libraries, creating the data-table which is presented in the

line chart and creating a chart object with an id. The chart can be customized to its final look with a JSFiddle tool. The last thing to be done is to create a <div> element with the same id which was chosen in a chart object.

The Python script needs to generate this data table for the line chart. The data table is based on the criteria which have been sent from the website. For instance, the data table needs to be different when the observer chooses a different time scale.

The criteria from the website are sent to the Python script by the get-method. The criteria are then read in the Python script by using a cgi-module. With the cgi-module it is possible to get the values from an HTML form. Below is shown an example of the get-method passing the HTML form criteria to the Python script.

#URL code passing the criteria to the Python script
http://yourdomain/cgi-bin/script.py?time_selector=2&room_selector=1

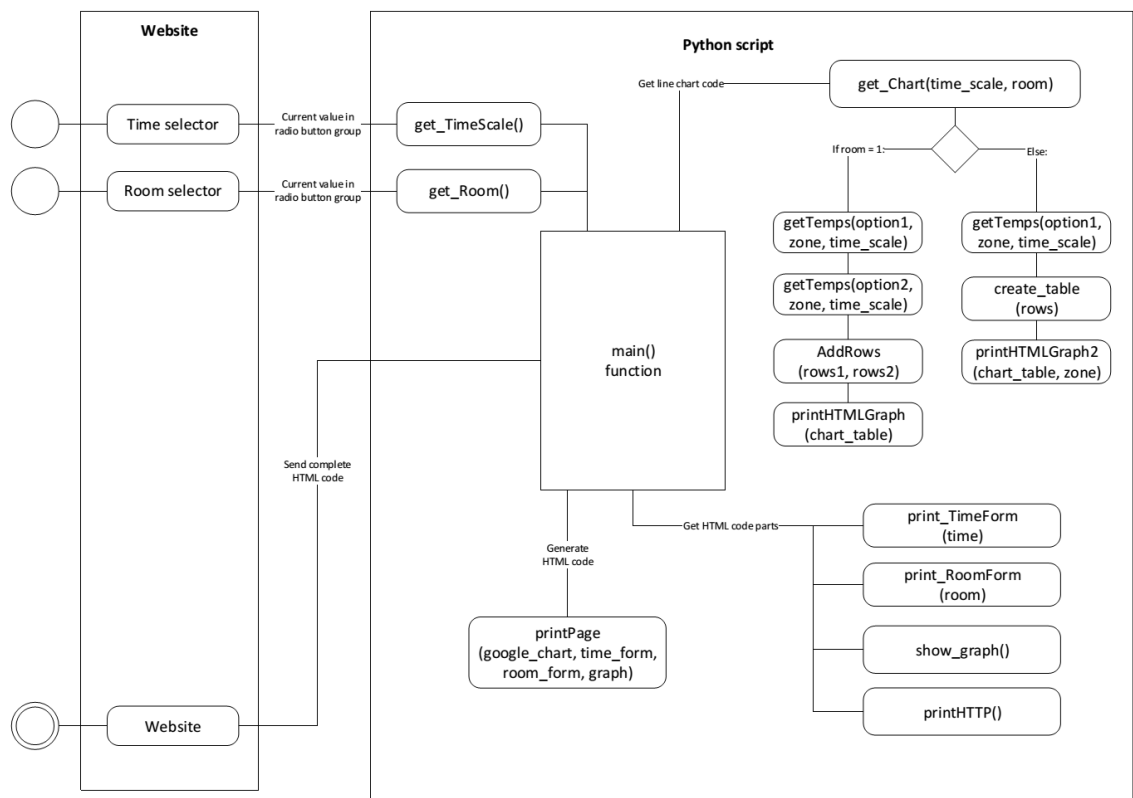


Figure 14. Activity diagram of the website

The activity diagram (Figure 14) shows how the main-function gathers the information and creates the data-table based on the criteria. The main-function updates the HTML forms, creates a div element for the chart, and finally all the information gathered in the main-function is fed into the PrintPage-function. The PrintPage-function literally prints the new HTML page. This page is then showed for the website visitor who is observing the temperature line chart.

