Smart Thermostat

S. Eppe, S. Vansummeren, A. Vermeir

Version 1.0 - October 7, 2015

1 Introduction

The central heating thermostat is one of the oldest and simplest feedback controllers. In its original form, it measures the current room temperature. If this temperature is below a desired target temperature (called the **set-point temperature**), then the thermostat switches on a central boiler. The boiler heats water that is transmitted to radiators throughout the building. The radiators, in turn, heat the rooms that they are located in. When the thermostat notices that the set-point temperature is reached, it switches the boiler off again. Of course, care needs to be taken to avoid overshooting the set-point temperature and to avoid pendeling of the boiler.[1] In this project, you will be required to build a **multizone** and **smart** thermostat.

Multizone ... A traditional thermostat is installed in a central room whose temperature is supposed to be representative of the entire building. This implies that when the target temperature is reached for that room, the boiler will switch off. However, neighboring rooms that have not yet reached the target temperature will hence remain below this temperature. With the use of wireless communication, this inconvenience can be overcome. Indeed: subdivide a building in multiple thermal zones, and equip each zone with a "thin thermostat": a microcontroller that is equipped with a temperature and presence sensor, and that has the ability to open or close a thermostatic valve of the radiator in the zone. The thin thermostats wirelessly communicate their measurements to the central thermostat, which controls and drives the boiler. In zones where the target temperature is reached the valves are closed to avoid overheating. But only when each zone has reached its target temperature (or when the controller sees that with the current radiator water temperature the target temperature can easily be reached), the boiler is switched off. Figure 1 illustrates this concept.

... and smart. More recent thermostats take other parameters, such as the outdoor temperature and the heating inertia of a building into account to significantly reduce heating energy consumption. The newest generation of thermostats, the so-called "smart thermostats" take this approach even further and provide facilities such as the following.

- Operation of the thermostat at a distance through the World Wide Web.
- Sensing whether building occupants are home (either through IR sensors, or through geo-fencing of smart-phones) and (1) automatically lowering the target temperature if the building is un-occupied; (2) pre-heating the building when it is detected that occupants are likely to be home soon (e.g. by explicit user notification; through geo-fencing or because of an automatically-learned schedule).

[1] Pendeling is the activity where the boiler continuously switches off for a short period of time, and then on again for a short period of time.

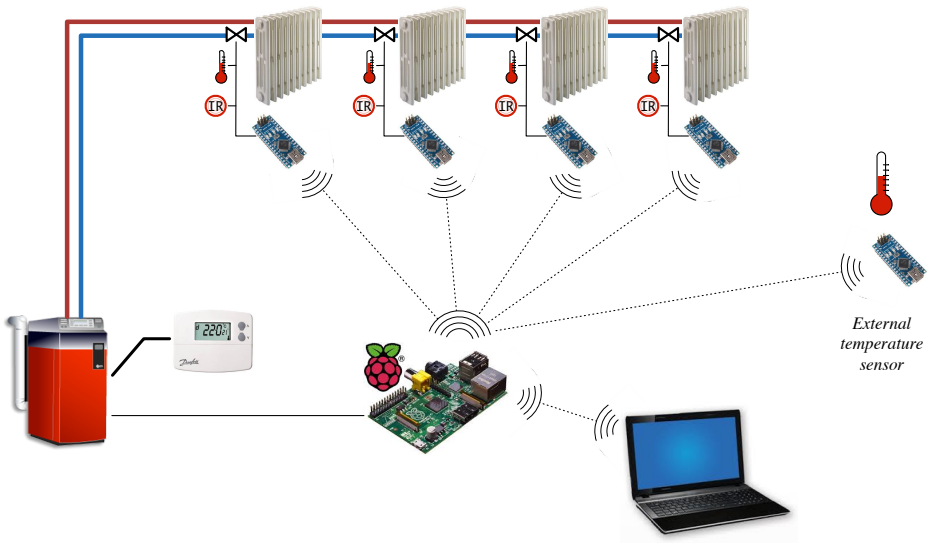


Figure 1: General setup of a multi-zone thermostat.

