Chapter 15

Problem: occupancy estimators

15.1 Problem Statement

You are going to be more autonomous than before to solve this problem. Just remember that your codes have to be object oriented. It means that you have to design the architecture of the code before coding, avoiding sequential solving. Classes of objects have to be coded in the most general way as possible i.e. as independent as possible of the other classes for the sake of readability but also for the sake of re-usability. As a result, your main classes should be almost empty (usually something like new Class()) and your class files should be small (200 lines max but smaller is usually better).

15.1.1 General objective

The objective is to develop a tool in Java to support the design of occupancy estimators for an office. Here is description:

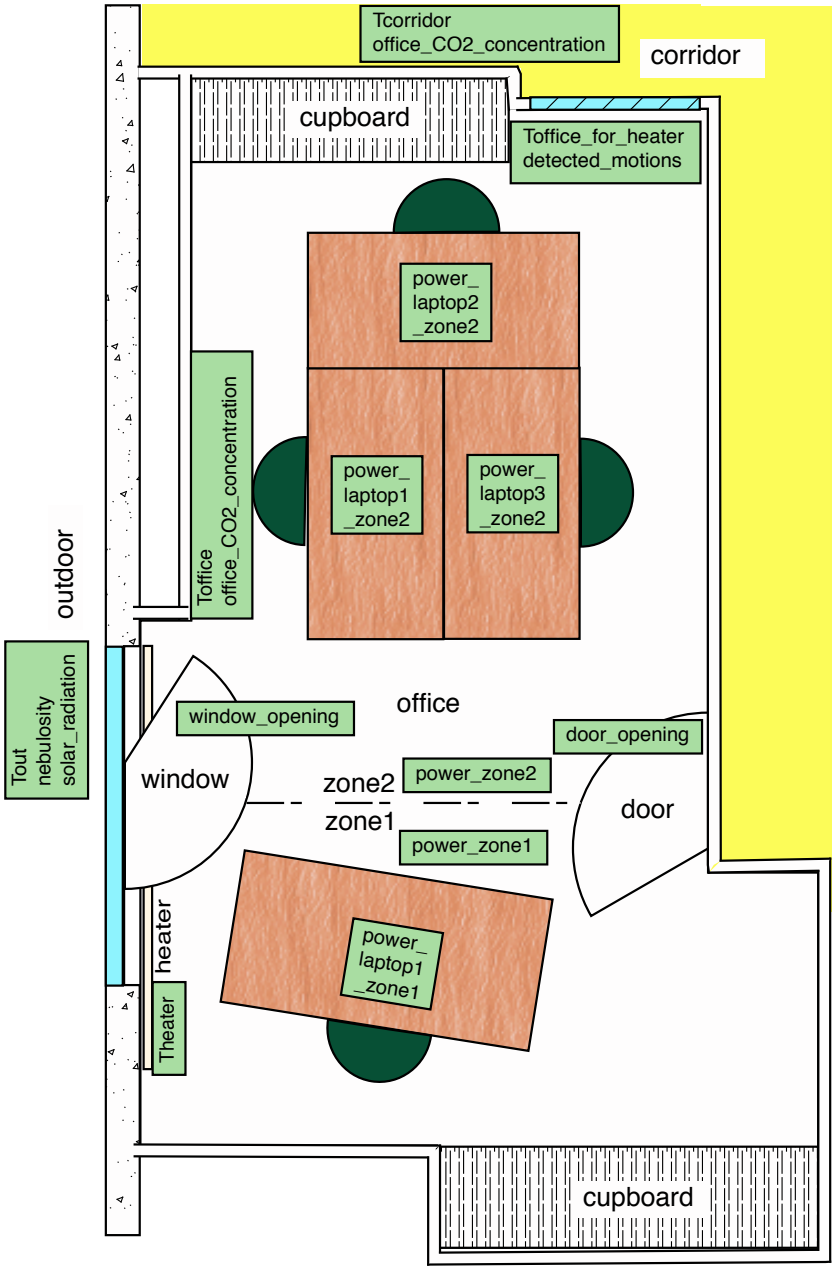

Figure 15.1 Plan of the office under consideration

Sensor names appear into green squares. The corresponding measurements covering the period April, 1st to June, 30th 2015 are given in the zip file 'occupancy-help.zip', in file named 'office.csv'. Data have been sampled with a 1 hour time resolution. The comma-separated values (see https://en.wikipedia.org/wiki/Comma-separated_values) text file format has been chosen because it is easy to read and widespread. Even LibreOffice and Excel can read it. In subsection 15.2.1, an easy way to read

this file is given.