3.9 Optimization

Optimization is the final part of the bachelor work. It aims to improve the efficiency of the actual programs and to shorten the website's loading time. One of the major optimizations is to define how many data points are presented in the temperature line chart. At first the temperature was measured every 15 minutes. This produced in total 672 data points in a time scale of the last seven days. (Formula 3)

Defining the number of data points:

$$\frac{M * H * D_{time\ scale}}{T_{interval}} = \#\ of\ data\ points \quad (3)$$

$$\frac{60minutes * 24hours * 7\ days}{15\ minutes} = 672\ data\ points \quad (4)$$

Where

- M Minutes in an hour
- H Hours in a day
- $D_{time\ scale}$ Observing time scale
- $T_{interval}$ How often the script is performed

A fast way to shorten the website's loading time in this situation is to lower the number of data points by changing the time interval of temperature measurements in Cron. Increasing the time interval to 30 minutes halves the data points to 336.

Another possibility to reduce the data points without changing the measuring interval would be calculating the average temperatures in a certain time and using the average values in the line chart's data table.

4 SUMMARY

The outcome of the thesis was a completely automated home surveillance system for a low purchasing price. The capability of the Raspberry Pi was astonishing. It has potential for many practical tasks. However, the implementation requires a lot of knowledge and research before it can be exploited. Raspberry Pi is a capable product for a hobbyists and people who are interested in computer science. However, basic knowledge of programming is helpful.

The thesis meets well the requirements which were given. The surveillance system turned out to be reliable, easily accessible and, what was most important, it was also user friendly. The graphics of the temperature data exceeded all expectations.

Data logging with 1-Wire devices is made quite simple but presenting the gathered data in a nice form can be a time consuming and complex task. The home surveillance system mainly focused on temperatures but combining more elements, for instance observing power consumption, would create a complete surveillance system for a household.

This project was interesting in many ways and there are many features that could still be improved. Taking the optimization of the graphical user interface even further would fasten the website and it would require less disk space which, in this project, was limited to the size of a micro-SD card. Because of the tight schedule the heating element does not have visualization on the website.

In the future the household measurements will become more popular, and therefore, these kinds of automation applications might grow in popularity.

REFERENCES

Kuutti, W. & Rantala A. 2007. Linux. Jyväskylä: WSOY.

McManus, S. & Cook, M. 2013. Raspberry Pi For Dummies. Hoboken: John Wiley & Sons, Inc.

Monk, S. 2013. Programming the Raspberry Pi: Getting Started with Python. USA: The McGraw-Hill Companies.

Upton, E. & Halfacree G. 2012. Raspberry Pi User Guide. Chichester: John Wiley & Sons, Inc.

Adafruit. 2015. GPIO Port. [WWW-article]. Adafruit Industries. [Ref. 25.3.2015]. Available: <https://learn.adafruit.com/introducing-the-raspberry-pi-model-b-plus-plus-differences-vs-model-b/gpio-port>

Computer Hope. No date available. Linux and Unix crontab command. [WWW-article]. Computer Hope. [Ref. 16.3.2015]. Available: <http://www.computerhope.com/unix/ucrontab.htm>

DS18S20 Datasheet. 2015. [PDF-document]. Maxim Integrated. [Ref. 25.3.2015]. Available: <http://datasheets.maximintegrated.com/en/ds/DS18S20.pdf>

Element14. No date available. Raspberry Pi Model B+. [Website]. Element14 Community. [Ref. 22.2.2015]. Available: <http://www.element14.com/community/community/raspberry-pi/raspberry-pi-bplus?ICID=rpi2-comp-chart>

Kirk, M. 2012. Raspberry Pi Temperature Sensor Tutorial. [WWW-article]. University Of Cambridge, Computer Laboratory. [Ref. 22.2.2015]. Available: <https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/temperature>