The interested reader is referred to Appendix A for an overview of currently commercially-available smart thermostats.[2]

2 Objectives and requirements

2.1 Prototypes

The objective in this project is to design and implement a multi-zone and smart thermostat system. There are two technical components whose prototypes need to be developed.

- The **thin thermostats**. Each thin thermostat has two sensors and one actuator:
  - A temperature sensor to measure zone temperature.
  - An infrared sensor to detect whether there are people in the zone.
  - A valve actuator, to open or close a thermostatic valve.

  Sensor readings are communicated wirelessly to the central thermostat unit, actuation commands are received wirelessly from the central thermostat unit. To simplify the project, only a single thin thermostat prototype needs to be realized. Also, the thermostatic valve will be connected to a power resistor that simulates a central heating system (boiler, hot water transfer pipes, radiators). This simplified setup is illustrated in Figure 2.
- The **central thermostat**. It communicates wirelessly with the thin thermostats. It aggregates their sensor readings and contains the control algorithm, which is responsible for two tasks: (1) actuate the thermal valves in those zones where it is determined that heating is required; and (2) determining in which thermal zones heating is (or will soon be) required. We detail each task next.
  1. To actuate the thermal valves in the zones where heating is required, the algorithm first needs to determine, based on the current zone temperature; the desired temperature; and the outdoor temperature, the level by which the valve needs to be actuated (from

[2]It is worth remarking that in 2014 Google acquired Nest Labs, producer of one of the first smart thermostats, for more than 3 billion dollars.

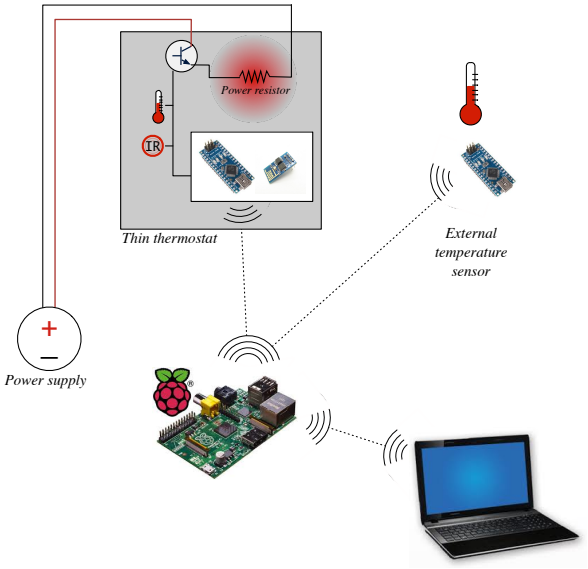


Figure 2: Simplified setup

0%, i.e., fully closed, to 100%, i.e., fully opened). It then sends this level to the respective thin thermostats, which performs the actual acutation. You will need to develop a suitable model that allows to control the desired actuation level. The algorithm itself should avoid (significant) overshoot (i.e., heating the room to above target temperature) and pendeling (continuous on-and-off switching).

2. The control algorithm is also responsible for detecting that heating is required. Here, we discern two levels.

   Reactive detection The controller simply uses the presence sensor: heating is required in zone X if a person is present in zone X and the current temperature in X is below the set-point temperature.

   Smart detection This makes the thermostat a "smart thermostat". Using the World Wide Web, the user can indicate to the central unit that she will be home in x minutes and that by that time, the set-point temperature in zones Y and Z should be reached. Using this information the central unit can anticipate, and determine its optimal heating plan.

   During the project, you will first implement a control algorithm based on reactive detection. Then, in a second phase, you will build a control algorithm based on smart detection.

For debugging purposes as well as evaluation of your control algorithm, it is required that the state of the entire system can be externally monitored. To this end, the central unit should provide a dashboard that can be displayed in a web browser.