The problem shall be solved following these steps:

1. design a graphic user interface to generate plots with selected variables, in order to be able to visualize measurements as well as new generated data
2. develop an occupancy estimator based on laptop power consumptions
3. develop an occupancy estimator based on motion detections
4. develop an occupancy estimator based on CO2 concentrations
5. analyze the results and summarize

A discussion about the the pros and cons of the difference estimators is requested. Practical considerations should be taken into account. Note that a communicating power meter is about 80 euros, a motion detector 120 euros, a contact sensor for door or window 50 euros and a CO2 sensor about 250 euros.

The code your are going to deliver must comply with the Java naming standard and be easy to read. It must be objected oriented. Because you are Java beginners, an example of code architecture is given in 15.2.2.

Design a graphic user

The graphic user interface should look like:

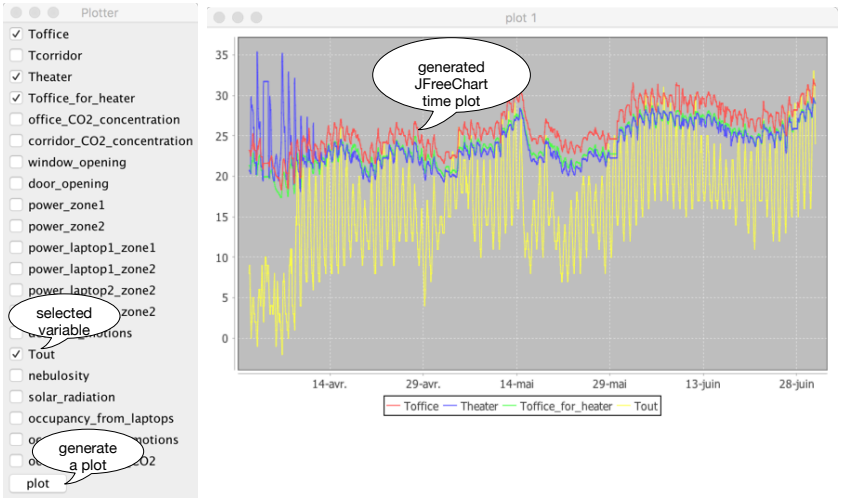

Figure 15.2 Expected GUI type for visualizing measurements and estimations

Occupancy estimator based on laptop power consumption

Occupancy estimator based on laptop consumption is very easy to do. The consumption of each one of the 4 laptops is measured by a power meter (variable 'power_laptopX_zoneY'). If the average consumption is greater than the standby consumption (15W), it is considered that someone is working on the computer.

Occupancy estimator based on motion detections

Estimator based on motion detections is also easy to do. It is considered that the number of motions detected within a hour is proportional to the number of occupants in the office. In order to determine the best proportional coefficient, a simple dichotomy can be done, considering the laptop consumption based estimations as the actual occupancies. Scale the coefficient to minimize the error between laptop consumption and motion detection based occupancy estimators. Algorithm is given in 15.2.3.

Occupancy estimator based on CO2 concentration

The CO2 based estimator relies on an air mass balance. Let's call Q_{out}, the renewal air flow exchanged from indoor and outdoor. It can be decomposed into $Q_{\text{out}} = Q^0_{\text{out}} + \zeta_{\text{window}} Q^{\text{window}}_{\text{out}}$ where Q^0_{out} stands for air infiltration, $Q^{\text{window}}_{\text{out}}$ for the average constant renewal air flow going through the windows when it is opened and ζ_{window}, the window opening. It is worth 0 when it is always closed and 1 when it is always opened during a considered time period. An intermediate value represents the fraction of the period the window was opened. This value corresponds to the variable 'window_opening' of the CSV file.

In the same way, the renewable air flow through the door is given by: $Q_{\text{corridor}} = Q^0_{\text{corridor}} + \zeta_{\text{door}} Q^{\text{door}}_{\text{corridor}}$. ζ_{door} corresponds to the variable 'door_opening' whose values are coming from a contact sensor.

Obviously, Q_{out} and Q_{corridor} are time-dependent because of openings ζ_{window} and ζ_{door}.