Linke, B. 2008. Overview of 1-Wire Technology and Its Use. [WWW-document]. Maxim Integrated. [Ref. 10.2.2015]. Available: <http://www.maximintegrated.com/en/app-notes/index.mvp/id/1796>

Maxim Integrated. No date available. 1-Wire Tutorial. [Video]. Maxim Integrated. [Ref. 15.3.2015]. Available: <http://www.maximintegrated.com/en/products/1-wire/flash/overview/>

Raspberry Pi Foundation. No date available. FAQ. [Website]. Raspberry Pi Foundation. [Ref. 25.3.2015]. Available: <http://www.raspberrypi.org/help/faqs/>

Raspberry Pi NoIR Camera Board. No date available. [Website]. Adafruit Industries. [Ref. 25.3.2015]. Available: <https://www.adafruit.com/products/1567>

Raspbian. No date available. FAQ. [Website]. Raspbian. [Ref. 15.2.2015]. Available: www.raspbian.org

RaspiCam Documentation. 2013. [PDF-document]. Raspberry Pi Foundation. [Ref. 15.3.2015]. Available: <http://www.raspberrypi.org/wp-content/uploads/2013/07/RaspiCam-Documentation.pdf>

SQLite. No date available. About SQLite. [Website]. SQLite. [Ref. 18.2.2015]. Available: www.sqlite.org/about.html

The Apache2 project. No date available. What is the Apache HTTP Server Project?. [Website]. The Apache Software Foundation. [Ref. 19.2.2015]. Available: <http://httpd.apache.org>

The Geek Stuff. 2009. Linux Crontab: 15 Awesome Cron Job Examples. [WWW-article]. The Geek Stuff. [Ref. 16.3.2015]. Available: <http://www.thegeekstuff.com/2009/06/15-practical-crontab-examples>

The Linux Kernel Archives. 2013. [Website]. The Linux Kernel Organization. [Ref. 26.3.2015]. Available: <https://www.kernel.org>