Note that there is one extra device in this scheme that communicates wit the central thermostat: the **outdoor thermometer**. It only has a temperature sensor, and measures the outdoor temperature.

2.2 Communication protocol

The central and thin thermostats will communicate over WIFI using the HTTP-based [8] communication protocol specified in Appendix B. You will need to implement this communication

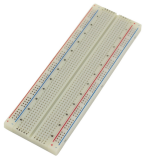
(a) Raspberry Pi 2 Model B

(b) Arduino Nano

(c) ESP8266 wifi module

(d) Presence sensor

(e) Thermistor



(f) Breadboard

Figure 3: Implementation hardware.

protocol yourself. Strict adherence to the protocol is required.

2.3 Supplied hardware

A wide variety of hardware platforms and microcontroller chips are conceivable to implement the central and thin thermostats. In our experience, however, these can be time-consuming to correctly set up. To prevent that time is lost in solving configuration errors, we therefore provide you with the implementation hardware pre-selected and pre-configured (see Figure 3). You hence will not need to buy hardware for this project.

- The central thermostat unit needs to be implemented on a Raspberry Pi 2 Model B [6] (a "Pi" for short). The Raspberry Pi is a low cost, credit-card sized computer. Despite its small size it is a full computer with main memory, CPU, and a mini-SD card that acts as its hard-drive. Your Pi runs the Linux operating system; has the Python programming language pre-installed; and is equipped with a working WI-FI USB dongle that will allow it to communicate with the thin thermostats. The central control algorithm must be implemented in python on the Pi.
- The thin thermostat needs to be implemented using an Arduino Nano microcontroller [2]. For wireless communication, an esp8266 WIFI module is provided. (cf. Figure 3b). An Arduino is programmed using a language called Arduino Programming Language [3] (a variant of C++). Familiarizing yourself with the Arduino Programming Language as well as reading signals from the sensors and sending an output signal on an Arduino is part of the project.
- Finally, you are provided with wires and modules kit needed for the thin thermostat prototype. A thermistor and infrared sensor are included to implement the temperature and presence sensors, respectively. The power resistor is not provided; this is part of the physical environment (see below). To actuate the power (heating) resistor, the arduino will need to output a control signal on one its I/O pins.

All provided equipment needs to be returned after the final project evaluation.

2.4 Physical environment

In addition to the above-mentioned hardware, you will also be provided with a **physical environment**. The **physical environment** is a box (Figure 4) in which your thin thermostat can

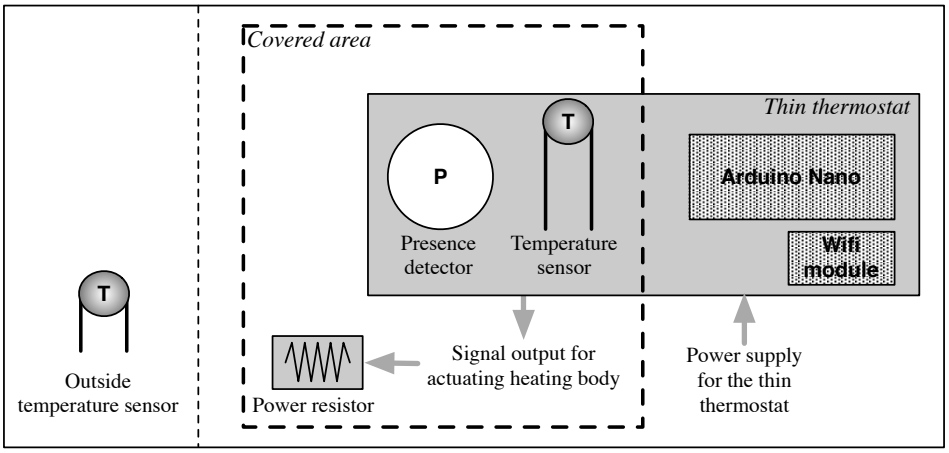


Figure 4: Schematic representation of the physical validation environment.

be placed and tested. This environment has two aims: 1) provide a simple heating mechanism (compared to hot water piping) that works in a narrow volume and that will allow you to test your thin thermostat in a relatively controlled environment and on small time spans; 2) provide an additional sensor that will wirelessly report the outside temperature to the central unit.