The air mass balance leads to:

$$V \frac{d\Gamma_{\text{office}}}{dt} = -\left(Q_{\text{out}} + Q_{\text{corridor}}\right)\Gamma_{\text{office}} + Q_{\text{out}}\Gamma_{\text{out}} + Q_{\text{corridor}}\Gamma_{\text{corridor}} + S_{\text{breath}} n$$

with:

- Γ, the CO2 concentration and $Q_{\text{out}} \approx 395\text{ppm}$, the outdoor average CO2 concentration
- V, the room volume i.e. $55m^3$ for the office
- S_{breath}, the average CO2 production per occupant. It is estimated at $4\text{ppm}.m^3/s$
- n, the number of occupants in the office

Considering average values over a time period of $T_s = 3600$ seconds, the differential equation can be solved to get a recurrent equation (X_k means average value of X during period $[kT_s, (k+1)T_s)$):

$$\Gamma_{\text{office},k+1} = \alpha_k \Gamma_{\text{office},k} + \beta_{\text{out},k}\Gamma_{\text{out,k}} + \beta_{\text{corridor},k}\Gamma_{\text{corridor,k}} + \beta_{n,k} n_k$$

with:

- $\alpha_k = e^{-\frac{Q_{\text{out},k}+Q_{\text{corridor},k}}{V} T_s}$
- $\beta_{\text{out},k} = \frac{(1-\alpha_k)Q_{\text{out,k}}}{Q_{\text{out},k}+Q_{\text{corridor},k}}$
- $\beta_{\text{corridor},k} = \frac{(1-\alpha_k)Q_{\text{coridor},k}}{Q_{\text{out},k}+Q_{\text{corridor},k}}$
- $\beta_{n,k} = \frac{(1-\alpha_k)S_{\text{breath}}}{Q_{\text{out},k}+Q_{\text{corridor},k}}$

The estimator comes directly from that equation. It yields:

$$\hat{n}_k\left(Q^0_{\text{out}}, Q^{\text{window}}_{\text{out}}, Q^0_{\text{corridor}}, Q^{\text{door}}_{\text{corridor}}\right) = \frac{\Gamma_{\text{office},k+1} - \alpha_k\Gamma_{\text{office},k} - \beta_{\text{out},k}\Gamma_{\text{out,k}} - \beta_{\text{corridor},k}\Gamma_{\text{corridor,k}}}{\beta_{n,k}}$$

$Q^0_{\text{out}}, Q^{\text{window}}_{\text{out}}, Q^0_{\text{corridor}}, Q^{\text{door}}_{\text{corridor}}$ are constant values that should be determined using a simple simulated annealing optimization algorithm given in 15.2.4. The problem is to find the best values, considering the laptop consumption based estimations as the actual occupancies. Following initial values are proposed:

- $Q^0_{\text{out}} = \frac{25}{3600} m^3/s$
- $Q^0_{\text{corridor}} = \frac{25}{3600} m^3/s$
- $Q^{\text{window}}_{\text{out}} = \frac{150}{3600} m^3/s$
- $Q^{\text{door}}_{\text{corridor}} = \frac{150}{3600} m^3/s$

Note down and discuss the 4 estimated constant air flows.

15.2 Clues

15.2.1 Reading a comma-separated values (CSV) from Java

Here is a simple example (it cannot be used in this way because it is not object oriented):

```
 import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadCSV { // be careful: this is not object oriented

   public static void main(String[] args) {
      String fileName = "office.csv";
      BufferedReader bufferedReader = null;

      try {
         bufferedReader = new BufferedReader(new FileReader(fileName));
         String line;
         while ((line = bufferedReader.readLine()) != null) {
            String[] tokens = line.split(",");
            for(String token: tokens)
               System.out.print(token + ",  ");
            System.out.println();
         }
      } catch (IOException ex) {
         ex.printStackTrace();
      } finally {
         if (bufferedReader != null) {
            try {
               bufferedReader.close();
            } catch (IOException e) {
               e.printStackTrace();
            }
         }
      }
      System.out.println("Done");
   }

}
```