APPENDICES

APPENDIX 1. Frequently used commands

APPENDIX 2. Activity diagram of the website

APPENDIX 3. Monitor.py program

APPENDIX 4. Heating.py program

APPENDIX 5. Camera.py program

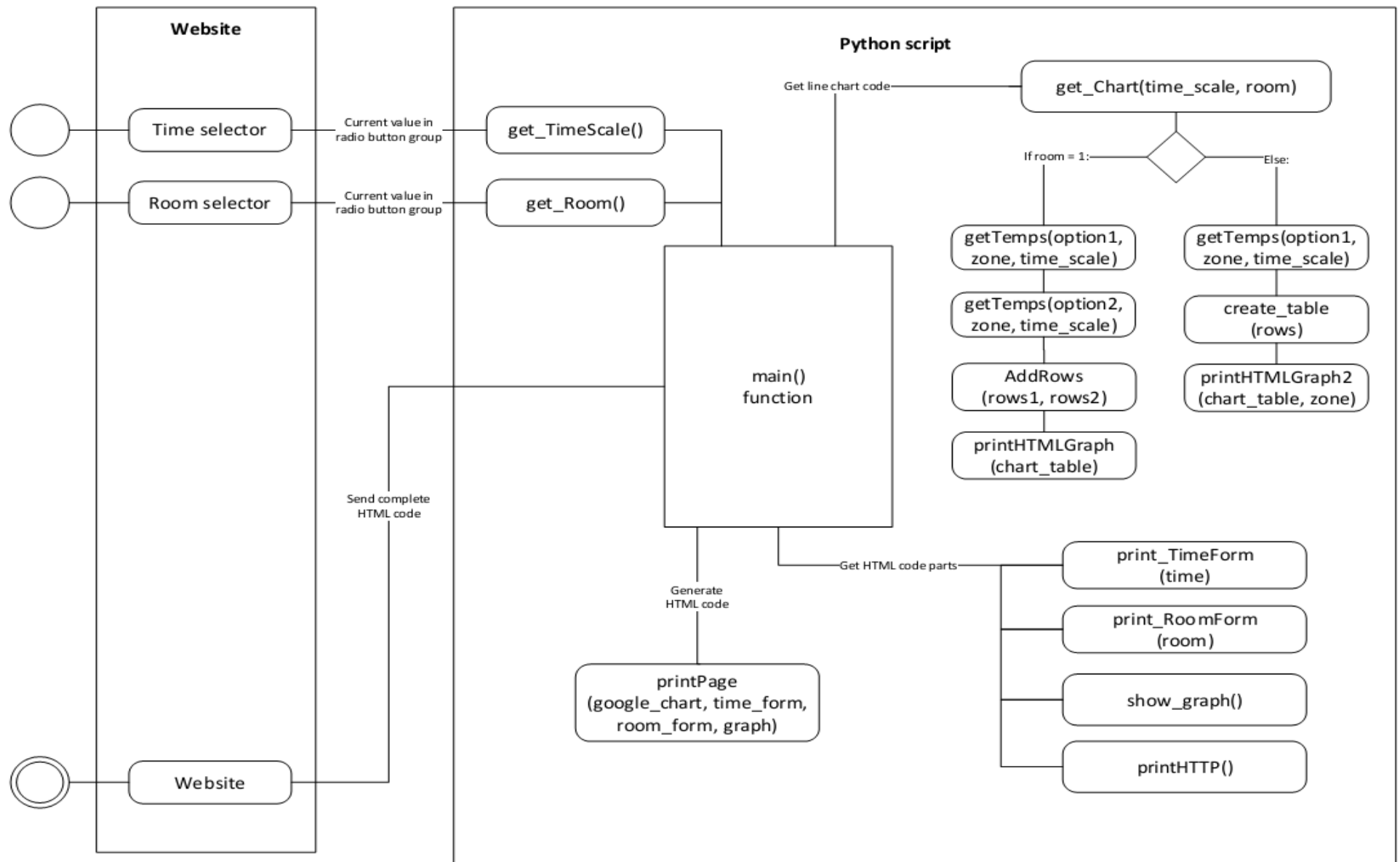
APPENDIX 6. MinMax.py program

APPENDIX 7. Selector.py program

APPENDIX 1. Frequently used commands

sudo	Gives the root privileges.
raspi-config	Enters to the Raspi configuration menu.
startx	Starts the graphical environment.
reboot	Reboots the system.
shutdown	Shutdown the system.
cd	Change a directory.
cp	Copy the file.
rm	Remove the file.
mkdir	Create a directory.
rmdir	Remove a directory.
mv	Change the name of directory.
ls	Short listing of directory contents.
chown	Change the file owner and group.
chmod	Change the file access permissions.
ifconfig	Query and configuration of TCP/IP network settings.
cat	Reads the content of a file.
crontab	Opens the cron table.

APPENDIX 2. Activity diagram of the website



APPENDIX 3. Monitor.py program

```
#!/usr/bin/env python

import sqlite3
import cgitb

cgitb.enable()

#Rooms where the sensors are:
sensor1_room="Kitchen"
sensor2_room="Garage"

#Sensors ID-numbers:
sensor1_ID="/sys/bus/w1/devices/10-000802e614ec/w1_slave"
sensor2_ID="/sys/bus/w1/devices/10-000802e67b6f/w1_slave"

#Database name
dbname='/var/www/temperatures.db'

def getTemperature(ID, sensor_room):
    tfile=open(ID)
    text=tfile.read()
    tfile.close()

    #Split the text with new lines (\n) and select the second line
    secondline= text.split("\n")[1]
    #Split the line into words, referring to the spaces and select the 10th word
    temperaturedata= secondline.split(" ")[9]
    #The first two characters are "t=", so get rid of those and convert the temperature from
    a string to a number.
    temperature=float(temperaturedata[2:])
    #Put the decimal point in the right place and display the temperature
    sensor_data = temperature/1000
    #Round the result to the 2decimals
    sensor_data = round(sensor_data, 2)

    #Insert temperature to database:
    #Connect to the temperature database
    db=sqlite3.connect(dbname)
    cur = db.cursor()

    cur.execute("INSERT INTO temperatures VALUES(date('now','localtime'),
    time('now','localtime'), (?:), (?:));", (sensor_room, sensor_data,))
    db.commit()
    db.close()

def main():
    getTemperature(sensor1_ID, sensor1_room)
    getTemperature(sensor2_ID, sensor2_room)

main()
```