Practically, the physical validation environment is equipped with a power supply to drive the thin thermostat. It is also equipped with the power resistor that your thin thermostat can actuate through a control signal to provoke actual heating of the validation environment. In addition, it will also play the role of the external temperature sensor, that will register to the central unit and report the temperature outside the "system", i.e., the closed volume where the presence detector and temperature sensor of your thin thermostat are located. Note that, while your thin thermostat will be connected to the power resistor and (possibly) to the power supply, it will not be connected directly to the outside temperature sensor.

Each group will be provided with a complete physical environment approximatively 2 weeks after the start of the project. The aim is to first let you get acquainted with the Arduino Nano board, the wifi module and the sensors (PIR and thermistor).

2.5 Supplied Software

Example/library code that illustrates how you can use the ESP8266 wifi module on the arduino, as well as program a web server on the raspberry pi is provided in the GIT repository[3] available at:

`http://wit-projects.ulb.ac.be/rhodecode/TRAN-H-201/2015-2016/Supplied-Software`.

This repository also contains a program that you can use to test that the central unit implements the communication protocol correctly.

2.6 Expected results

In summary, the expected results include:

[3] `http://git-scm.com/documentation`

– Design of the control algorithm.
– Implementation (programming) of control algorithm (in python) and implementation of thin thermostats (assembly and programming); including implementation of their communication protocol described in Appendix B.
– Quality measures. You are asked to develop one or more criteria that permit to quantitatively measure the degree to which you succeed. These criteria should help you evaluate the different solutions that you consider and/or develop within the project. Possible criteria are: measures for overshoot, measures for the speed of reaching the set-point temperature, measures for energy consumed, ...
– Testing strategy. Develop a testing strategy to ensure that the different components that you develop work as expect, both at the unit level, and integrated system level. Be sure to report on this strategy and the test results in your report.
– Group work. The final note takes into account the quality of the group work; the resolution of problems that can be caused by such group work; as well as the discovery and use of tools that facilitate the group work.
– Building instructions of the thin thermostat. These instructions need to fully specify the physical interface of the thin thermostat, so that others can use it. In particular, it needs to include a wiring plan specifying how to hook the thin thermostat to the sensors, and to the heating resistor.

It is important to stress the following points:

– While you are only required to realize a single thin thermostat prototype, your central thermostat unit needs to be able to control multiple thin thermostats.
– The structure of your control algorithm should allow it to be used in physical validation environments with different thermal characteristics (e.g., an environment that is heavily insulated). In other words: you should not hard-code in the control algorithm the thermal response characteristics of the actual physical environment that is provided to you. In the final report, you must include a description of how your system can adapt to another environment with its specific thermal characteristics.
– While it is possible to model a heating system as a system of equations, most notably by means of the so-called heat equation, we firmly recommend that you consider the system (i.e., the zone that is to be heated) as a black box: the heat equation is a parabolic partial differential equation whose solution, even numerical solution, is outside the scope of this project.

3 Deliverables

For the mid-term evaluation. For the mid-term evaluation you will have to produce an intermediary report (including a detailed bibliography). For evaluation by the other members of the jury, your intermediary report needs to include a page listing the group members (name, student ID, email, picture). You will also orally present the current state of the project, including the state of your prototypes.

For the final evaluation. For the final evaluation, the deliverables are:

– The final report (including a detailed bibliography, and page with group members);
– the functional prototypes (central and thin thermostats);
– the code produced for both prototypes;
– the as-built plan of the thin thermostat; and

– an oral presentation of the project in front of the jury.

