Diploma Thesis

Intelligent, Learning Systems ABI Mark II

Stephan Kei Nufer
<snufer@ini.phys.ethz.ch>

Mathias Buehlmann
<mbuehlma@ini.phys.ethz.ch>

Advisors
Prof. Dr. Rodney Douglas, Institute of Neuroinformatics, ETH/University Zurich
Prof. Dr. Josef Joller, University of Applied Sciences Rapperswil
Tobi Delbruck, Group Leader, Institute of Neuroinformatics, ETH/University Zurich

A cooperation between

Computer Science Department
University of Applied Science Rapperswil
Oberseestrasse 10
8640 Rapperswil, Switzerland
http://www.hsr.ch

Institute of Neuroinformatics
University and ETH Zurich
Winterthurstrasse 190
8057 Zurich, Switzerland
http://www.ini.unizh.ch

Compiled: May 10, 2006

Typeset by LaTeX

Preface

by Nufer Stephan Kei and Buehlmann Mathias

This is the diploma thesis of our research work here at the Institute of Neuroinformatics (INI). Next to this paper there are a other related documents which were either also part of this thesis or part of the latter term project ([NB05a]).

These are:

- The RBC Protocol Specification ([NB05c])
- The RBC API ([NB05b])

Acknowledgment

Special thanks goes to Josef M. Joller and Rodney Douglas who gave us this wonderful opportunity to conduct this diploma thesis here at the Institute of Neuroinformatics. We appreciated the calm environment as well as all valuable discussions we've had with lots of people here. We would also like to thank Tobi Delbruck for his moral support and advice and for implementing the enhanced PC Presence detectors.

Further credits goes to:

- Simone Schuhmacher, Kathrin Aguilar Ruiz-Hofacker, Mr. fruit fly Steven Fry, Tobi Delbruck and Giacomo Indiveri for letting us control their room
- Hans Eichenberger and Ueli Rutishauser for valuable discussions
- Bobby Rohrkemper for revising the abstract of this thesis
- Albert Cardona for his mental support and discussions
- All other predecessors that have been working on ABI before

Zurich, Institute of Neuroinformatics(INI), Switzerland, October 2005

Stephan Kei Nufer and Mathias Buehlmann

Abstract

Intelligent learning systems have become increasingly common due to improvements in user comfort and security provided by automated robots and self-adapting systems.
The field of autonomous building control systems at the Institute of Neuroinformatics(INI) ([Ins]) has gained our research interest.

Novel building architectures have been gradually equipped with an entire communication network. Now, this establishes a new world of possibilities since internal building devices can programmed and controlled remotely.
Depending on occupants' needs, such a programmable system could improve user comfort and save more energy than a manually controlled system. Currently, ordinary lighting controls systems require frequent maintenance such as replacement or re-calibration of sensors and effectors. Self-adapting systems can prevent these burdensome and costly tasks. Systems that account for sensor degradation can enhance user comfort.

However, this seemingly easy task is complicated since structural changes must be detected and incorporated in real-time.
A major obstacle for autonomously learning dynamic space behaviors and configurations is that sensors perceive and react very differently from the human brain. For instance, we perceive fog with a different luminance intensity than measured by sensors. Furthermore, ambient light levels change dramatically even with small atmospheric changes such as a momentary scattering of particles over the sun. Hence, any prediction system needs to be flexible enough to react to such environmental changes.
Furthermore, in order to control an environment an IB must gradually receive feedback from its occupants to adapt to constantly changing needs. An additional difficulty is that user instructions are very sparse and thus make learning behavior tedious.

In a term project, we implemented an advanced adaptive framework, built on the Open Services Gateway initiative (OSGi), which allows the control of a building. In this diploma thesis, we have made several major improvements including incorporating additional sensors and effectors that can be assessed through a dedicated fieldbus network (LonWorks).

Here, we also present a novel IBF (Intelligent Building Framework) that incorporates a set of IB Agents, each of which tries to learn its dynamic living and working environment. Among other things the IB Framework introduces a design that integrates the most credible information currently available from several prediction algorithms. In addition to the overall infrastructure, we implemented and studied several machine learning algorithms. We analyzed them using our own developed real-time simulation platform environment.

Furthermore, we introduce a variety of novel user interaction agents that allow environmental parts to be administered and controlled using the Java Web Start technology. This facilitates any deployment work that would otherwise be necessary.
Also, a database application has been written which records all sensor, effector, and environmental changes to the INI structure. Later, this information may be useful to employ data mining, a procedure which requires real (rather than simulated) data.

Table of Contents

List of Figures

Part I

Introduction

Chapter 1

Motivation

With the advent of programmable systems, building managers must gradually adjust lighting levels in response to occupant requests and thus definitely opens the possibility for an intelligent building (IB) to handle such tasks by considering aging of devices, changing sky and user behaviors as well as taking possible structural variations into account. An IB can lower the energy consumption as well as maximizing visual user comfort and also lessens cost-intensive maintenance that elsewise would be necessary.

Many different type of building control systems have been developed by the industries these days but however most of them suffer from problems that need more advanced techniques to enhance user comfort and parallely achieve a low energy consumption. It should be noted that we clearly want to distance ourselves from the ordinary term lighting control. What we're interested in, is the development of an IB (Intelligent Building) that should be adaptive to real and specific user behaviors, not average or such. Furthermore the IB should be adaptive to building characteristics (self-adaptive) by considering different climatic factors.

In order for any control system to interact with a real working and living environment, access to different indoor as well as outdoor sensors and effectors must be provided (See figure: 1.1).

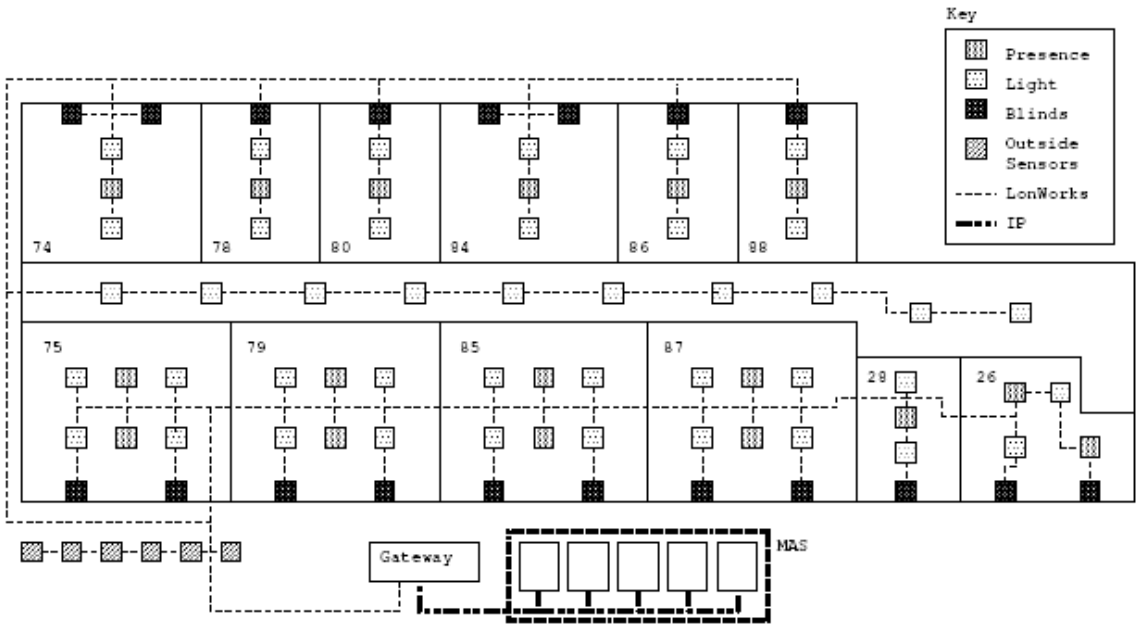


Figure 1.1: Multi-sensor environment, [TZ03b]

Our study is concentrating in developing an Intelligent Building (IB) system that provides maximal user comfort in an environment where occupants interact with effectors such as lights and blinds. Thus the comfort in our context can be limited to provide a maximal *visual comfort*. Casually speaking the system must from time to time be adapted to changing lighting conditions.
Our system should therefore act with the environment through common effector devices such as lights and window blinds as well as senses from illumination-, temperature-, radiation-, daylight sensors and presence detectors (Multi-sensor environment) (See figure: 1.1).

Nevertheless future environments might even allow a much broader set of effectors to be controlled and are all together targeted to be intelligently managed as well.
However the goal of our building control system can almost be restricted to user goals only since the aim of our system is to adapt user needs as far as possible rather then trying to achieve a low energy consumption in the first place. User satisfaction and acceptance is extremely important to us since a building is not meant to make use of the least possible energy while not maintaining a good "residential qualities" such as user comfort. Annoyances caused by the system, such as glare, temporary sudden changes in brightness, or irritating mechanical noise, will reduce the system's acceptance.

The luminance provided by the sky is variable compared to the luminance provided by synthetic lights. This variation creates one of the fundamental differences between synthetic lights and the outer daylight. Additionally the sky light intensity and resulting luminance vary with latitude, time of day and even with seasons. Within seconds the luminance of an environment may change as a result from the density and movement of clouds. It's not an easy task to setup desired luminance levels since thresholds may vary in consequence of user behaviors and environmental changes. Another difficulty, that has already been mentioned, is that we need to deal with a non-stationary environment. Such an environment signifies that stability can not be assumed since structural changes must be able to be detected and incorporated in any point of time as well. Additionally each sensor within an environment gradually requires a recalibration to account for degradation (Also known as *commissioning*). Perhaps even certain lamps are to be replaced or cleaned as part of normal maintenance.

Considering such a variability and non-stationaryness within an environment we can conclude that certain environments can not optimally be controlled by just providing regular daylight sensors. Also providing an adequate quantity and quality of daylight in interior spaces and parallely trying to achieve a low energy consumption are quite contradicting goals.

In summary we can freeze on to the fact that a building intelligence should:

- **Not disturb occupants:** User wishes are upper priority. For instance: Blind motor noises, flashing or flickering lights, etc. should be minimized. User desires must be manually configurable and should not end up into opposing actions.
- **Be reliable:** An IB application that would suddenly stop controlling an environment because of a network failure would ultimately result in discomfort.
- **Minimize energy consumption:** Decrease the energy consumption, without affecting the user comfort (especially visual comfort). Broadly speaking we can say that the system must provide a reasonable payback period.
- **Incorporate all available sensor information:** In order to enhance the predictions, an IB should consider all possible sensory information that can be received from an environment. Since the bigger the information flow the better adequately a building can be controlled (Multi-sensor environment).

Chapter 2

Introduction

Basically two adaptive building intelligence systems (ABI Systems) have now been developed by several term projects and diploma theses. Evidentially each system turned out to be very unstable and sometimes took unexpected actions and this still after a couple of months of being tested. Neither system barely was able to survive a month without having crashed. Evidentially the inhabitants that were using either systems solutions were not satisfied at least judged by the users acceptance or annoyance level. Maintenance or adaption was questionable due to their sparse documentation and design issues.

Hence in a term project (Intelligent, Learning System built on the Open Services Gateway initiative), a new ABI (Adaptive Building Intelligence) System has been developed that was based on OSGi. OSGi has emerged into a very powerful and adaptive system framework that enables an entirely new category of smart devices due to its flexible and managed deployment of services to be implemented and integrated.
The OSGi service platform specification delivers an open, common architecture for service providers, developers, software vendors, gateway operators and equipment vendors to develop, deploy and manage services in a coordinated fashion ([OSG]). With the usage of such a framework we achieved a system that is low coupled, stable, easy to maintain, expandable, easy to get acquainted with and most importantly we reduced the point of failures to a minimum. Furthermore the ABI System has been developed in a way that possible system developers don't necessarily need to know the entire system when adding some functionality. Thus we can state the system to posses hot-plugging capabilities.

In the latter term project ([NB05a]) we tested the ABI System infrastructure with the integration of wireless devices (so called Falcons). It is the goal of this diploma thesis to incorporate new sensors and effectors that are to be addressed by a dedicated fieldbus network, LonWorks ([loc]) and start a detailed analysis of possible learning algorithms in an ambient environment.

In retrospection to our major predecessors ([TZ03b], [RS02]) we'll combine our research work and among other things present a new approach of making a building act intelligently. Further we review some couple of issues that one will face in a multi-sensor environment. In particular we illustrate why such an easy seeming task is above all quite complicated to fulfill and excessively discuss one of the most important topics when it comes down to learning dynamic space behaviors and configurations (See chapter: 11).

A brief retrospection of the work of our predecessors might be quite useful. Hence we'll introduce their work and approach to understand what the open issues and problems were.

2.1 Ambient intelligent Environment

We now extend the motivation (See chapter: 1) by continuing the discussion to other relevant and related topics. Among other things we'll be talking about sensor problems but also other interesting facts which will be essential to perform any environmental learning tasks.

One of the shortcomings of the sensors, are the problem of reflectance factors, i.e. a large load of white papers are spread out on the work surface. In a certain degree this problem can indeed be overcome by a finding better placement of the sensors but since we assume to deal with a non-stationary environment not a really sufficient solution. Moving down the blinds may also not be an appropriate answer either since some occupants may not accept daylighting without a view.

Generally, occupants perceive the performance of an IB quite differently. If occupants perceive the environment created by the system to be uncomfortable or disturbing in any way (noisy or too abrupt in its on-off switching), the system is likely to be rejected or an attempt will be made to compromise it.

Also, experiences at the INI have shown that occupants don't effectively, manually control their effectors and quite often leave electrical lights on, once they're switched on and barely make a seem to switch them off again even if the illumination is at a level that would be considered adequate.
Apparently there is definitely a major improvement of visual comfort as well as lower energy consumption possible.

Further issues that need to be resolved are *reliable presence detection* and *opposing system interactions*.
The reliability of presence detectors have already been discussed by several theses and term projects ([RS02], [TZ03b], [BG04a]). In order to address those complaints we should consider a delay upon each presence spike that automatically will get extended to the maximal delay value that continuously degrade within the time being. Keeping a simple fixed delay however may not be suitable for every environment. However for our current purpose enough sufficient.

Reliable presence detection is one of the crucial factors when it comes down to controlling an environment since it directly impacts energy savings. Additionally it is of high importance for us since in contrast to our predecessors we don't perform any learning tasks when not sensing presence in the first place. Hence switching off the lights upon failing to correctly detect the occupancy of an environment will start to annoy inhabitants.
Providing reliable presence detection has always been a problem and thus we considered and integrated additional presence detectors that are known by the name *PC Presence detectors* (See chapter: 9) which track down the mouse motions and keyboard hits of a user's workstation.

The other issue that has been mentioned above is *opposing system interactions*.
Questionnaires ([CP01]) have shown that occupants quickly get angry if the system does not take their wishes into account. Wishes such disfavoring desired blind position in re-positioning them right upon they've been re-positioned by the user.
This is a major problem that starts to annoy occupants since their needs, evidentially don't get acknowledged by the system. This was also one of the major complaints of the latter systems since they didn't account a suitable delay upon each user interaction. Additionally we even noticed that such interactions are even considered as "favorable" for an IB to achieve. (See: [RJD04])
However in order to avoid such scenarios we held up the control of a device during a fixed amount of time (typically one hour). Recent studies performed by a related project called EDIFICIO ([CP01]), also determined this issue and applied a similar procedure to disburden such situations from actual IB tasks.