APPENDIX 4. Heating.py program

```
#!/usr/bin/env python

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM) #Ribbon cable mode

#Set GPIO pins to outputs
GPIO.setup(17, GPIO.OUT)
GPIO.setup(18, GPIO.OUT)

#Sensor ID-number:
sensor_ID="/sys/bus/w1/devices/10-000802e614ec/w1_slave"

#Limits for the heating:
temp_limit_on=float(24)    #22 degrees
temp_limit_off=float(26)   #25 degrees

def getTemperature(ID):
    tfile=open(ID)
    text=tfile.read()
    tfile.close()

    #Split the text with new lines (\n) and select the second line
    secondline= text.split("\n")[1]
    #Split the line into words, referring to the spaces and select the 10th word
    temperaturedata= secondline.split(" ")[9]
    #The first two characters are "t=", so get rid of those and convert the temperature from
    a string to a number.
    temperature=float(temperaturedata[2:])
    #Put the decimal point in the right place and display the temperature
    sensor_data = temperature/1000
    round(sensor_data,2)
    return sensor_data
```

```

def main():
    starting_temp=getTemperature(sensor_ID)
    starting_temp=float(starting_temp)

    if starting_temp <= temp_limit_on:
        GPIO.output(17, GPIO.LOW)
        #Start heating
        GPIO.output(18, GPIO.HIGH)
        temp=float(0)    #Temporary definition for the temp

        while(temp < temp_limit_off):
            time.sleep(10)    #Wait 120s
            temp=getTemperature(sensor_ID) #check room temp
            print ("Heating on, temperature currently: %s" % temp)

            #Turn the heating off after the temperature is raised above the limit
            GPIO.output(18, GPIO.LOW)
            GPIO.output(17, GPIO.HIGH)
            print ("Heating off, temperature currently: %s" % temp)

        else:
            GPIO.output(17, GPIO.HIGH)
            print "Room temperature is above the heating limit"

main()

```

APPENDIX 5. Camera.py program

```
#!/usr/bin/env python

import cgitb
import datetime
import picamera
import time

cgitb.enable()

def takePicture():
    location='/var/www/camera/'
    date=datetime.datetime.now()    #Get current date
    file_name=date.strftime("%Y-%m-%d %H%M") #Format the string

    #configuration for the pictures
    camera = picamera.PiCamera()
    camera.resolution = (1280,720)
    camera.start_preview()

    #Camera warm up time
    time.sleep(10)
    # Capture the picture and saved it with the current date
    camera.capture("%s%s.jpg" % (location,file_name), quality=75)
    camera.stop_preview()
    camera.close()

takePicture()
```