You also need to hand in the implementation hardware that was provided to you.

Delivery method. You will have to create a GIT repository[4] in the `TRAN-H-201/2015-2016` repository group at `http://wit-projects.ulb.ac.be/rhodecode/TRAN-H-201/2015-2016` to submit your report; code; and as-built plan to the jury. The username and password to login to this system correspond to your ULB/VUB NetID. The repository must be named `group<X>`, where `<X>` is your group number. This repository must be made private. It is recommended that you create this repository *as soon as possible* to avoid last minute technical difficulties, and that you use it throughout the project as a version control system and synchronize your changes.

Tip. If you attempt to push a large set of changes to a GIT repository with HTTP or HTTPS, you may get an error message such as `error: RPC failed; result=22, HTTP code = 411.` This is caused by a GIT configuration default which limits certain HTTP operations to 1 megabyte. To change this limit run within your local repository

```
git config http.postBuffer *bytes*
```

where `*bytes*` is the maximum number of bytes permitted.

4 Project milestones and grading criteria

The project has the four milestones described below. Each completed milestone will be quoted on 5 points out of 20. For example, if you complete milestones 1 and 3, but not 2 and 4, you can get a score of at most 10/20. For each milestone the score that you obtain will of course depend on the prototype(s) developed but also on the manner in which you have effectively been able to work as a group; manage the project; validation your approach through pertinent quality measures; as well show your understanding of the problem and work.

Note that some of the following milestones can be developed in parallel, while others require the completion of dependent milestones. You are free to choose how you plan reaching the deliverables. However, after at most 3 weeks you will need to fix and discuss a work plan with your tutor, giving a clear description of your approach.

Milestone 1: thin thermostat

– setup of arduino; learning to program using Sketch
– wiring of arduino; interfacing with sensors and actuators
– implementation of parts of the communication protocol that are relevant for the thin thermostat;
– unit testing: are sensor readings correct? is actuation working?
– as-built plan.

Milestone 2: central thermostat with a reactive control algorithm.

– set up of the raspberry pi;
– background reading, rough modeling of control algorithm;
– design and implementation of control algorithm based on reactive detection;

[4] `http://git-scm.com/documentation`

– implementation of parts of the communication protocol that are relevant for the central thermostat part;
– unit testing of communication protocol based on software simulator included in Supplied Software .

Milestone 3: Integration and validation Builds upon the results of Milestone 1 and 2 and entails:

– complete integration of central + thin thermostats;
– full validation of the integrated system using the physical testbed;
– critical analysis and synthesis of system's performance;
– modifications to central and thin thermostats if required;

Milestone 4: central thermostat with a smart control algorithm

– modeling the anticipation behavior of the control algorithm;
– modeling related to dynamic heating (determining the optimal timepoint to start/stop heating);
– design and implementation of control algorithm based on smart detection;

References

[1] Requests: Http for humans. `http://www.python-requests.org/en/latest/`.

[2] Arduino. Getting started with arduino. `https://www.arduino.cc/en/Guide/HomePage`.

[3] Arduino. Sketch. `https://www.arduino.cc/en/Tutorial/Sketch`.

[4] Internet Assigned Numbers Authority. Media types. `http://www.iana.org/assignments/media-types/media-types.xhtml`.

[5] Bottle: Python web framework. `http://bottlepy.org/docs/dev/index.html`.

[6] Raspberry Pi Foundation. Raspberry pi - teach, learn and make with raspberry pi. `https://www.raspberrypi.org/`.

[7] Ecma-404 the json data interchange standard. `http://www.json.org`.

[8] James Marshall. Http made really easy. `http://www.jmarshall.com/easy/http/`.