15.2.2 Possible architecture for the application

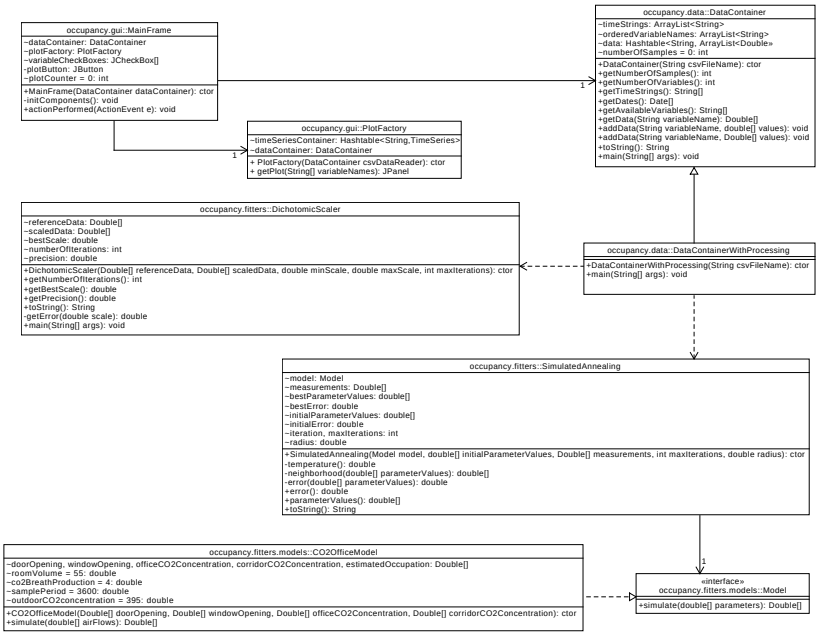

Figure 15.3 Class diagram of a possible architecture for the application

15.2.3 Dichotomic search algorithm

Finding the minimum of a 1-variable convex function can be done by dichotomy. Consider a function $f(x)$ defined on $[m, M]$. The algorithm is given by:

$l \leftarrow m$ [l for lower]
$u \leftarrow M$ [u for upper]
for $i \leftarrow 0 \dots i_{max}$ **do**
  $c \leftarrow \frac{l+u}{2}$ [c for central]
  if $f(l) < f(u)$ and $f(c) < f(u)$ **then**
    $u \leftarrow c$
  else if $f(c) < f(l)$ and $f(u) < f(l)$ **then**
    $l \leftarrow c$
  else
    break
  end if
end for
$x^* \leftarrow c$
precision $\leftarrow \frac{u-l}{2}$

15.2.4 Simulated annealing algorithm

Simulated annealing (https://en.wikipedia.org/wiki/Simulated_annealing) is an multi-dimensional improvement optimization algorithm with diversification. Let's consider a function $f(X)$ of n variables $X = x^0, \ldots, x^{n-1}$ that must be minimized. The algorithm performs random jumps in a neighborhood. Let's define the neighborhood random jump function $J(X_k) \rightarrow X_{k+1}$:

for $i \leftarrow 0 \ldots n-1$ **do**
  $x^i_{k+1} \leftarrow x^i_k(1 + (2 \times \text{random}(0,1) - 1)r)$
end for

where $r \in (0,1)$ is the radius defining the neighborhood as a fraction of the current variable value x^i_k, k is the current iteration and random$(0,1)$ is a random value between 0 and 1.

Then, if the new variable values lead to a better $f(X_{k+1})$ result, it can be kept for a next random search and forget otherwise.

But simulated annealing is more than that. Always trying to improve a result can lead to local minimal. To get around this problem, a diversification mechanism is added. It means that sometimes, even if the result is worse, it can be kept, expecting latter better results. A temperature function is used to allow more worse results at start than at the final iterations. Therefore, the temperature must decrease with the iteration number. It must also decrease if the current best value for $f(X)$ is decreasing. We suggest the following simple temperature function $T(k, f^r) = \beta f^r \frac{k_{max}-k}{k_{max}}$ where β is a freely tunable parameter that impacts the convergence ($\beta = 1$ is a good start). Note that f^r is the current reference function for stepping and f^* is the current minimum function value.

The simulated annealing algorithm can then be given:

$X^c \leftarrow X_0$ [c for candidate]
$X^r \leftarrow X_0$ [r for reference]
$X^* \leftarrow X_0$ [$*$ for best]
for $k \leftarrow 0 \ldots k_{max} - 1$ **do**
  $X^c \leftarrow J(X^r)$
  if $f(X^c) < f(X^*)$ **then**
    $X^* \leftarrow X^c$
  end if
  if random$(0,1) < e^{\frac{f(X^r)-f(X^c)}{kT(k,f^r)}}$ **then**
    $X^r \leftarrow X^c$
  end if
end for