APPENDIX 6. MinMax.py program

```
#!/usr/bin/env python

import sqlite3
import datetime
import cgitb

cgitb.enable()

#Database name where to search
dbname='/var/www/temperatures.db'

def getCurrentTemperature():
    #Connect to the temperatures database
    db=sqlite3.connect(dbname)
    cur = db.cursor()
    cur.execute("SELECT Temperature FROM temperatures WHERE rowid=(SELECT max(rowid) FROM
    temperatures WHERE Room='Kitchen')");
    current_temp=cur.fetchone()
    return current_temp

def getMaxTemperature():
    #Connect to the temperatures database
    db=sqlite3.connect(dbname)
    cur = db.cursor()
    cur.execute("SELECT MAX(Temperature) FROM temperatures WHERE Day=date('now')");
    max_temp=cur.fetchone()
    return max_temp

def getMinTemperature():
    #Connect to the temperatures database
    db=sqlite3.connect(dbname)
    cur = db.cursor()
    cur.execute("SELECT MIN(Temperature) FROM temperatures WHERE Day=date('now')");
    min_temp=cur.fetchone()
    return min_temp

def getLastUpdate():
    #Connect to the temperatures database
    db=sqlite3.connect(dbname)
    cur = db.cursor()
    cur.execute("SELECT * FROM temperatures WHERE rowid=(SELECT max(rowid) FROM
    temperatures)");
    last_row=cur.fetchone()
    date=last_row[0]
    time=last_row[1]
    last_update="%s, %s" % (date,time))
    return last_update

def getDate():
    day=datetime.datetime.now()
    date=day.strftime("%d.%m.%Y")
    return date
```

```

def main():
    date=getDate()
    current_temp=getCurrentTemperature()
    max_temp=getMaxTemperature()
    min_temp=getMinTemperature()
    last_update=getLastUpdate()

    print "Content-type: text/html\n\n"
    print """<link rel="stylesheet" type="text/css" href="/autumn05.css" />"""
    print """<div class="leftcolbox">
        <div class="leftcolboxtop"></div>
        <h1>Feed Box</h1>"""

    print "<p>Today is: %s</p>" % (date)
    print "<p>Last update:<br>%s</p>" % (last_update)
    print "<p>Current temp in kitchen:<br>%s &deg;C</p>" % (current_temp)
    print "<p>Max temp: %s &deg;C</p>" % (max_temp)
    print "<p>Min temp: %s &deg;C</p>" % (min_temp)

main()

```

APPENDIX 7. Selector.py program

```
#!/usr/bin/env python

import sqlite3
import cgi
import cgi

cgi.enable()

#Database name where to search
dbname='/var/www/temperatures.db'

def printPage(google_chart, time_form, room_form, graph):
    page= """
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "
http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Raspi Project</title>

    <link rel="stylesheet" type="text/css" href="/autumn052.css" />

</head>

<body>

%s

<div id="container">
<div id="navarea">
<ul id="nav">

    <li><a href="/index.html" title="home">Home</a></li>

    <li><a href="#" title="about">Monitor</a></li>

    <li><a href="/camera/" title="archive">Camera</a></li>

    <li><a href="#" title="contact">FREE</a></li>

    <li><a href="#" title="links">FREE</a></li>

</ul>

</div>

<div id="hdr"> <span id="sitetitle">Raspberry
Pi</span><br />

<span id="subtitle">project 2015</span></div>

<div id="lftcol">
<div class="leftcolbox">
<div class="leftcolboxtop"></div>
<h2>Time selector:</h2>
%s
```

```

</div>

<div class="leftcolbox">
<div class="leftcolboxtop"></div>
<h2>Room selector:</h2>
%s
</div>

</div>

<div id="maincol">
<div class="rule">
<h1>Welcome to Raspberry Pi project website</h1>
</div>
%s
<br>
<br>
<p> This webpage contains temperature monitoring data and
pictures taken with the Raspberry Pi's PiNoIR camera module. In
Monitoring section visitor can observe the temperatures within certain
time range.&nbsp;</p>

<p></p>

<p>The camera section contains pictures taken with Pi NoIR camera
module by every 15 minutes. New pictures appears to the web-page
automatically.</p>

<p></p>

<p>This web-page is based on open source template from
&nbsp;<span style="font-weight: bold;"><a id="bttmnav"></a>