A Overview of commercially-available smart thermostats

Quite a number of so-called "smart thermostats" currently already exist. The most notable are:

- the NEST termostat (`https://nest.com`) which was aquired by Google for > 3 billion dollars a few years ago.
- the Honeywell Lyric (`http://lyric.honeywell.com/`) – Honeywell has been building thermostats since the late 1800s
- the Ecobee 3 (`https://www.ecobee.com`).
- the tado (`https://www.tado.com/gb/heatingcontrol-savings`)

All of these thermostats are connected by WIFI to the internet and can be controlled from a smartphone/tablet app at a distance. In addition:

- the NEST thermostat has motion sensors and learns the habits of a user during a week-long setup phase. If it detects absense it will lower the desired temperature, if it detects presence it will increase it. If arrival is expected, it may already start increasing the temperature. Multiple NEST thermostats can communicate with each other to share motion sensor data (but have their own learned schedule).
- the Honeywell Lyric does not have a learning algorithm (it claims that most people's habits are too irregular to gain any benefit). It does, however, support geo-fencing: based on the location of paired smartphones (that are being geo-tracked) it will lower the temperature if the smartphone is outside of a certain radius, and start heating when it sees the smartphone approaching the home.
- the Ecobee 3 measures the outside temperature based on the thermostat's geo-location and up-to-date outside weather data retrieved from web services. It also has a number of in-house remote sensors that can communicate with the central thermostat. This way, if you are e.g. working in your home office but the thermostat is in the living area, you would still get heat. (The demand temperature is the average of all areas where presence is deteted.)
- the tado uses geofencing to check if people are out, checks the weather-forecast to avoid overshooting, and learns from its mistakes (feedback loop)

B Communication protocol

The central thermostat communicates with the thin thermostat and the outdoor thermometer using the protocol described in this section. Note that strict adherence to this protocol is required.

B.1 General background

All communication is done using HTTP [8]. HTTP is the same protocol as web browsers use to download web pages from the Internet. It is also used by many other applications to communicate on the World Wide Web.

HTTP is a so-called *request-reply* protocol. This means that one party, called the *http client* makes a request to the other party, which is called the *http server*.[5] The server inspects the request, sees if it can provide a meaningful answer to the request, and gives a response. In order to be able to make the request, the client needs to now the IP address (e.g., `192.168.10.1`) of the server, and the port (e.g., `8080`) that the server is inspecting for requests. The server typically runs continuously, looking if there are new requests to service.

Technically, both the request and the response are just text strings that are formatted according to particular conventions. These conventions are not important for the purpose of this project, since you should use existing libraries (e.g., [5, 1]) to make and respond to requests.

More important to understand the project's communication protocol, however, is that a request is composed of a number of parts:

- The **HTTP method**, which specifies what should be done. Table 1 lists the valid methods and what they mean.
- The **path** on which the method should be applied. For example, to retrieve the web page at `https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol`, a web browser will first resolve the host name `en.wikipedia.org` into an IP address, connect to this address at a pre-defined port (by default: port 80), and then make a `GET` request (i.e., retrieve) to path `wiki/Hypertext_Transfer_Protocol`.
- A number of **headers**. A header is a (key, value) pair where the names of certain keys are standardized. Headers allow to give to more information about how a certain request should be processed.
- The **entity body**. This is data that is sent from the client to the server. The entity body is typically empty for `GET` requests. However, for a `PUT` or `POST` request, when the client wants to send data to the server, the entity body contains the actual data.
  Technically, the entity body is just a text string. The `Content-Type` header of the request will specify how the string needs to be interpreted. For example, a `Content-Type` header with value `image/png` specifies that the entity body should be interpreted as an image in the PNG format. Or a `Content-Type` header with value `text/plain` specifies that the entity body should just be interpreted as a piece of text. The set of official content types can be found in reference [4].

The response is also composed of a number of parts.

- A **status code**. For example a status code of `200` means that the request was correctly serviced. A response code of `400` means that the client made a bad request to which the server cannot answer.