Actually our system was suppose to control blinds as well as lights within different spaces but due to severe blind problems that have been detected in several rooms, we could't succeed in getting them to work properly. This is because the blinds are, in the current condition, not really suited for open loop control. For instance we noticed that certain blinds even fail to respond to orders given by the LonWorks tools ([iLO]) at all. For instance, one of the problems that we identified, was that certain limit switches do not properly function and thus cause the blinds to go out of synchronization which subsequently starts to impact in the correctness of the returned position and rotation values.
On the other hand we cannot simply ignore blind inputs completely because the interior daylight is heavily depending on them by for instance blocking direct sunlight and glare that may start to get uncomfortable for building occupants. Speaking in general the exterior daylight as well as all synthetic lights and shading devices cannot be considered separately.

More discussion and details, especially in reference to IB can be found in chapter: 11.

2.2 Related Work

Ueli Rutishauser and Alain Schaefer [RS02]) introduced an approach that basically has been adapted and improved by Raphael Zwicker and Jonas Trindler ([TZ03b]). Their approach has been inspired by one major paper that has been published from J. L. Castro, J. J. Castro-Schez, and J. M. Zurita with the title named: *Learning maximal structure rules in fuzzy logic for knowledge acquisition in expert systems* ([CCSZ21]).

The inputs to the learning process are real valued variables acquired from sensors, the output of the learning algorithm is a model consisting of a number of fuzzy rules. These fuzzy rules are used by a fuzzy logic controller to take decisions. Feedback acquired from the environment is continuously used by the learning process to adapt the fuzzy logic rules ([RS02]).

The core idea behind such an approach is to cover-up as much of the input space as possible with the most general rules. This can be achieved by subsuming them into a definitive rule-set. This process is basically endlessly repeated as new training rules are provided by the system every once in a while (online). Such systems are also known as hierarchical fuzzy systems which in a certain degree try to overcome the curse of dimensionality ([ZK04]).

In contrast to the algorithm presented in the paper, they adapted the algorithm to work online rather then offline. Additionally since user inputs have been taken into account in form of a punish/reward feedback mechanism this system can also be considered as a derivative of classical LCS Systems that have chiefly been introduced by Holland([Hol86],[Bul05]) back in 1986.
Raphael Zwicker and Jonas Trindler ([TZ03b]) then introduced a long and a short term memory that counteracted the problem, identified by [RS02], that already learnt rules could be rejected by only one new instruction.

We belief that one of the most severe problems that makes learning with such an approach inadequate is that fuzzy logic has been taken into account. Thus the entire classification process is heavily depending on the design of the used membership functions which we identify to be one of the most important problems since they basically do not differ from manually having to calibrate the sensors again. Membership functions should change over time and thus makes this approach not self adaptive. Self-adaptive in the sense that the software shouldn't be aware of any specific space peculiarities.

We rely our conclusion to the following statement that has been given in the thesis of Raphael Zwicker and Jonas Trindler:

> As we have seen in our analysis an accurate design of the fuzzy membership functions is a really important task. The whole learning algorithm relies on this design. The current fuzzy membership functions are an approximation which were made by our predecessors in ([RS02]). However, we believe the accuracy of the fuzzy reasoning could be increased if these are defined based on statistical conditions which regard also different seasons. It would be thinkable to tune them also with a learning algorithm in order to adapt them to the current used range of the sensor. A normal sensor will sense for example during the wintertime in another range than in summertime. ([TZ03b])

Those important facts brought us to the conclusion and the intention of striking a completely different path of trying to make a building act intelligently. Nevertheless the results clearly showed us how complex the situation is here and thus were very valuable to us.

2.3 Terms and definitions

For the sake of clarity we use the term IB to refer to intelligent building or building intelligence in the same context. Also note that sometimes the term AI is used that corresponds to the term IB as well.
In the next chapters however we'll be using a row of definitions that mostly concern ABI System specific terms and definitions which were already introduced in our term project ([NB05a]). Hence since the new ABI System has completely been built using an OSGi Framework, which can be quite specific, we recommend to consult other sources such as ([OSG] or [KNO]) to get acquainted with it.
Some parts will introduced their own terms and definitions to simplify the reading. Other terms and definitions are to be looked-up in the glossary (See glossary section: 28)

2.4 Preconditions

This thesis presumes that the reader is already familiar with the following paper: [RJD04].

For the next chapters we also assume that the reader is acquainted with following diploma theses or term projects: ([NB05a], [NB05c], [NB05b], [TZ03b], [RS02])

Other theses or term projects that might be of interest are: [Fen04], [BG04a], [BG04b]

2.5 Document structure

This thesis has been structured into nine parts and twenty eight chapters.
In an introduction part, we covered some fundamental discussions and problems and most importantly motivated why it is worth to try to intelligently control a building (See: I). We also substantiated why we belief, the usage of fuzzy logic to be an inadequate solution for a self adapting system, by discussing the work of our predecessors.
We then present the extensions which needed to be made to the existing OSGi based ABI System, which is quite specific and thus will probably only be applicable for system maintainers to be read (See: II).

We then introduce the part where all developed distributed agent applications are presented (See: III). Distributed agent applications can hereby be split into: user interaction agents, logging agents and IB device agents.

In order to get acquainted with building intelligence, we'll start with a pre-part (See: IV) that mostly discusses some essential questions in outlook to the actual learning part (See: VI), where each applied and realized algorithm is presented and discussed. However before we examine each of the developed algorithm, we will first introduce our IB framework, as well as the developed Real-Time Simulation (RTS) software that allows synthetical environments to be created, in which each developed prediction algorithm will be tested with (See: V).

To summarize all artifacts we'll be discussing all results, gained from either real or simulations tests in a dedicated part (See: VII).

In the final part (See: VIII), we conclude our results and discuss encountered drawbacks that couldn't have been solved on time. In a future work chapter we further capture some of the identified issues, as well dissuss minor suggestions that will need to be fixed in the near future.

Finally we come to the end that maintains the Glossary and a Bibliography (See: IX).

This thesis is actually not really intended to be read page by page from beginning to end since this thesis has different type of readers. Hence readers are invited to go directly to the sections that address their interests. For example, readers seeking general knowledge about learning should go straight to part: VI. Likewise for those readers which are interested in ABI System specific constraints, should move to part: II. When being interested in distributed agent applications such as the Area Controller Agent (See chapter: 7) or the Area Logger Agent (See chapter: 10), we recommend to jump to part: III.
Specific knowledge about the environment as well as how different influences will impact the learning of a building and its occupants should advanced to part: IV.

We would like to point out that some chapters or parts provide their own preconditions, definitions and conditions.

Part II

ABI System

Chapter 3

OSGi based ABI System

3.1 Introduction

In our previous term project ([NB05a]) we mentioned that the LON bus integration couldn't have been conducted, due to an urgent time consuming refactoring and thus needed to be postponed and realized within the very beginning of the diploma thesis.

In order to bridge to our latter term project we briefly retrospect its architecture and outline the extensions that will need to be incorporated to gain a fully functioning OSGi based ABI system. We clearly show that the learning will benefit from the developed platform in the sense that it rather needs to concentrate in the development of different learning methodologies which are to be recorded and monitored.

The next chapters within this part are rather meant to be used as a reference book since they explain each new developed bundle in detail.
Hereby, each bundle has dedicated a separate chapter, which will explain how they have been designed and implemented. Most sections within each chapter provide a mixture between descriptive text, UML diagrams and code snippets. We thought that it was worth to provide a combination of the three since certain things are in one way easier coded then described and on the other sometimes easier described or illustrated then provided in code snippets.

3.2 ABI System Overview

In figure: 3.1, we can depict the OSGi ([KNO]) based ABI System that is completely constructed using bundles, rather then one single software. One major benefit of such a lo of each other and plugged at runtime without impacting other running bundles. For instance when one of the bundles such as the RBC Server bundle would produce a major breakdown (i.e. losing one of its threads due to an exception) it won't impact other running bundle services bundles can anytime be stopped and re-launched. Hereby, temporary knowledge can be stored in their context and re-initialized, as nothing would have happened.
In general we can state that each bundle contributes a bunch of services to the ABI System. One of the most important services is hereby provided by the RBC Server bundle. This service implements a RBC Protocol conformable server which will ultimately allow, distributed agents to communicate with the ABI System.

3.2.1 RBC Protocol

The purpose of the RBC Protocol (See figure: 3.1) is to describe the protocol to be used for exchanging device information between the ABI System (RBC Server Bundle) and custom agent applications, that need

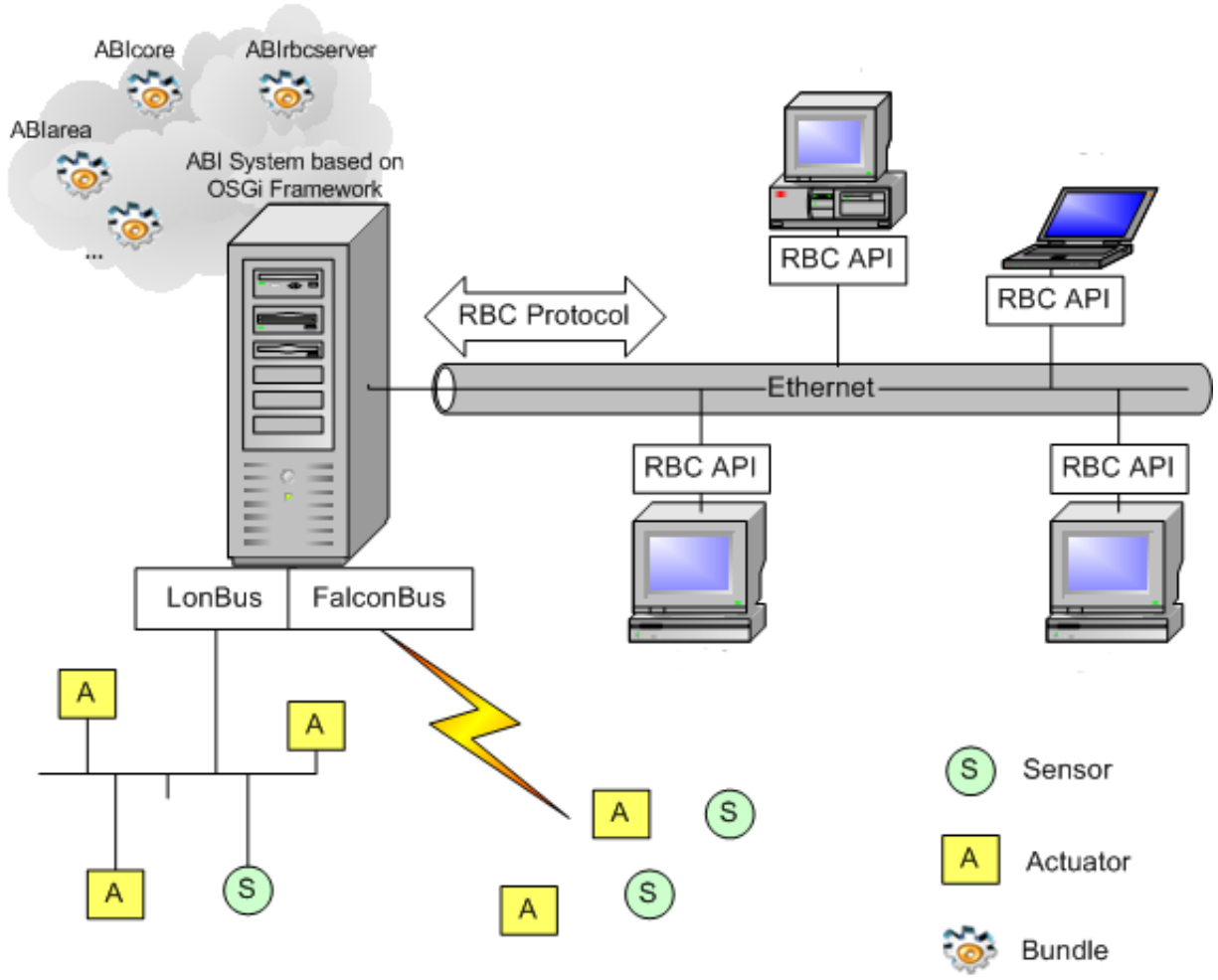


Figure 3.1: ABI System Overview ([NB05a])

guaranteed reliable transmission of data in a simple, ascii-based protocol. One major use of this protocol is to enable agents to retrieve changing device information and on the other hand commands which are executed in the RBC Server similar to remote procedure calls (RPC). Hereby it provides a standard that all ABI Agent applications need to adhere when communicating to the system.

3.2.2 RBC API

The RBC API specifies a new compact application layer standard which exemplary has been implemented by a concrete Remote API that implements the RBC Protocol requirements as it was prescribed by its specification. This concrete implementation remotely communicates to the RBC Server and herewith provides an ideal basis for distributed agents such as Device, User Interaction or Logging Agents to use for their realization.

A benefit of such a system is that all distributed Agents such as the Area Controller Agent (7) can now be developed independently without having to deal with the ABI System itself. Thus improving any development practices such as complex integration testing, debugging and maintenance remarkably.

With this approach we can also relocate any building intelligence to distributed device agents instead of providing a monolithic architecture inside the ABI System. Hereby each agent owns its own framework that is logically quite similar to that one provided by the ABI System.

3.2.3 LON and Falcon Bus

As depicted in figure: 3.1, the entire ABI System allows devices from two buses to be involved in the entire system. It might be worth to lose some words about the Falcon bus in order to give some basic understanding on how the LON ([loc]) bus must be integrated. According to the bus concept that has been introduced by our predecessors ([BG04a], [BG04b]) each device-set such as the falcon device-set needs to implement a separate bus. The reason why such a distinction has been made is because the concept presumes that each device-set is controlled by some kind of controller. In the LON bus this happens to be a gateway called LNS Server ([iLO]).

The Falcon Communicator, as it has been realized within our latter term project ([NB05a]), communicates with its devices using wireless instead of the LON technology. Hence, one can simply say that each bus basically needs to implement a proxy that replicates such a controlling entity. Hereby it must provide a way to connect and disconnect to respectively from the bus and furthermore should implement methods that allow any devices to be registered and unregistered as well.

3.3 ABI System Architecture

According to our term project we subdivided the whole system into smaller subsystems. Thereby all subsystem are composed of a set of bundles, each of which conceive the difficulties and provide a suited solution to comply its subsystem (See figure 3.2).
In order to support the new LON devices, one of the major subsystems needs to be extended to finally provide new devices from the LON to be incorporated. The category where this subsystem falls into is, according to our term project called `Bus and device abstraction`.

This part is impacting two major bundles. The Core Bundle (See chapter: 4 or [NB05a]) and a new bundle that we call **ABI LON Bundle** (See chapter: 5). Similar to the ABI Falcon Bundle (See figure: 3.2), this bundle must provide its own bus implementation that needs to address its own devices.