```

```

def printTimeForm(time_scale):
    form0= '<form name="time_selector" action="/cgi-bin/Selector.py" method="get">'

    if time_scale == "1":
        form1= '<p><input type="radio" name="time_selector" value="1" checked="checked"
        onclick="this.form.submit();"> 24 Hours</p>'
    else:
        form1= '<p><input type="radio" name="time_selector" value="1"
        onclick="this.form.submit();"> 24 Hours</p>'
    if time_scale == "2":
        form3= '<p><input type="radio" name="time_selector" value="2" checked="checked"
        onclick="this.form.submit();"> 48 Hours</p>'
    else:
        form3= '<p><input type="radio" name="time_selector" value="2"
        onclick="this.form.submit();"> 48 Hours</p>'
    if time_scale == "3":
        form4= '<p><input type="radio" name="time_selector" value="3" checked="checked"
        onclick="this.form.submit();"> 5 Days</p>'
    else:
        form4= '<p><input type="radio" name="time_selector" value="3"
        onclick="this.form.submit();"> 5 Days</p>'
    if time_scale == "4":
        form5= '<p><input type="radio" name="time_selector" value="4" checked="checked"
        onclick="this.form.submit();"> 7 Days</p>'
    else:
        form5= '<p><input type="radio" name="time_selector" value="4"
        onclick="this.form.submit();"> 7 Days</p>'

    form=form0+form1+form3+form4+form5
    return form

def printRoomForm(room_option):
    form0= '<form name="room_selector" action="/cgi-bin/Selector.py" method="get">'

    if room_option == "1":
        form1= '<p><input type="radio" name="room_selector" value="1" checked="checked"
        onclick="this.form.submit();"> All rooms</p>'
    else:
        form1= '<p><input type="radio" name="room_selector" value="1"
        onclick="this.form.submit();"> All rooms</p>'
    if room_option == "2":
        form2= '<p><input type="radio" name="room_selector" value="2" checked="checked"
        onclick="this.form.submit();"> Kitchen</p>'
    else:
        form2= '<p><input type="radio" name="room_selector" value="2"
        onclick="this.form.submit();"> Kitchen</p>'
    if room_option == "3":
        form3= '<p><input type="radio" name="room_selector" value="3" checked="checked"
        onclick="this.form.submit();"> Garage</p>'
    else:
        form3= '<p><input type="radio" name="room_selector" value="3"
        onclick="this.form.submit();"> Garage</p>'

    form4= '</form>'

    form=form0+form1+form2+form3+form4
    return form

```

```

def printHTTP():
    print "Content-type: text/html\n\n"

def get_TimeScale():
    form=cgi.FieldStorage()
    if "time_selector" in form:
        option = form["time_selector"].value
        return option
    else:
        return None

def get_Room():
    form=cgi.FieldStorage()
    if "room_selector" in form:
        option = form["room_selector"].value
        return option
    else:
        return None

def edit_TimeScale(time_scale):
    if time_scale == "1":
        time_scale="-1 Days" #-12Hours
        return time_scale

    elif time_scale == "2":
        time_scale="-2 Days" #-48Hours
        return time_scale

    elif time_scale == "3": #-5 Days
        time_scale="-5 Days"
        return time_scale

    elif time_scale == "4": #-7 Days
        time_scale="-7 Days"
        return time_scale

def getTemps(option, zone, time_scale):

    time_scale=edit_TimeScale(time_scale)

    if option == 1:

        #Connect to the temperatures database
        db=sqlite3.connect(dbname)
        cur = db.cursor()
        cur.execute("SELECT * FROM temperatures WHERE Room=? AND Day BETWEEN
datetime('now','localtime', ?) AND datetime('now', 'localtime')", (zone, time_scale,)
        )
        rows=cur.fetchall()
        db.close()
        return rows

    elif option == 2:
        #Connect to the temperatures database
        db=sqlite3.connect(dbname)
        cur = db.cursor()
        cur.execute("SELECT Temperature FROM temperatures WHERE Room=? AND Day BETWEEN
datetime('now','localtime', ?) AND datetime('now', 'localtime')", (zone, time_scale,)
        )
        rows=cur.fetchall()
        db.close()
        return rows

    else:
        return None