[5] In the context of a web browser, the web browser is the client and the host that it is connecting to is the http server, sometimes also called the web server.

Method	Explanation
GET	request a resource from the server (HTML page, data, . . .)
POST	send data to the server
PUT	create or update a resource on the server
DELETE	delete a resource
OPTIONS	discover what HTTP methods are supported by the given path
HEAD	similar to GET, but only retrieves headers, not entity body

Table 1: HTTP methods

– A number of **headers**, which serve the same purpose as for requests.
– The **entity body**, which contains the data sent from the server to the client (if any). Again, the `Content-Type` header specifies how the entity body should be interpreted.

B.2 The protocol

When a thin thermostat is powered-up, it needs to connect to the wireless network whose SSID is "TRANH201-GXX" (where XX is your group number), and obtain an IP address from this wireless network. Your raspberry pi needs to be switched on at this pint, since it is configured to host this wireless network. Once connected and once received an IP address, the thin thermostat needs to register itself with the central thermostat. During registration, the central thermostat acts as http server, and the thin thermostat as http client. Once registered, the central thermostat will regularly poll its registered thin thermostats to obtain sensor readings and actuate the valves. During that stage, the central thermostat acts as http client, and the thin thermostats as http server.

The registration protocol

The central thermostat is configured to always get the IP address `192.168.10.1`. The HTTP server on the central thermostat that listens for registration requests needs to be configured to listen on port `8080`.

To register, the thin thermostat connects to the central thermostat at the above-specified fixed IP address and port and makes a `PUT` request on path `/register` where the entity body contains content of type `application/json` [7] according to the following template.

```
{ "ip": "192.168.10.51", "port": 9000, "type": "thin" }
```

Here, the values mentioned in the `ip` and `port` fields need to be replaced with the IP address of the thin thermostat, and the port at which the HTTP server of the thin thermostat runs , respectively. The central thermostat stores these values so that he can make requests to its registered devices.

The `type` field can take one of two values: `thin` (for thin thermostats) and `outside` (for the device that represents the outside temperature).

The central thermostat responds with status code `200 OK` in the case of a correct request, and with `400 BAD REQUEST` otherwise.

The polling and actuation protocol

At regular intervals, the central thermostat will make HTTP requests to registered clients (at the ip address and port that were specified by the thin thermostat during its registration).

- **Retrieve temperature**
  - Request: GET /temperature
  - Response: `200 OK` with an entity body of content-type "text/plain" containing a value like, e.g. "298.15". The value is expressed in Celcius.
- **Retrieve presence**
  - Request: GET /presence
  - Response: `200 OK` with an entity body of content-type "text/plain" containing either the value "true" (presence detected) or "false" (presence not detected).
- **Actuate valve**
  - Request: PUT /valve with entity-body of content-type "text/plain" containing an integer number between 0 and 100 inclusive. (expressed in percent)
  - Response: `400 BAD REQUEST` if illegal request. `200 OK` if valid request.
- **Retrieve Actuation value**
  - Request: GET /valve
  - Response: `200 OK` with an entity body of content-type "text/plain" containing an integer value between 0 and 100 inclusive (expressed in percent)

Note in particular that the device that measures outside temperature can only measure temperature; it will respond with `400 BAD REQUEST` to all other requests.

C Connecting to the Raspberry Pi

There are two ways that you can connect to your Raspberry Pi:

1. Connect the pi to an external monitor using the provided HDMI cable. In addition, plug in a USB keyboard and mouse.
2. Connect to the wireless network with SSID "TRANH201-GXX" of your Raspberry pi, with XX your group number. Then, use SSH[6] to connect to the IP address of your PI, namely 192.168.10.1.

Your username is `pi` and your password is `raspberry`.

You can also hook a fixed ethernet cable to the Pi's ethernet port. The PI will then try to automatically get an IP address through DHCP in order to make itself available on your local network.

[6] Using the linux command ssh or using the windows program PUTTY (`http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`).