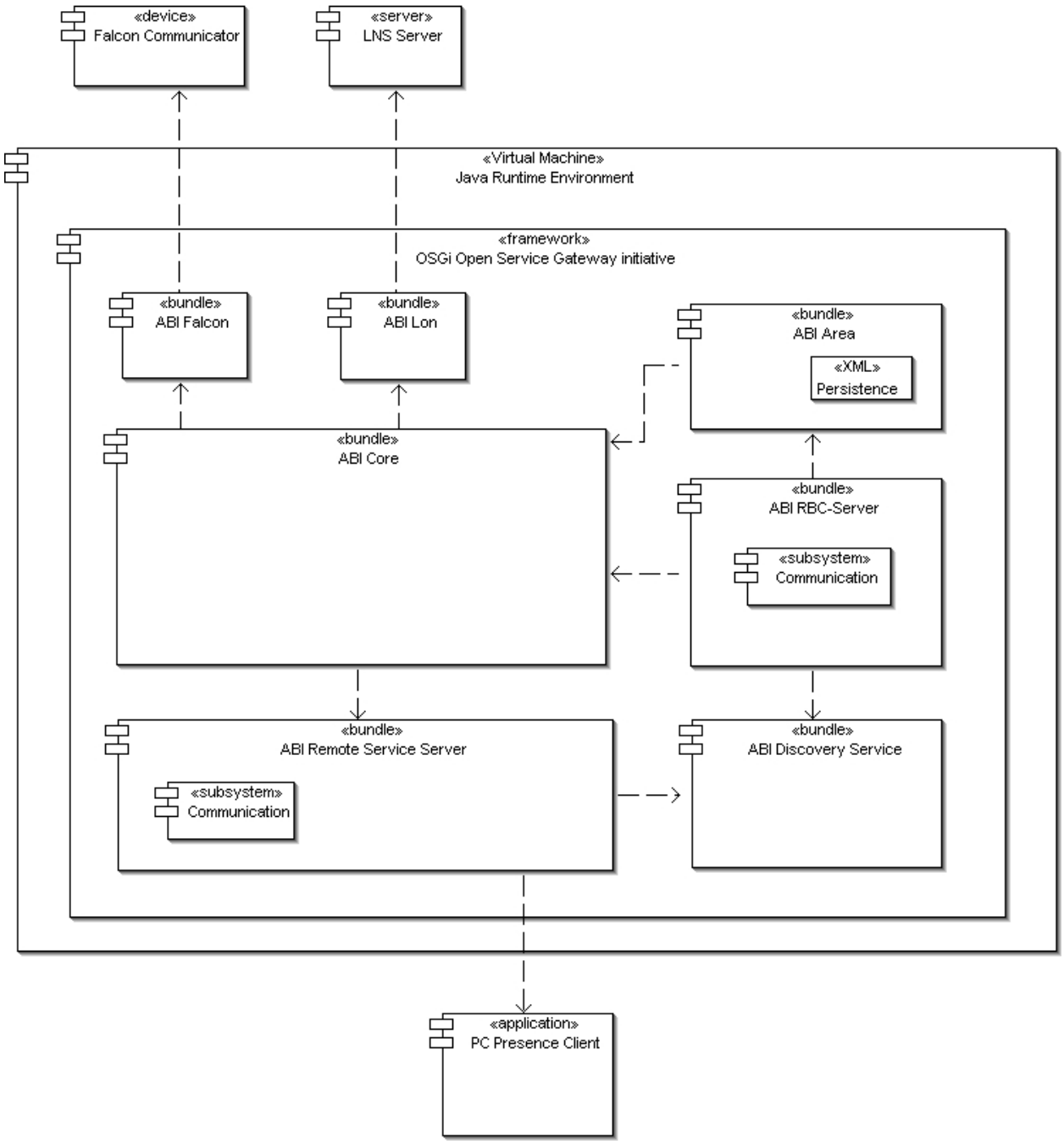


Figure 3.2: ABI System Architecture Overview ([NB05a])

Chapter 4

ABI Core Bundle

To feature the basic functions of the ABI Core Bundle we briefly summarize the most important tasks of it.

The design of the ABI Core bundle can basically be subdivided into three main parts. The first one is the general Abstract Bus concept which provides the generic abstract bus interface and a multiplexer that manages the device objects. The second part rather defines a set of interfaces which prescribe all currently supported services (i.e. a presence service). The last part actually puts the generic property concept into practice that has been introduced in our term project ([NB05a]). This is actually the part where we need to put our new extensions onto. By integrating the LON Bus, additional new devices can be incorporated that provide a much broader set of functions, then the Falcon devices such as additional sensory input devices as well as new actuators. Hence to integrate those new devices, corresponding services must be prescribed that eventually need to be implemented (triggering and control) (See the subsections: 4.1, 4.2).

4.1 Sensor services

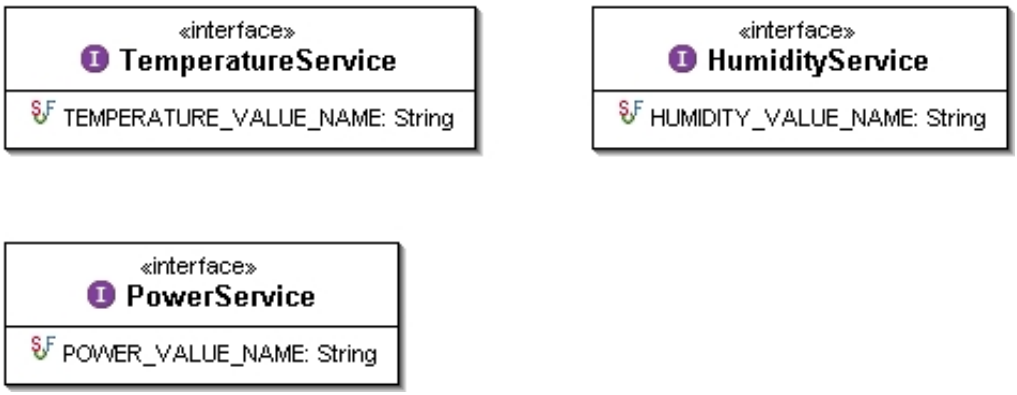


Figure 4.1: LON sensor services

The `TemperatureService` exposes an interface that each service must implement when representing a device that is capable of measuring the temperature. In the Institute of Neuroinformatics (INI) ([Ins]) there is only one such device that is located on the roof of the building.

```
1 public interface TemperatureService
2 {
3     // The value itself should preferably be a floating point type.
4     public static final String TEMPERATURE_VALUE_NAME = "temperaturevalue";
5 }
```

Similar to the `TemperatureService`, the `HumidityService` exposes an interface that each device must implement, when representing a service that is capable of measuring the humidity. In the Institute of Neuroinformatics there is also only one such device that is located on the roof of the building as well.

```
1 public interface HumidityService
2 {
3     // The value itself should preferably be a floating point type.
4     public static final String HUMIDITY_VALUE_NAME = "humidityvalue";
5 }
```

One of the interesting features that the LON Bus brings along is a device that is capable of measuring the power consumption from the base floor. As a result the `PowerService` exposes an interface that a concrete service must implement, when representing a device that is capable of measuring the power consumption. Here at INI there're a couple of such devices that do such a job.
This service might be of interest when trying to comply to the goal, of achieving a low power consumption next to maximizing user comfort by intelligently controlling the devices within a room according to the needs of its occupants.

```
1 public interface PowerService
2 {
3     // The value itself should preferably be a floating point type.
4     public static final String POWER_VALUE_NAME = "powervalue";
5 }
```

4.2 Actuator services

Within the integration of the LON Bus, one additional device shows up on our list that correspond to a blind services that supports a large variety of features to control LON-based blinds. Features such as lifting up entire blinds but also movements with advanced accuracy can be achieved with them.
The first pre-specified set of properties hereby corresponds to blind commands, which can be utilized to achieve casual movements with less accuracy. On the other side we provide two supplementary properties (blindposition and blindrotation) which are readable as well as writeable and thus supply more accurate ways to adjust the blinds.

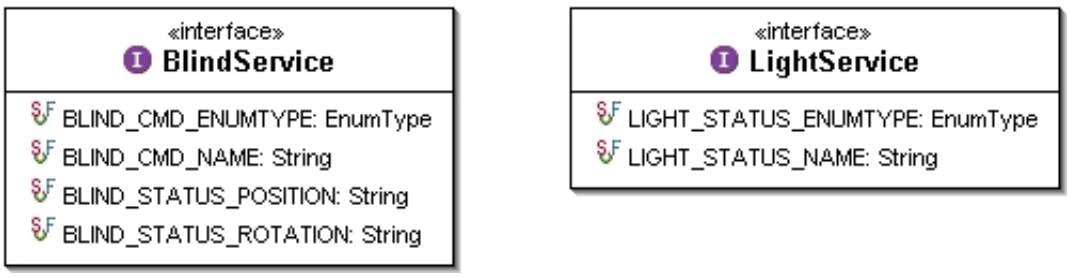


Figure 4.2: Sensor services overview

The `BlindService` exposes an interface that each device must implement when owning a service that is capable of measuring the daylight.

```
1 public interface BlindService
2 {
3     // Writable properties which basically serve as commands
4     public static final String BLIND_CMD_NAME = "blindcmd";
5
6     public static final EnumType BLIND_CMD_ENUMTYPE = new EnumType(
```

```
 7           new String[] { "UP", "DOWN", "STEPUP", "STEPDOWN", "STOP" });
 8
 9     public static final String BLIND_STATUS_POSITION = "blindposition";
 10
 11    public static final String BLIND_STATUS_ROTATION = "blindrotation";
}
```

More information about the Core Bundle should be obtained from our term project ([NB05a]) since it wouldn't make sense to recap entire concepts in here again. So keywords such as `Property Concept` or the `Abstract Bus Concept` should not be unfamiliar to you. It should be clear that by supporting additional devices from a new technology, new concrete services will need to be supported as well. Therefore a new bus has been implemented that addresses its new devices.
By taking advantage of the properties we can thereby specify each feature supported by any device in detail. Details such as ranges or types. For instance a temperature device measures values within a distinct range that will provide scaling information when it comes down to *input-normalization*. Additionally each concrete device should specify how its data should be represented i.e. as floating point.

Chapter 5

ABI LON Bundle

5.1 Overview

This section discusses how we realized the controlling part of the LON devices as well as how we finally incorporated them into the ABI System.
According to the term project's proposition, a new concrete bus has been implemented that is capable of communicating to the LON devices. Hence just like the Falcon Bus, we implemented a new specific bus that is responsible for the low level communication part that happens to be Ethernet this time.

Since the communication is based on Ethernet, no further separation of the domain is really necessary since Echelon ([loc]) provides the necessary API's which facilitates any device access in a very convenient way. Therefore, the only part we need to take care of, is the successful integration of the new concrete bus and its devices (that we call *LON Bus and LON devices*) into the ABI System.

5.2 LON Bus

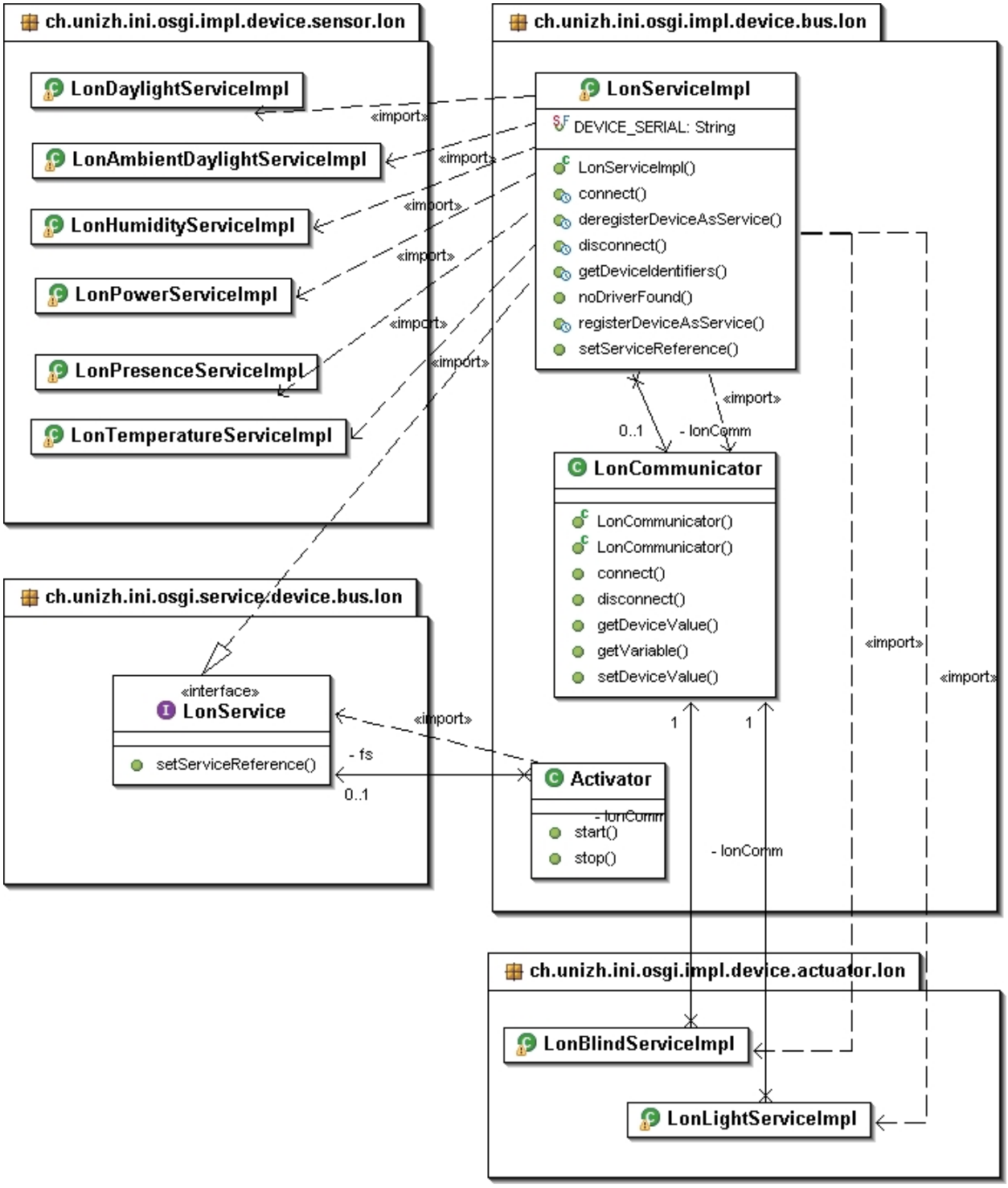


Figure 5.1: Lon Bus

In order to facilitate the reading of this chapter we tried to comply to the documentation style that has already been used in our term project ([NB05a]).