```

```

def addRows(rows1, rows2): #Add two queries together
    chart_table=""
    x=int(0)
    for row in rows1[:-1]:
        rowstr="['{0} {1}', {2}, {3}]\n".format(row[0],row[1],row[3],rows2[x])
        chart_table+=rowstr
        x=x+1

    row=rows1[-1]
    rowstr="['{0} {1}', {2}, {3}]\n".format(row[0],row[1],row[3],rows2[x])
    chart_table+=rowstr
    chart_table=chart_table.replace("(","")
    chart_table=chart_table.replace(")","")

    return chart_table

def create_table(rows):
    chart_table=""
    for row in rows[:-1]:
        rowstr="['{0} {1}', {2}]\n".format(str(row[0]),str(row[1]),str(row[3]))
        chart_table+=rowstr

    row=rows[-1]
    rowstr="['{0} {1}', {2}]\n".format(str(row[0]),str(row[1]),str(row[3]))
    chart_table+=rowstr

    return chart_table

def printHTMLGraph(chart_table):
    ## # google chart snippet
    chart_code="""
    <script type="text/javascript" src="https://www.google.com/jsapi"></script>
    <script type="text/javascript">

    google.load("visualization", "1", {packages:["corechart"]});
    google.setOnLoadCallback(drawChart);

    function drawChart() {
        var data = google.visualization.arrayToDataTable([
            ['Time', 'Garage','Kitchen'],
    %s
            ]);
        var options = {
            title: 'Temperatures in household',
            hAxis:{
                title:'Time',

            },
            vAxis:{
                title:'Degrees Celsius'

            },
            colors:['#a52714', '#097138']

        };
        var chart = new google.visualization.LineChart(document.getElementById('chart_div'));
        chart.draw(data, options);
    }
    </script>"""

    return (chart_code % (chart_table))

```

```

def printHTML2Graph(chart_table, zone):
    ##      # google chart snippet
    chart_code="""
    <script type="text/javascript" src="https://www.google.com/jsapi"></script>
    <script type="text/javascript">

    google.load("visualization", "1", {packages:["corechart"]});
    google.setOnLoadCallback(drawChart);

    function drawChart() {
        var data = google.visualization.arrayToDataTable([
            ['Time', '%s'],
%s
        ]);
        var options = {
            title: 'Temperatures in household',
            hAxis:{
                title:'Time',

            },
            vAxis:{
                title:'Degrees Celsius'

            },
            colors:['#a52714', '#097138']

        };
        var chart = new google.visualization.LineChart(document.getElementById('chart_div'));
        chart.draw(data, options);
    }
    </script>"""

    return (chart_code % (zone, chart_table))

def show_graph():
    line1= """<div id="chart_div" style="width: 900px; height: 700px;"></div>"""
    return line1

def get_Chart(time_scale, room):

    if room == "1":
        zone="Garage"
        zone2="Kitchen"
        rows1=getTemps(1, zone, time_scale)
        rows2=getTemps(2, zone2, time_scale)
        chart_table=addRows(rows1,rows2)
        google_chart=printHTMLGraph(chart_table)
        return google_chart

    else:
        if room == "2":
            zone="Kitchen"
        elif room == "3":
            zone="Garage"

        rows=getTemps(1, zone, time_scale)
        chart_table=create_table(rows)
        google_chart=printHTML2Graph(chart_table, zone)
        return google_chart

```



```
def main():
    time_scale=get_TimeScale()
    room=get_Room()

    if time_scale is None:
        time_scale=str(1)

    if room is None:
        room=str(1)

    javascript=get_Chart(time_scale, room)

    time_form=printTimeForm(time_scale)
    room_form=printRoomForm(room)
    graph=show_graph()
    printHTTP()

    printPage(javascript, time_form, room_form, graph)

main()
```