According to the ABI Core interfaces (See chapter: 4) each device provides a corresponding device im-

plementation that addresses its physical device and within its features.
As depicted in figure: 5.2 one will notice the class called `LonServiceImpl`. This class replicates the hardware that provides the main entry point to communicate to all interconnected LON devices. Therefore you can think of the `LonServiceImpl class` as a virtual proxy that has been implemented as a separate bus. When taking a deeper look at this class, one will notice the two methods called: `connect()` and `disconnect()`. The `connect()` method initializes and establishes a connection to the LNS Server which is chiefly responsible for receiving and sending all the data that basically can be considered as simple queries.
The real connection establishment is actually forwarded to a class called `LonCommunicator` which does all the required initializations that is necessary to query any device currently hocked-up on the LON Bus in the first place. Hereby we can state the bus as "CONNECTED" and proceed with any additional commands. Inversely the `disconnect()` method shuts down the Ethernet connection from the LNS Server and hereby breaks up any pending connections.
The following code snipped has been taken out of the `LonServiceImpl class`:

```
1 public synchronized boolean connect()
2 {
3     if (connected)
4         return true;
5
6     lonComm = new LonCommunicator(FIELDBUSADDRESS, FIELDBUSPORT);
7     connected = lonComm.connect();
8     return connected;
9 }
```

Hereby the FIELDBUSADDRESS correspond to the IP-Address: **172.16.2.200** and the FIELDBUSPORT to **2540**. As one might know, each device supported by the LON Bus needs to be addressed by so called **LNS-Network-Variables**. Each device within our network hereby physically owns a variety of such network variables. Unfortunately each thesis was having troubles with them since certain variables where incorrectly defined in either the database of the LNS Server, or in their own software. The reason why all our predecessors and we had issues with them, is because certain devices differ between whether a device notifies its state change or physically conduct changes, i.e. when switching of the lights. Hence sometimes, two such variables need to be defined since in one way we need to be capable of ordering a devices to change its state and on the other hand we must provide a way to receive changing device information. Imagine of not being capable of knowing whether a light has been turned off by a regular wall switch. Hence if any device has troubles in changing or updating its state, this might be one of cardinal reasons to look out for.

5.2.1 Example: LON Temperature Service

Instead of providing a detailed description on how a LON service is implemented, we rather visualize some implementation aspects on how we've realized a specific LON service. We take the temperature service as an example since the most other services should basically more or less match with the example provided by this section. This temperature service is hereby represented by a class called `LonTemperatureServiceImpl`
The following code corresponds to the `LonTemperatureServiceImpl class`:

```
1 public class LonTemperatureServiceImpl extends PropertyProvider implements
2     TemperatureService, LNSRawUpdateListener
3
4 //...
5
5 public LonTemperatureServiceImpl(LonCommunicator _lonComm,
6           LNSNetworkVariable _nv1, BundleContext _btxt)
7 {
8       //...
9
```

```
10      // Add custom properties that features the Temperature Service
11      // The values must be within the range of -274 and 6279.5 C'
12      FloatType floatTypeTemperature = new FloatType(MIN_VALUE, MAX_VALUE);
13      addProperty(TEMPERATURE_VALUE_NAME, "0", floatTypeTemperature, true,
14           false);
15
16      /**
17       * Retrieve inital values...
18       */
19      String initial;
20      try
21      {
22         initial = this.temperatureNV.getValue();
23
24         setPropertyValue(TEMPERATURE_VALUE_NAME, initial, "SYSTEM");
25
26         // Fire update
27         updateAllConnectedWires();
28      }
29      catch (IOException e)
30      {
31         e.printStackTrace();
32      }
33  }
34
38  public void onRawUpdate(LNSRawUpdateEvent arg0)
39  {
40      byte[] rawBusValue = arg0.getValue();
41
42      // Extract temperature
43      String temperature = ((float) SNVT_temp.getValue(rawBusValue) / 100)
44             + "";
45
46      setPropertyValue(TEMPERATURE_VALUE_NAME, temperature, "SYSTEM");
47
48      // Fire update
49      updateAllConnectedWires();
50  }
51
52  public synchronized void propertyChanged(String name, String value,
53       String _trigger)
54  {
55      // Not necessary since the temperature service does not have any
56      // writeable properties.
57  }
58
59  //...
```

Since we must adhere to a temperature service we must comply to the exposed interface called `TemperatureService` defined in the ABI Core (4). Also since we're interested in receiving changing device information, we must also implement the `LNSRawUpdateListener` interface provided by the Echelon API (See: 5.2.2). Additionally, we want to benefit from the `PropertyProvider` that provides us with the necessary properties and its change methods (`updateAllConnectedWires()` and `propertyChanged()`).
When scanning through the code snippet one might have noticed line 12-13, where the device capabilities have been set. Also take note that we don't need to implement the `propertyChanged()` method since we haven't added any properties that are writeable.
In line 46 we set the temperature status property upon receiving any valid temperature status data. After

having read and set the properties, we commit the update by calling `updateAllConnectedWires()` that one might already be familiar with ([NB05a]). Line 43 makes use of a static class called `SVNT_temp` that provides converter methods which are capable to transform any byte representation to an appropriate floating point value.

5.2.2 Libraries

As mentioned in the beginning of this chapter (See section: 5.1) a couple of third party libraries have been used to communicate to the LNS Server. These are:

- lnshmi.jar
- lonmark.jar
- stdtypes.jar

Basically only the *lnshmi.jar* file would have been necessary since it serves as a wrap-up library which should include all other functions as well, that are given by the other two jar files. Conveniently though, it has been proven that especially *stdtypes.jar* provides a very convenient way to read out the network variables and thus has preferably been used in our implementations. The *lonmark.jar* is actually needed by the *stdtypes.jar* and hence no way to go around this library.

The LNS Server is available with LonMaker Integration Tool, LNS DDE Server, LNS Application Developer's Kit for Windows ([iLO]), and any third-party product that includes an LNS Server redistribution. Since we noticed that new software updates are gradually available for download, at: *http://www.echelon.com/lns*, we recommend to check this side every once in a while since any bug in the LNS Server as well as in the API might provide some help, to increase the stability in the ABI System. By the way, the current version of the LNS Server is version **3.01**.

Part III

Distributed Agent Applications

Chapter 6

Introduction

In our term project [NB05a], we introduced the RBC API ([NB05b]) that will play a central role in the next couple of chapters. We briefly retrospect its major goals from an architectural point of view.

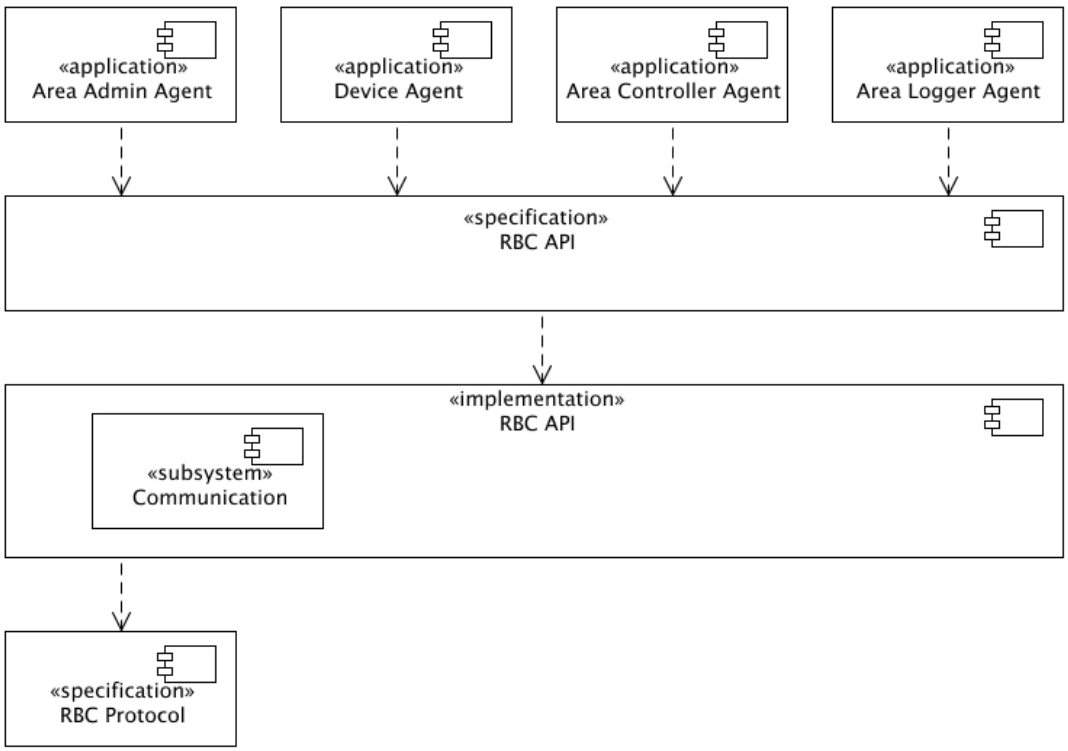


Figure 6.1: Architectural overview

Figure 6 illustrates the architecture as it has been developed in our term project. We see that the architecture is composed of three main parts. Further the bottom is constructed out of two sub-parts.
In these chapters, we concentrate on the development of custom distributed agent applications. As depicted in figure: 6, a distributed agent application does not depend on other parts but the RBC API specification.

This advantage will now be put in practice since such a simple approach facilitates the development process remarkably.

We recall that the major benefit of such an infrastructure is that any agent within the Multi-Agent-System (MAS) such as the Area Controller Agent (See chapter: 7), can now be developed independently without having to deal with the entire ABI System.

In order to test the hot plugging capabilities we developed a variety of distributed agents that will either help to test the overall infrastructure or will provide some advanced management utility functions that can be used to structure the entire building and within its interconnected network of devices. It should be clear that by such a huge environment, with lots of occupants in it, we need to consider a convenient technique to facilitate the usability of such applications. Not only the usability of the agents are meant hereby but also to achieve a low effort in installing and configuring the agents. One might wonder why such agents are needed in the first place. The answer is quite simple. Some workstations within a larger space, might actually rather correspond to laboratories then a regular office and hence suffer from directly accessing available effectors such as lights or blinds. By providing a limited view into such an environment, we can easily potentiate the manual control over those devices with web based agent application. Web-based since they are available through Java Web Start.

This part is constructed of 4 major distributed agent applications.

- **Area Controller Agent**: This agent allows any environmental part to be accessed in a distributed kind of fashion (More in chapter: 7).
- **Area Controller Agent PC Presence**: This agent combos the latter agent with an additional remote PC Presence detector software piece (More in chapter: 8).
- **Area Admin Agent**: The Admin stands for ABI System Administrator and primary serves as a global management tool with the purpose to provide a solution to centralize any ABI system security (More in chapter: 9).
- **Area Logger Agent**: An agent that logs any environmental sensory and effector information into a relational database (More in chapter: 10).

It should be noted that most of the agents will not contain any detailed information about their design and implementation since this would go beyond the scope of this thesis. Hence we skipped on doing any such documentation in order to keep the focus to more essential chapters.

Chapter 7

Area Controller Agent

7.1 Overview

This agent allows any environmental part to be controlled in a distributed kind of fashion. Tasks such as: Lifting up desired window blinds, switching on and off the lights and a whole bunch of other features are supported herewith. Since the task of the ABI System is aiming at controlling a room or even an entire building intelligently, we must first ensure that a building can be remotely controlled by a regular interaction agent also which would provide the necessary preconditions that any remote IB could benefit from when controlling a building. The reason why we do this first is because an ABI System must first stand the test of time without crashing or noticing when something has gone wrong and automatically tries to recover. Secondly most of the previous systems didn't have a proper abstraction and implementation of the domain and hence caused any IB to be very tightly coupled with the entire system which made any corrections or different strategies and algorithms unable to swap or reimplement and not to mention, testable.

7.2 Requirements

To reduce the maintenance in the ABI network we applied the Java Web Start technology to facilitate any deployment work that would elsewise be necessary to be employed for each user since it is quite hard to ensure that each user is aware about new recent updates and its proper installation again.

The Java Web Start technology works with any browser and any Web server. Each application developed for use with the Java Web Start software specifies which version of the Java 2 platform it requires, e.g., version 1.4 or 1.5, and each application runs on a dedicated Java Virtual Machine (JVM). Besides most of the machine here at the INI should have already a Java virtual machine installed by default and hence should be capable of launching such an application without a problem. More information about Java Web Start should be obtained from: *http://java.sun.com/products/javawebstart/*

7.3 Usage

The software can be obtained and launched from: *http://www.ini.unizh.ch/~snufer*, directly since its a Java Web Start application. This tool only works within the floor of the INI, only INI-Members can make use of this software since it needs necessary authorization. The major benefit by using the Java Web Start technology is that standalone Java software applications can be deployed with a single click over the network. Further Java Web Start ensures the most current version of the application to be deployed, as well as the correct version of the Java Runtime Environment (JRE).
It doesn't matter on which Operating System the Area Controller Agent will be running on. It is the re-

sponsibility of each OS to provide provide Java Web Start. In fact most major Operating Systems such as Linux, Windows and MacOS should have already Java Web Start installed by default since the JRE usually distributes and installs it by itself.

A major benefit when using Java Web Start is, whenever starting a Java Web Start application it will automatically check the corresponding website for any recent changes. If they were some, it will automatically download them appropriately. So maintenance issues should be solved herewith.

Upon launching the agent, the main screen will then prompt for the initial configuration that allows a desired area to be configured (See figure: 7.1) with.

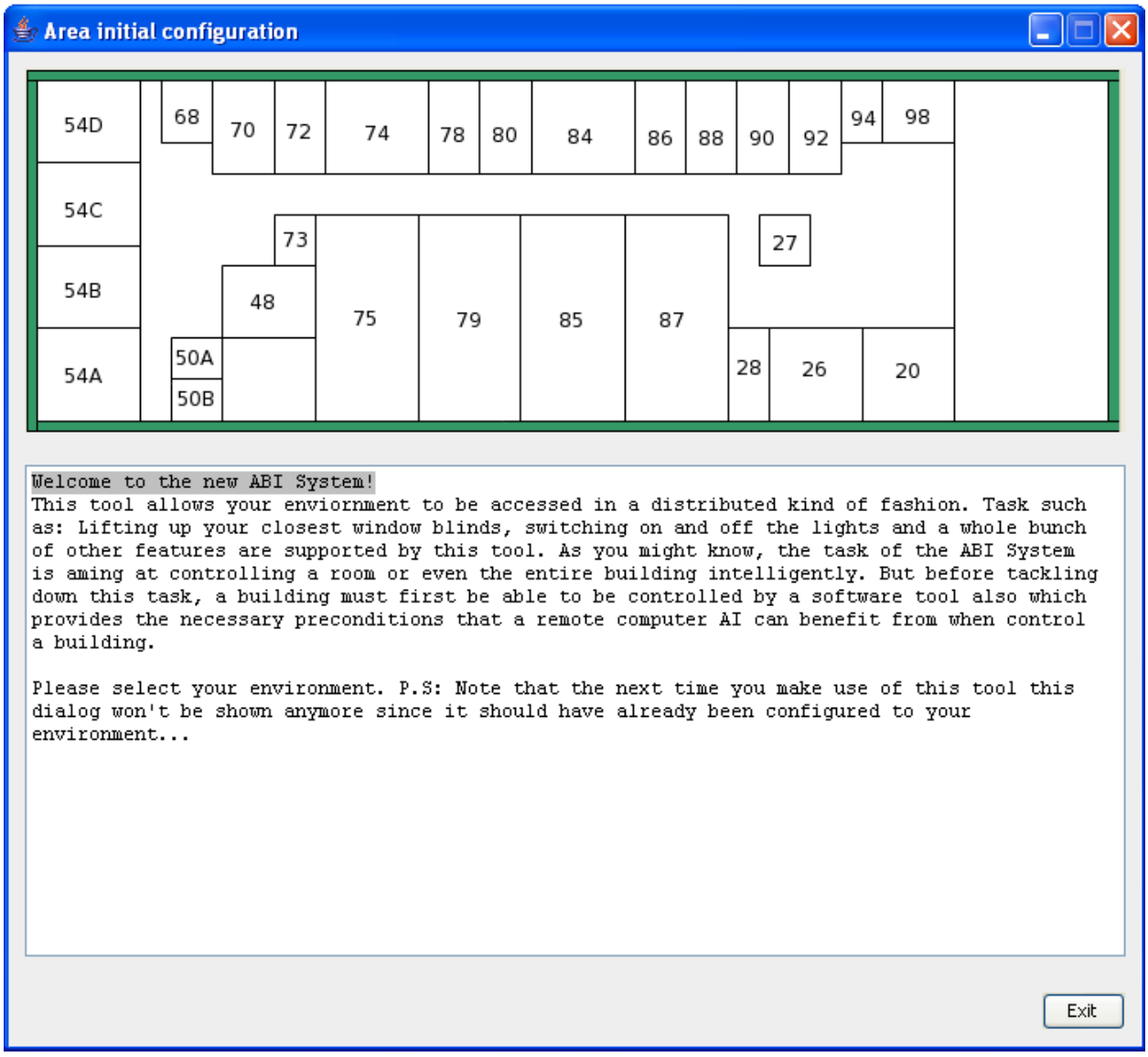


Figure 7.1: Initial Configuration

The software can be configured by simply selecting the desired area the occupant is located in. Herewith the software will try to load all environmental data (devices information, etc. currently available in this selected area) (See figure: 7.2). When the loading and initialization process (See figure: 7.2) has been completed the

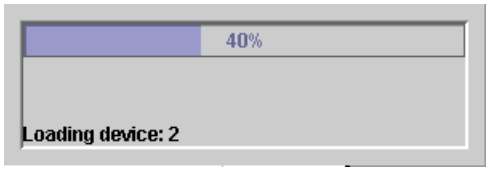


Figure 7.2: Loading devices

user will be seeing the main dialog (See figure: 7.3) that presents each device that can be accessed by an either configurable or read-only way.

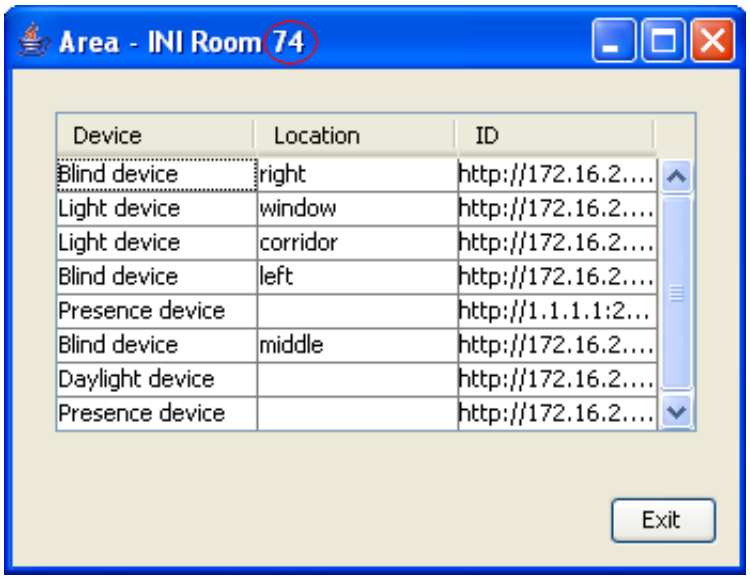


Figure 7.3: Area Dialog

Some features:

- The red circle indicates the name of the controlling entity.
- The user can sort the columns by its name by **simply clicking** on the particular header.
- The user can access the device by **double clicking** into the desired table cell.

For instance in order to access the blinds, a simple double click on the desired blind (See figure: 7.4) is required.

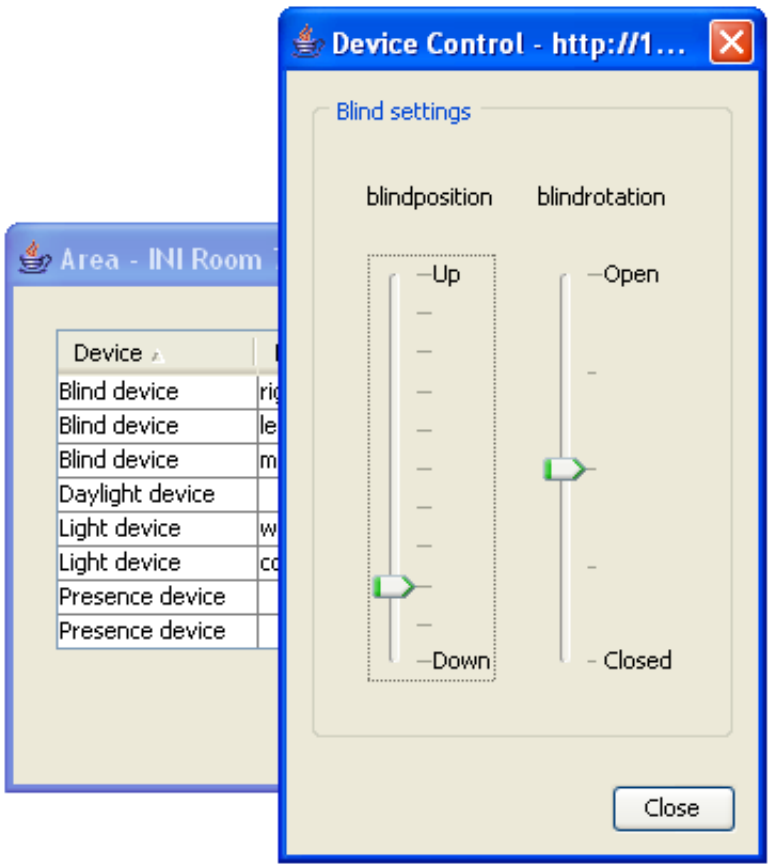


Figure 7.4: Accessing a device

7.4 Libraries

Since this is a distributed user interaction agent application, we utilized a set of libraries that we developed in our previous term project ([NB05a]). Please note this is only a cut of the current versioning since frequent updates might have been provided meanwhile.

- RBC API:
  - **Library name:** *RBCAPI.jar*
  - **Version:** 1.0
- RBC Remote API:
  - **Library name:** *RBCRemoteImpl.jar*
  - **Version:** 1.0
- RBC Multicast Receiver:
  - **Library name:** *MulticastReceiver.jar*
  - **Version:** 1.0

Chapter 8

Area Controller Agent PC Presence

8.1 Overview

This user interaction agent combos the latter listed agent with an additional Remote PC Presence detector software piece. This sensor serves as an additional person presence detector that tracks down the users mouse motions and keyboard hits. The reason of its development is the urgent need of additional presence detection due to the fact that each presence sensor we are currently working with, rather correspond to regular movement detectors which are only capable of detecting movements of persons within a certain range. Hence when movements are getting sparse or when the sensors simply don't have direct intervisibility to occupants, regular PIR sensors start to fail to correctly detect the presence of an environment. To counteract this issue so called *PC Presence detectors* have been developed that help to sense presence of a room. In order to enhance the deployment of the PC Presence detectors, we let them run as thin client applications, right underneath the regular Area Controller Agent (See figure: 8.1).

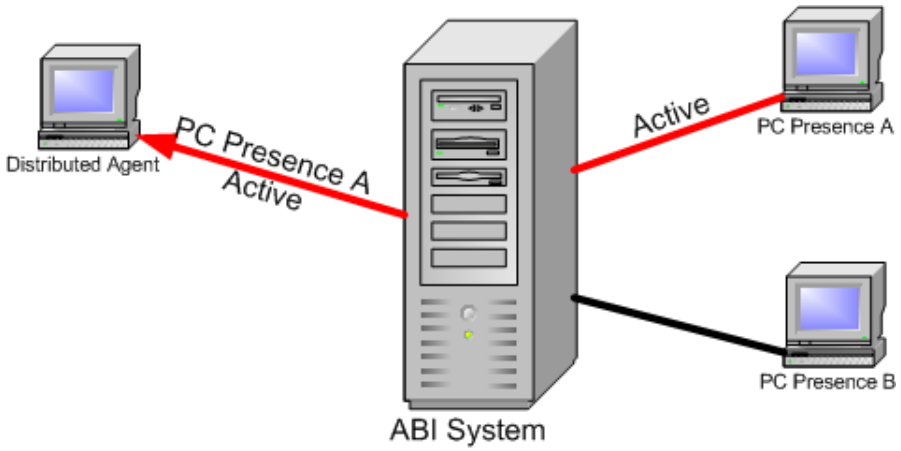


Figure 8.1: PC Presence detector ([NB05a])

Note that only a subset of people here at the INI are able to download and install this software on their machine since it is only required when performing real environment tests, that ultimately needs to provide a more reliable presence information of a room. So for the sake of simplicity and chiefly to minimize network traffic also, probably not suitable for each INI-Member.

As soon as the agent is installed and running on a user's machine, the ABI System will automatically be notified about the additional PC Presence detector.

8.2 Libraries

Since this is a distributed user interaction agent application as well, we utilized the same libraries as the Area Controller Agent application (See section: 7.4). Additionally though since we need to provide a way to retrieve presence information of a room (See figure: 8.1) three extra libraries have been used that contain native code that is capable of sensing presence on three major platforms: Linux, Windows and MacOS.
The libraries have been written by Tobi Delbruck at the Institute of Neuroinformatics (INI) [Ins]. In order to webstart the application a wrap-up library named *native.jar* was necessary to provide. Elsewise, it won't not be downloaded and therefore causes the agent not to function properly.

- Overall Library:
  - **library name:** *native.jar*
  - **Version:** 1.0
- Linux:
  - **Library name:** *libpcoccupancylinux.so*
  - **Version:** 1.1
- MacOS:
  - **Library name:** *libpcoccupancymacosx.so*
  - **Version:** 1.2
- Windows:
  - **Library name:** *PCOccupancyWindows.dll*
  - **Version:** 1.3

Chapter 9

Area Admin Agent

9.1 Overview

The Area Admin Agent stands for ABI system administrator agent and primary serves as a global management application with the purpose to provide a solution to centralize ABI system security. Tasks such as supervising rooms, make new device assignments or creating new areas and devices are ordinary jobs done with this user interaction agent. Since it provides global access rights it possesses the authorization to perform any actions that are possible within the range of control.

Due to the fact that this agent is quite mightiness, only a small distinct subset of users should be authorized to gain full access to the ABI System.

9.2 Usage

This section gives a brief overview on how to use the Area Admin Agent. It explains the most important features and screen elements.

Speaking in favor for the Java Web Start technology, this agent has also been implemented with the option to be deployed using Java Web Start. The software can be obtained and launched from:
http://www.ini.unizh.ch/∼snufer.

Having once obtained and started the application, the occupant will be prompted to connect to the ABI System (See figure: 9.1).

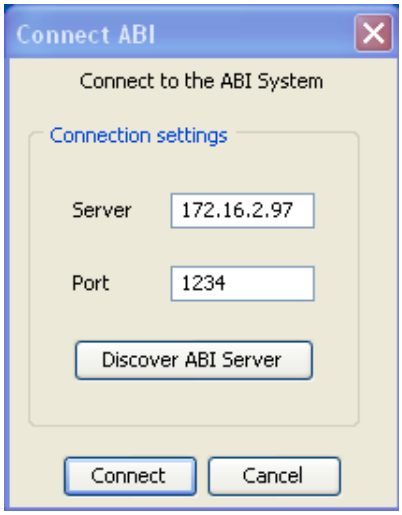


Figure 9.1: Connecting to the ABI System

Since the ABI System can be deployed in any place possible within the network and gradually might make ip address changes as well, an automated discovery function has been incorporated. If it should happen to fail it is likely that the network is either down or the corresponding router doesn't allow any multicast packets to be forwarded. For such cases the appropriate hostname or ip address and port should be entered manually.

If the connect was successful, the user is free to perform any kind of action such as assigning a new device to a location. Two main screens offer the necessary options that the occupant can choose from:

- The *Show Areas* window provides a list of all available areas (See figure: 9.2). Common tasks that can be accomplished herewith are: The assignment of new devices to dedicated areas such as mobile presence detectors (i.e. Falcon Presence Device ([NB05a]). Similar when new areas need to be created or even deleted to simulate a non-stationary environment, this is the place to perform such configurations.

- A full list of all devices, currently known to the ABI System can be obtained by selecting the sub-menu called *Devices* (See figure: 9.2).
  When new devices should be made public, it must first be created by simply selecting the appropriate menu called: *show devices* (See figure: 9.5). The red colored device addresses hereby indicate that such devices are currently not registered to the ABI System. Hence black colored device url's denote devices as registered (Ready to provide sensory information or changing its device state such as lifting up the blinds).

The occupant can further zoom into each area by double clicking into the desired fields (See figure: 9.3). All devices that have been assigned to an area will hereby be loaded into the local ABI System. It's important to note that upon performing such an action all devices are automatically registered to the ABI System since each of the device needs to be aware of its capabilities (i.e. If a device possesses the capability to measure the daylight or presence). Through the low coupling that the ABI System brings along this is a required process and a very important aspect to keep in mind.
Additional features such as supplying a device with a detailed location such as window or corridor might provide some help for users to identify their favorite devices quite faster. The change of a device location can be accomplished by using the corresponding button *device location*.

The device assignment can be initiated by dragging any device from the overall device window (See figure: 9.5) into the dedicated area window by simply dropping the selected item into the appropriate spot.

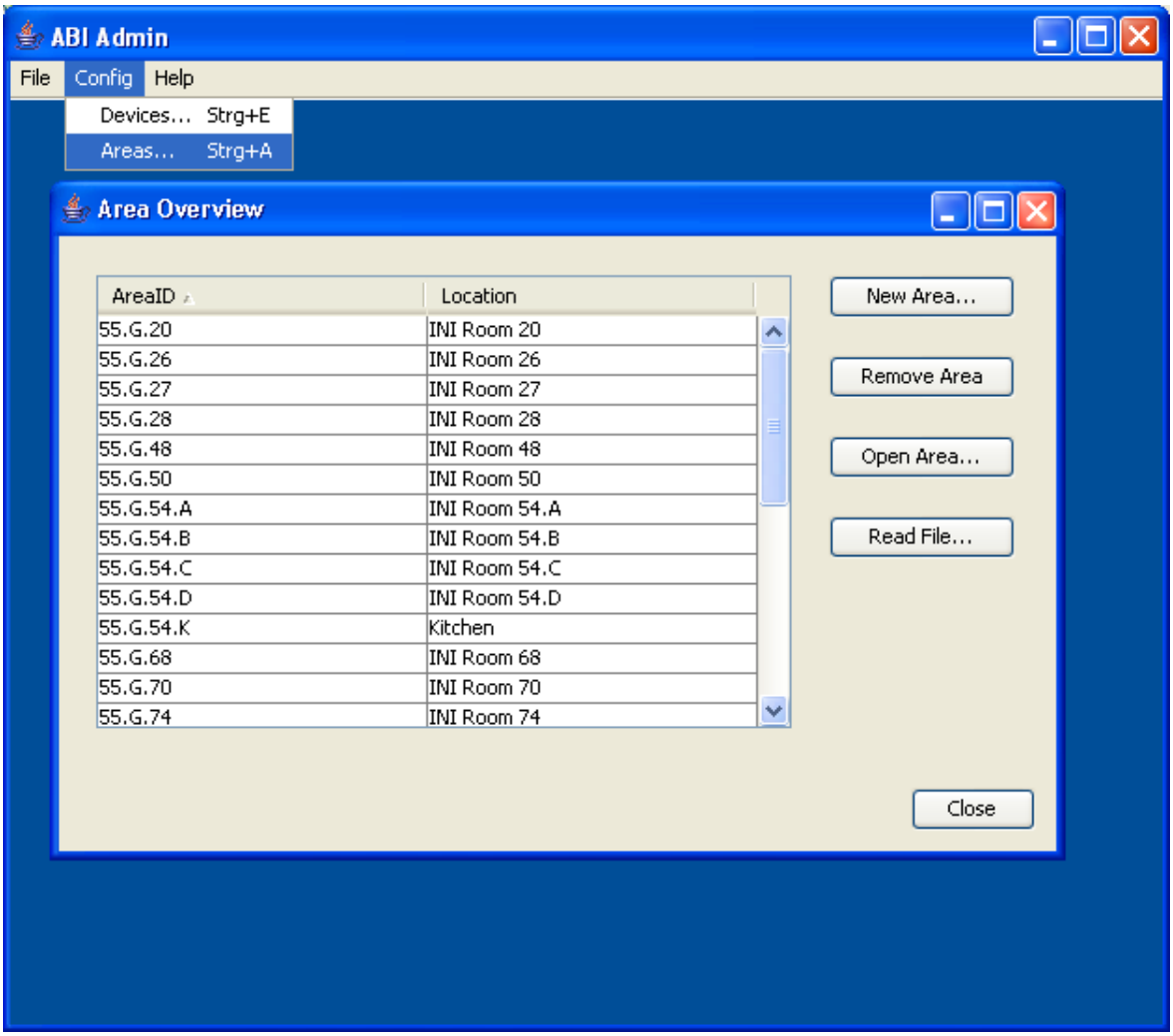


Figure 9.2: Show all areas

Similar to the Area Controller Agent (See chapter: 7.4) a device can be accesses by double clicking into the appropriate field (See figure: 9.4).

9.3 Libraries

Since this is a distributed user interaction agent application as well, we utilize the same libraries as the Area Controller Agent application (See section: 7.4). No third party libraries have been used.

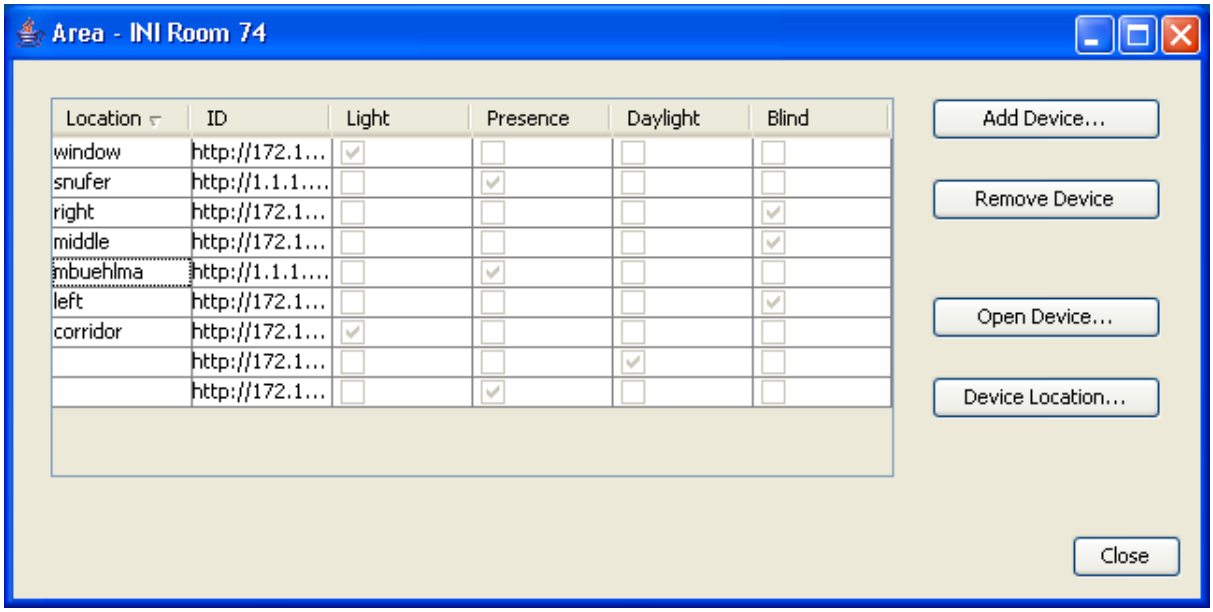


Figure 9.3: List currently registered devices within an area

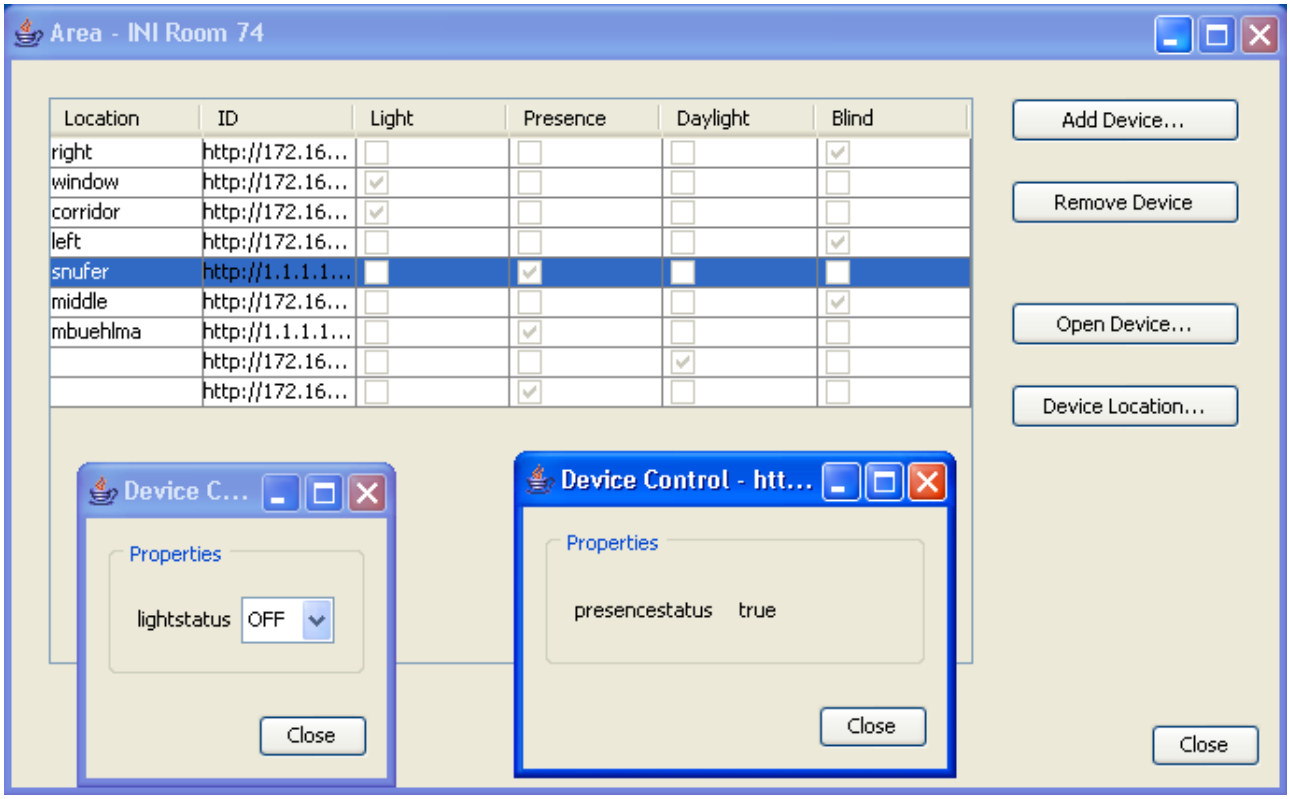


Figure 9.4: Accessing some devices, i.e. a light and a presence detector

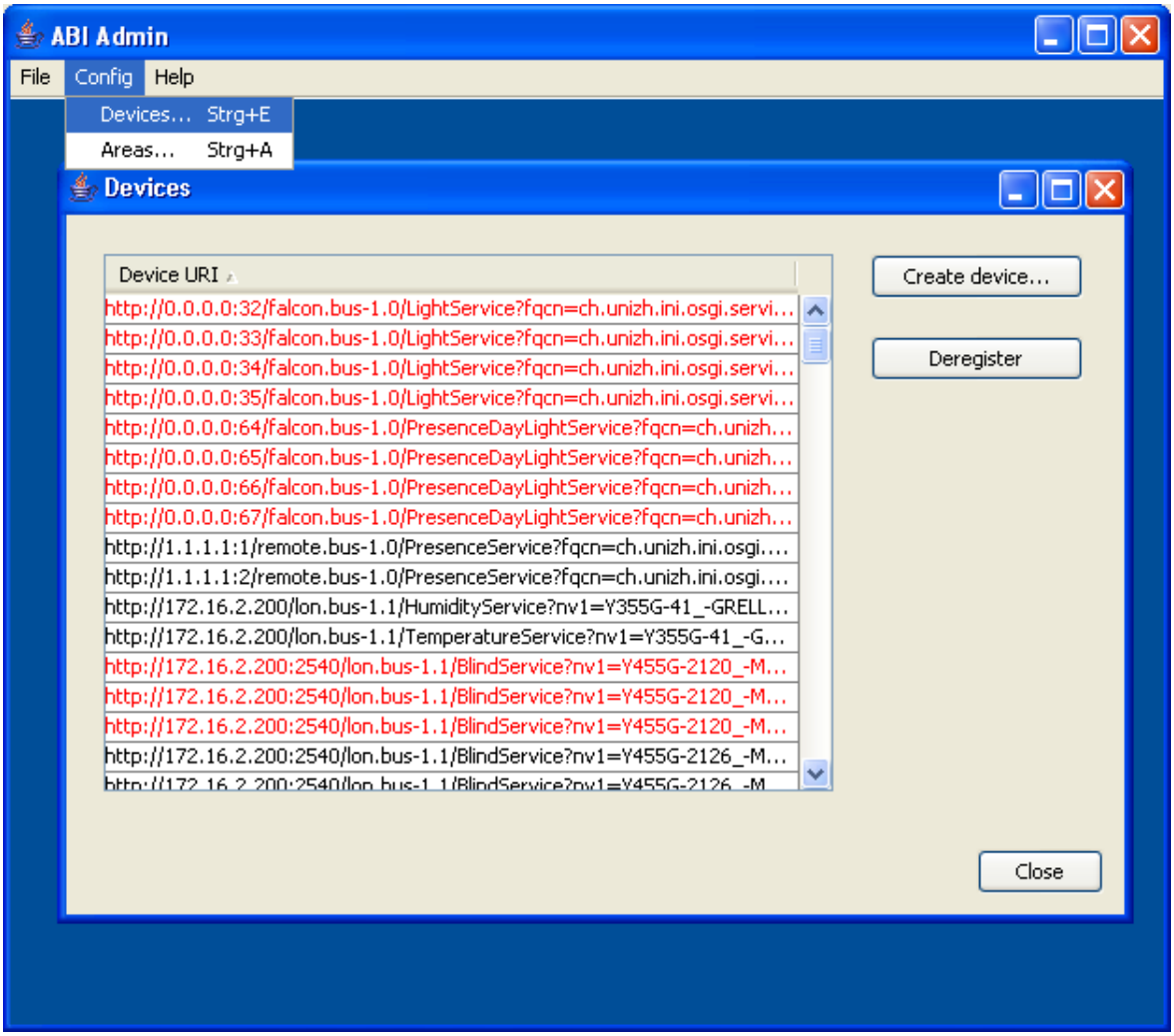


Figure 9.5: Show all devices currently known by the ABI System. A device that is not yet registered, is marked red and hereby doesn't receive any changes.

Chapter 10

Area Logger Agent

10.1 Overview

Our predecessors ([BG04a], already mentioned that a database should log any events within a specified building part, in order to be capable to replay their collected data. Unfortunately this is only practical to a certain point of degree since only sensory data can be replayed. Other data would actually depend on either IB interventions and thus from resulting user interactions as well.

Nevertheless it is crucial to provide such a mechanism, in particular for data mining or simply for plotting daylight curves, etc. Without such a possibility the ABI System would suffer from losing this knowledge and would be a major drawback when trying to validate any interacting intelligence. With reference to our concept that we introduced in our term project ([NB05a]), the domain model (See figure: 10.1) of the RBC API ([NB05b]) can basically be adapted in order to transform it into a corresponding relational data representation.

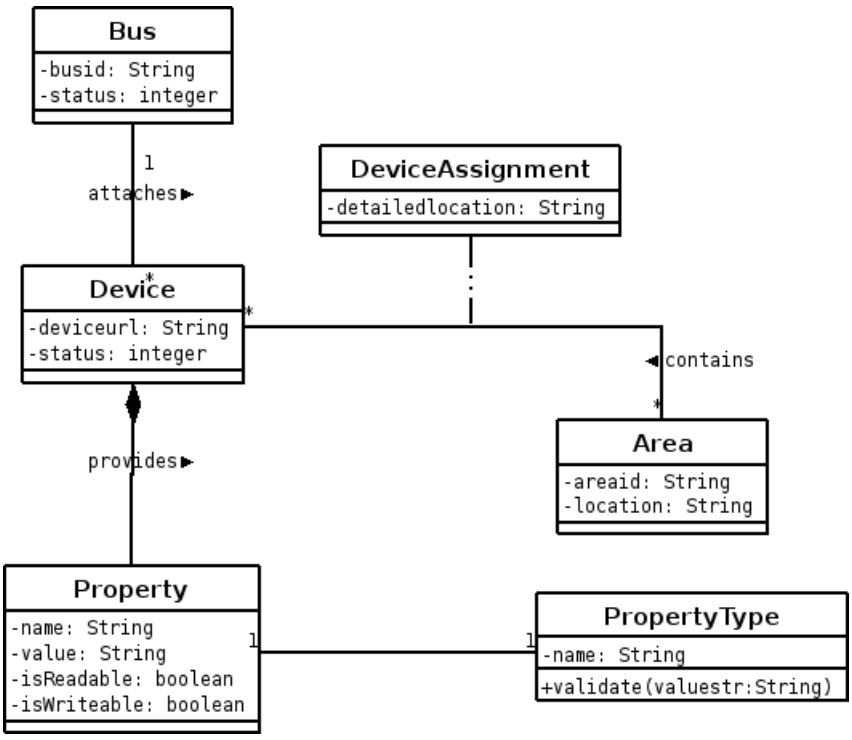


Figure 10.1: The domain model of the local ABI System ([NB05b])

10.2 Relational Model

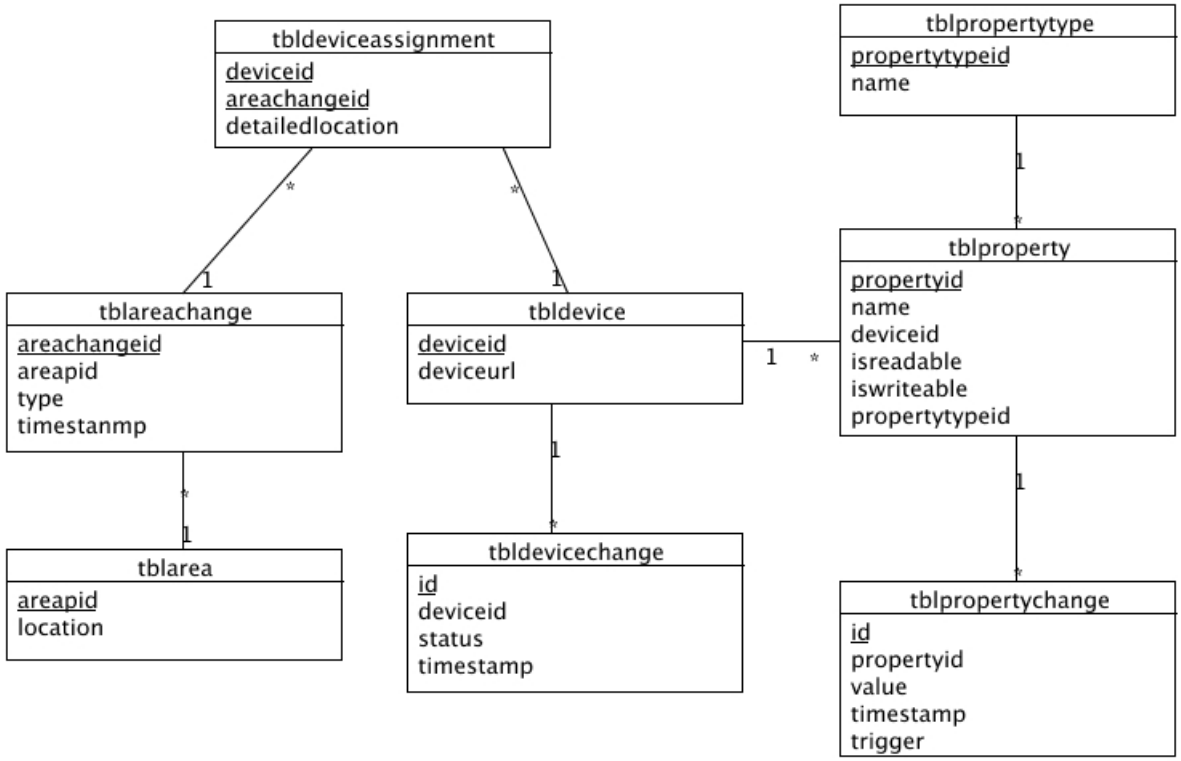


Figure 10.2: The Relational Database Model

As depicted in figure: 10.2, additional concepts needed to be introduced due to the fact that certain things might need to be added dynamically as a result of the *non-stationary* environment.

tblpropertychange One such dynamic concept are property changes, since every property in all likelihood, performs changes every once in a while. This is the reason why an additional table has been introduced that tracks any possible state changes. Hereby *trigger* corresponds to the initiator of the state change. For instance if the originator happens to be the Area Controller Agent application, the trigger will correspond to the user name of that occupant which initiated the state change.

tbldevicechange Since devices are gradually registered and sometimes unregistered as well, such information must be stored since they might play a central role in any event evaluation.

tblareachange Another concept that is needed in order to allow any device to be relocated, added or deleted from any area, an additional table that tracks any such movements within an area was needed to ensure a compact knowledge reproduction. It would be severe if a device would all the sudden lose the relation to its data upon deleting or relocating a device. In order to circumvent such anomalies, a supplementary attribute called *type* has been used to state devices, that have prior been initialized to an area, to *INIT* and further dynamic changes to:

```
enum('INIT', 'AREA_CREATED', 'AREA_REMOVED', 'DEVICE_ADDED',
  'DEVICE_REMOVED', 'DEVICE_LOCATION_CHANGED')
```

The other tables should speak for themselves since they can basically be flatten down to a relational data representation without any problem.

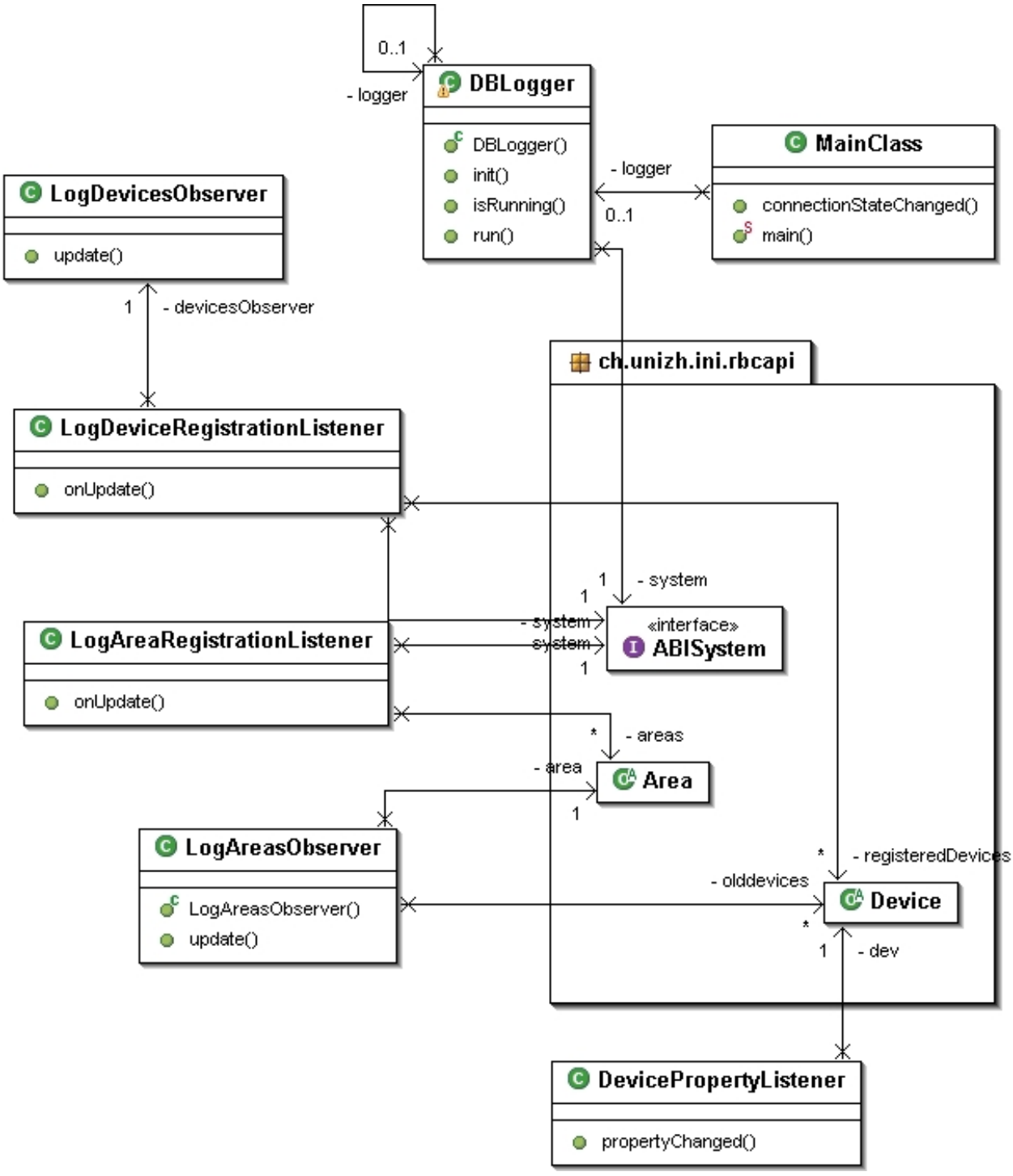


Figure 10.3: Class diagram of the ABI DB Logger

The following documentation part is intended for developers only since it explains how the relational model impacts the application as a whole.

According to figure: 10.3 the logger application initializes upon calling the `init()` method. Herewith two concrete registration listeners are created (`LogDeviceRegistrationListener` and `LogAreaRegistrationListener`). Subsequently each one of them performs individual initializations such as loading all known areas and devices into the local ABI System (proxy). The initializations are conducted onto the database as well (Unknown devices or areas are created and linked up).

To comply to the problem of the non-stationaryness, each of the registration listener classes provide a method (`onUpdate()`).

So for the `LogAreaRegistrationListener` this method gets called-up when new areas have been either created or removed (for instance through the ABI Admin (9)). If an area is new to the underlying API ([NB05b], 10.3) we don't physically need to create appropriate database entries for it. Further we need to log any change whether the area has been either created or removed. Note that areas are never removed in the sense that they are being deleted in the database completely since we need to provide access to past data even through a long period of time. This might be worthwhile because any data can now be analyzed in form of a history.

The `LogDeviceRegistrationListener` deals with the dynamics quite similar. In contrast to the `LogAreaRegistrationListener` though, it will only be called when new devices are either registered or unregistered or initiated upon calling `showAllDevices()` by an area. Generally speaking a device is created when the remote API recognizes a new device in its internal representation ([NB05b]).

In order to receive any device changes (i.e. when a device has been registered or unregistered at runtime), appropriate listeners must be registered accordingly. Hence the registrator listener classes are the candidate for such kind of tasks. The `LogDeviceRegistrationListener` is responsible for registering the `Observer, LogDevicesObserver` that will be notified about each device state change (registered or unregistered).
Analogical the `LogAreaRegistrationListener` must register area observers (`LogAreasObserver`) for each new area that is detected within the ABI System. With reference to the property concept ([NB05a]) each device owns a set of properties that can be set or read out. Since most properties may change over time we must ensure that property updates are being forwarded to a class which does all the logging for it. In order to receive any device changes, i.e. a new presence status, appropriate observers must be registered. Inevitably this forces the `LogDevicesObserver` to register an appropriate, separate listener class called `DevicePropertyListener class`, which in turn implements the well known public interface (`PropertyChangeListener`) provided by the RBC API ([NB05b]).
The following code snipped has been taken out of the `DevicePropertyListener class`:

```
1  public void propertyChanged(Property property, String value,
2          String trigger, Date time)
3  {
4     // Prepare the timestamp
5     SimpleDateFormat timeformat = new SimpleDateFormat(
6             "yyyy-MM-dd HH:mm:ss.S");
7     String t = timeformat.format(time);
8
9      // ...
10
11     // Obtain the propertyid
12    String query = "SELECT * FROM " + DBLogger.TABLE_PROPERTY
13            + " P" + " INNER JOIN " + DBLogger.TABLE_DEVICE
14            + " D ON D.deviceid=P.deviceid" + " WHERE P.name ='"
15            + property.getName() + "' AND D.deviceurl="
16            + "'" + dev.getDeviceString() + "'";
17
18     Statement st = conn.createStatement();
19     ResultSet res = st.executeQuery(query);
20     res.next();
21     int propertyid = res.getInt("propertyid");
22     res.close();
23     st.close();
24
25     // Log the property change into the TABLE_PROPERTYCHANGE
26     query = "INSERT INTO " + DBLogger.TABLE_PROPERTYCHANGE
```

```
27            + "(propertyid, value, timestamp, trigger) VALUES ("
28            + propertyid + "," + "'"
29            + property.getValue() + "'" + "," + "'"
30            + Timestamp.valueOf(t) + "'" + "," + "'" + trigger + "')";
31
32    st = conn.createStatement();
33    st.executeUpdate(query);
34    st.close();
35
36    // ...
37 }
```

It quickly demonstrates how property changes are flushed into the database.

10.3 Libraries

The underlying database that we've chosen is MySQL. Its large cross-platform support was one of the major reasons that we decided on using MySQL. Secondly, since its open source, several concrete JDBC implementations exist, which provide a convenient way to access the database across the standardized JDBC interface.
Another database that we also considered on using was PostgreSQL. But since we don't really need the foreign key feature and neither stored procedures nor triggers, we skipped that thought. Furthermore we had already experience using a MySQL database from Java.

Since this application is a distributed agent application as well, we utilized the same libraries as the Area Controller Agent application (See section: 7.4).

- MySQL database server **version:** MySQL 4.0.21
- JDBC API:
  - **Library name:** *mysql-connector-java-3.1.10.jar*
  - **Vendor:** Sun Microsystems Inc.
  - **Specification-Version:** 3.0

Part IV

Intelligent Learning System

Chapter 11

Introduction

The previous parts (II, III) were rather emphasizing in adding new features to the ABI System or covering topics such as the development and usage of different distributed agent applications such as the Area Controller Agent (See chapter: 7) or the Area Admin Agent (See chapter: 9).
No doubt, from a computer science point of view interesting, but to tell the truth they were rather additives in contrast to the subject of the next following parts (IV, V, VI). Additives in the sense that they're actually substantial for an IB to function properly (i.e. the PC Presence sensors, LON devices).

11.1 Part Structure

This parts is composed of two major chapters: An Introduction chapter: (Chapter: 11) and another chapter that presents the Intelligent Building Framework(IBF) (Chapter: 12).
In the introduction chapter we basically give a summary about the state of affairs and retrospect and discuss the term building intelligence.
We then dive into some crucial details that need to be known, when developing any building intelligence. Topics such as input pre-processing as well as sensor issues are meant hereby. We also explain why it is worth to investigate methods, that try to discover correlations among a categorical set of sensory inputs as well as a way of storing exceptional data over longer periods of time. The major gain that we hope to achieve, is a new approach to perform *ambient noise filtering*, that we belief is one of the most powerful methods to maximize the accuracy of the predictions.
In order to get to know our environment we performed a very intuitive environmental analysis that proves that an environment can basically be subdivided into two different categories. - *Simple and complex environments.*

The second chapter then introduces the Intelligence Building Framework (IBF) that contains a decision controller (DC) that is based on an algorithm that allows multiple prediction algorithms to compete against each other.

11.2 Building Intelligence

> *An intelligent building is one that doesn't make the occupants look stupid.*
> (Sam C M Hui, Department of Architecture at the University of Hong Kong)

The University of Hong Kong further distinguished between two major goals of Building Intelligence.

- Maximizing the support of occupants by providing an effective resource management with a minimum life cost.

- Adaptive to changing user needs and changes within the environment such as its structure.

Hereby the common objectives are:

- Responsive (to user needs / to climate)
- Efficient (building design and systems)
- Effective (operation and management)
- Better integration (with IT and within systems)

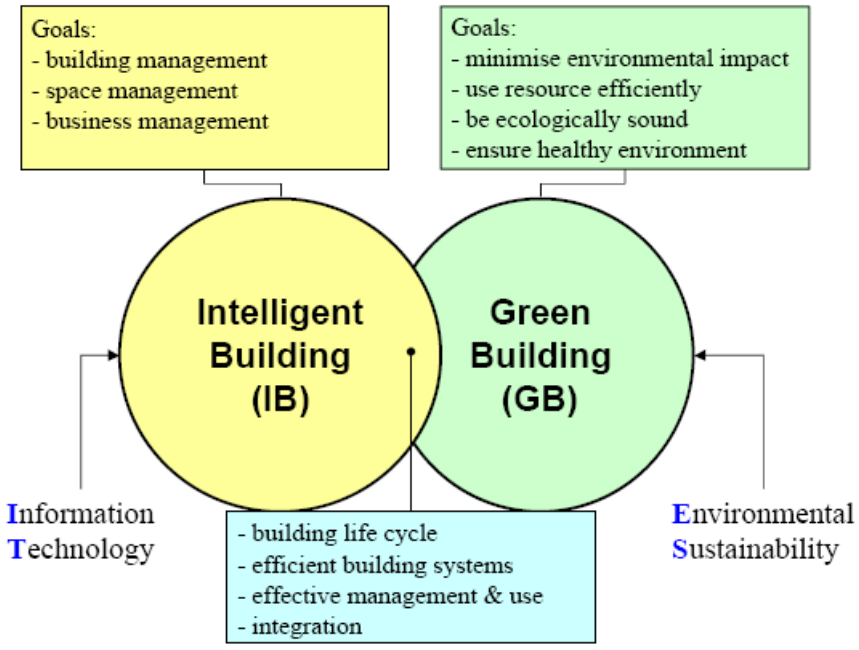


Figure 11.1: Intelligent Building ([BUI])

At first glance this looks quite logical but in some sense impractical since it doesn't actually point out the difference between for instance existing, regular daylighting control systems.
The evolution of control systems can, according to a related project called LESO ([LES]), be derived as depicted in figure: 11.2.

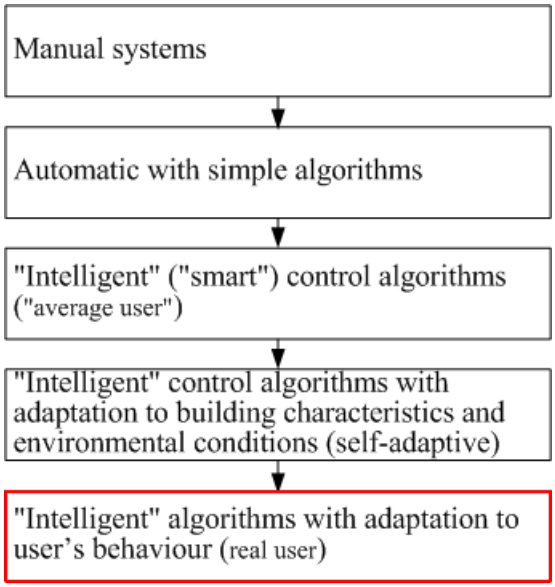


Figure 11.2: Evolution of control systems ([LES])

- Manual systems involve no control algorithms: building services are directly controlled by the users, either mechanically or using electrical actuators.
- Simple control algorithms are widely available on the market today. These algorithms are using very simple rules i.e. Protecting the window from strong winds
- "Intelligent" (or "smart") control algorithms include a more refined control strategy, for instance considering the detailed geometry for a blind, or predictive algorithms for the heating or cooling. They allow to provide an optimal comfort level for an "average user", and need a careful tuning at the commissioning phase.
- "Intelligent" control algorithms adaptive to building characteristics include similar rules than the previous category, but they allow to make the commissioning easier through self-adaptation to building characteristics, building use, and environmental conditions like the climatic situation.
- "Intelligent" control algorithms adaptive to user's behaviour represent a significant step in the evolution: they allow an adaptation to a real user (and not just an "average user"), thus minimizing the risk of rejection by users frustrated by an algorithm that would go against their wishes.

Our goal is (as already mentioned in the introduction part (See chapter: 1)), the development of the IB that is adaptive to real users and also one that takes continuously fluctuating climatic factors into account.

Ueli Ruthishauser and Alain Schaefer ([RS02]) already introduced that a building behaves from a computational point of view completely nondeterministic. Nondeterministic in the sense that individuals that we also call agents interact with the environment.

According to the aspects stated in the introduction part (See: I) of this thesis, one of the major goals is to discover a fundamentally different approach of making a building intelligent.

For weeks we had been struggling with the fundamental question how to improve the reliability of environmental predictions. Speaking in forecast we refer *reliability* as a major component that should only make

interferences when a set of sub-predictions have been consulted to form a collective hypothesis (See: 12). The major drawback that we encounter, is that we can only produce accurate sensory predictions if we know what actions are being performed from occupants. Naturally the resulting behavior is very sparse in nature and in some way also diffusing with exterior and interior environmental changes. Inevitably, by just providing the daylight value as the measure, probably won't be sufficient enough to successfully control all areas within a building (See 11.3).
One of the major problems that we also need to face is that a building intelligence must be capable of losing short as well as long term knowledge upon receiving different patterns of activity. For instance each device agent within a controlled office that is frequently visited by the same kind of occupants, may suffer from not being able to lose such an acquainted, acquired knowledge upon sudden changing habits.

It is often convenient to gather the input variables x_i together and denote them by a single vector $\vec{x} = (x_1, ..., x_d)^T$ where d is the total number of such variables, and the superscript T denotes the transpose. When observing the input variables that may be a mixture of sensory as well as effector outputs, we notice that certain variables may need to be *pre-processed* first in order to be useful for the device agents to work with. In the most simplest case, *pre-processing* may take the form of a linear transformation of the input data, and possibly also of the output data that is also known by the name *post-processing*. More complex *pre-processing* may even involve a reduction of the dimensionality of the input data. However such dimensionality reduction or cut might lead to performance issues or information loss, that needs to be considered of taking into account (depending on the variables).

When observing the daylight value for a while one will notice that regular humans do not distinguish between a certain level of luminance anymore (e.g 3000 or 8000 lux) since most occupants would by far start to feel uncomfortable and would respond with a set of opposing actions. For instance moving down the blinds.

Thus we conclude, upon scaling such an input value to its maximum, we face the problem of inaccuracy due to the fact that lower values may contribute or carry more pivotal information to allow a proper prediction to be made. Hence it may be appropriate to relativize such input data to enable a better suited scaling that can be more useful. Hence when taking the daylight as an example, it may be advisable to re-scale the values into a logarithmic representation $y_i = \log(x_i)$. Whereas y_i denotes the new input variable.

Additionally, certain input variables such as the daylight for example, might partly be unsteady or shaky which might lead to inconsistent predictions to be taken. Such characteristics can be damped by providing filtering.

For instance in order to improve or soften the input data $\vec{x}$ we might consider to integrate and average across a certain distinct number of input data x_i that have been collected within a time period δ.

$$y_i = \frac{1}{\delta} \int_{t-\delta}^{t} x_i \, dx_i \tag{11.1}$$

With such an additional new vector $\vec{y}$, a building intelligence might take advantage of since a steady input is more beneficial for device agents to maximize their predictions.
In general it shouldn't actually matter what kind of inputs we are providing to the device agents. However we've noticed that this isn't true due to the fact the certain algorithms might start to learn things we haven't really wanted them to. Tasks such as learning to switch off the window light upon switching on the corridor light. This might sound funny but it did happen.
Therefore care must be taken which of the inputs should be avoided in the first place and which of them rather enhanced. One way of enhancing has already been stated above but we can do more. It isn't always practical to directly use each input x_i and to scale each one of them. Furthermore we might also consider to provide additional inputs, such as a sum-up over past values or maybe even start to combine different input variables together. Therewith we could improve the predictions remarkably since it will be able to recognize when a strong input, such as the daylight has slipped by an instant light reflection. The problem though is that it isn't always apparent when to provide additional "synthetic" inputs. But in general they will help to avoid mis-predictions. Incidentally, it is always practical to provide some kind of backup inputs because we always need to consider that inputs may return just crap or even drop out completely.

In regard to long and short term knowledge keeping ([TZ03b]), we might find it hard to designate when to evict old knowledge and when to acquire and incorporate new knowledge. After all it would be a pity to lose knowledge that will be appearing anytime soon again. On the other hand, just continuously collecting huge amounts of data and carry out the predictions is less beneficial since new knowledge should appropriately weighted stronger then old knowledge. Furthermore an intelligence that is fully loaded with similar data will quite often suffer from making a clear prediction due to the fact that crucial inputs will basically will get swallowed and drowned by the number of irrelevant and redundant data with less distinction (See Shannon on information theory [Mit97], [Sha48]).

Evidentially we need something that is more adequate to filter-out essential input values among a giant amount of inputs which are almost negligible. The bottom line is to first pre-process all sensory inputs $\vec{x}$ in some way that some prior distinction could be gained from, in order to enlighten any future predictions.
Above all when being once competent to distinguish between relevant from insignificant inputs we could think of incorporating a knowledge representation that goes far beyond a regular short and long-term memory. Namely the storage of sparse, exceptional input data that has been collected even across decades. Thereby we would potentiate a reproduction of such collected data which will help us to enhance the reliability of the predictions, by reacting to sudden changing conditions quite faster.
For instance on a fogy day we encounter relative humidity values way above 95% and thus would provide us a lot of additional information on how occupants have been reacting upon it. Like switching on, one or even a row of lights (See figure: 11.3).

In contrast to a sunny day (See figure: 11.4) where we might rather tend to move down the blinds a bit.

Figure 11.3: Picture taken on the 9th of November 2005, from the balcony of Room 55.G.74: humidity: 95.16%, temperature: 6.66 ℃, inner daylight: 1065, blind-pos: 10, blindrot: 2

Figure 11.4: Picture taken on the 17th of November 2005, from the balcony of Room 55.G.74: humidity: 56.41%, temperature: 9.65 ℃, inner daylight: 886, blind-pos: 80, blindrot: 1

On top of that, most daylight sensors are more sensitive to the outer daylight rather then the light that is given up from interior synthetic lights. Nevertheless it is crucial to deal with such an environment and with so called "bad sensors", by considering the fact that certain sensors might even completely drop out due to some hardware or software problem.
Furthermore we can not presume an environment that is exposed to a window and neither we can presume an environment that is located in a basement with sparse outer daylight. Considering those facts we can identify that the need of a building intelligence is even more desired since it must try to detect which of the sensory inputs are irrelevant and which of them contribute and carry pivotal information in order to maximize a hypothesis to perform an effector state change.
Such a filtering process we refer to as *ambient noise filtering*. To give a practical example we assume an environment that is located in the middle of a building and hence the daylight sensors may hardly predicate whether to switch on and off the lights since from a time period point of view barely makes any jumps in its values. Thereby some other sensor need to take over the task to empower a reliable measure. In example the day-time, when a light is usually switched on or off. In such extreme cases it is difficult to make a proper prediction since most of the sensory inputs depend on radiation, temperature or even the humidity.

Particularly in such cases, sensory inputs rather stay in conflict with relevant sensory inputs and thus might rather tamper clear propositions that would have been possible. The most obvious approach is, what frankly speaking any artificial intelligence should try to discover are correlations such as linear dependencies among a categorical set of sensory inputs.

With reference to the motivation (See: 1), we further conclude that it isn't immediately apparent how pervasive a prediction needs to be, due to the fact that we human who live within such an environment have moving goals in such a way that regular lighting control system fail to make a proper decision.
In a certain extent we can put the blame on the sensors because they perceive and react totally different then the human brain does.
Then what we perceive is a combination of what we sense and what our memory-derived predictions tell us ([JH04]). Quite often we perform actions that we do all the time and anyhow pay little attention to.

11.3 Environmental Analysis

In a first row we conducted an investigation based on the assumption that certain environments might give us some insights about different habits of occupants, before mindlessly starting to develop any arbitrary learning algorithm that we cannot pre-estimate, at least to a certain degree, about its possible succession in the first place.

With an environmental analysis we proved that certain environments underlay a row of clear routines which can be approximated by just providing a few "strong" input variables. Such an environment we refer to as: *Simple Environments*. Vice versa, we denote environments that are more complex in nature and thus might need more input variables to state a prediction, as: *Complex Environments*.

Predicting a *Simple Environment* might involve:

- Recognize, that occupants rather tend to keep a fixed daily schedule, i.e. Starting to work at eight o'clock.
- Detect, whether occupants turn their lights on all the time.
- Notice that certain occupants rather tend to be light lazy, i.e. If the lights are once turned on, they barely get switched off again.
- Notice if occupants forget to turn off their lights upon leaving the office.

First we describe the assembly of our analysis and then illustrate the results in form of graphs (See subsection: 11.3.1 and 11.3.2), followed by a discussion section (See subsection: 11.3.3) where the results are presented, summarized and concluded.

Hereby the analysis assumed that the *daylight* and the *daytime* play a central role in certain environments. Herewith we tried to identify certain recurring patterns of activity within the daily cycle, which will give us some information on the behavior of inhabitants.

Depending on the location of an environment and its housing occupants, we can derive that certain occupants, either have a favor for lights or rather for the blinds.
Moreover the results clearly shows, that certain lights were on during entire days whereas other lights were on all the time. To get to know our environment we conceived a thought experiment to help us to convey our assumptions at that moment and performed two different statistical analysis in two different rooms.

The first analysis employed an analysis on a timely basis whereas we recorded, whether a light was either on or off. Thereby we splitted the entire day into small time episodes, whereas each such slice was responsible for counting whether the light was on or off during this slice (See: 1).
It has to be remarked that we periodically polled for the light status (Every 10 minutes). Hence, the episodes

have preferably been set to 10 minutes also.

Result: Light ON/OFF statistic per time slot s
foreach *slot* s *in* S **do**
  create history H_s with a fixed length l
  foreach *entry* h_j *in* H_s **do**
    $h_j = 0.5$
  end
end
input: A set of continuous input-pairs $\Psi\ \{x, y\}, x \in \mathrm{S}, y \in \{true, false\}$
repeat
  if $y = true$ **then**
    $q \leftarrow 1$
  else
    $q \leftarrow 0$
  end
  put q into H_x

  $\sigma_x \leftarrow \frac{1}{l} \sum_{k=0}^{l} h_k, h \in H_x$ whereas σ denotes the probability H_x of slot x

until *Stop condition reached*

Algorithm 1: Simple statistical analysis

By skipping the history and instead establishing a continuous probability function we can derive a graph, that gives us some more meaningful results to discuss. The reason why this more suitable in this case is because we need real probabilities to work with rather then a discrete bar chart that shows trends of a room. Additionally it is quite important to know how good we can rely on our statistic as well.
Thus we need something similar as an α (probability of error) that gives us some information on how much we can trust the probability at time slot s_t.
Speaking by example, if a light is accidentally once switched on for some reason, i.e. by a security officer, the corresponding time slice episode will be increased. Probably even straight to one since we presume that it's been the first time. For such cases we need a corresponding trust graph that carries the necessary information on how far we can trust this probability at time slot s.

The trigger of the second analysis was found by trying to discover, within which daylight range a light was usually on or off. Thereby the daylight range has been scaled logarithmically (See 11.2) and hence also split into slices as well (daylight slices s_d). The gained probability and its associated trust value can be derived according to the schema described above (analogical to the statistical analysis on a daytime basis). Care must be taken by providing the daylight results since in contrast to the daytime analysis, the slots may be scored several times a day since we periodically poll for the daylight-values just as well.

It's important to note that measurements have only been taken when someone was present in the specified environment since we don't actually need to collect any data when having prior knowledge on a room's presence status.

11.3.1 55.G.84

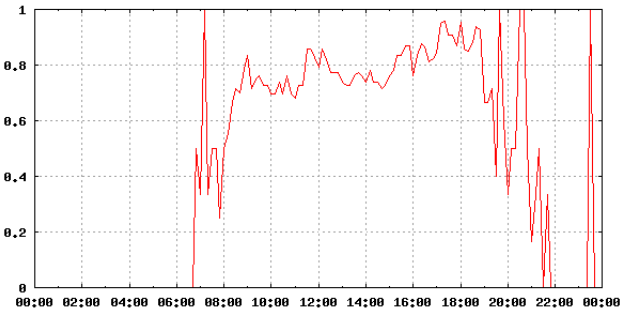


Figure 11.5: $\mathcal{P}(CORRIDOR\ LIGHT\ ON|s_t)$

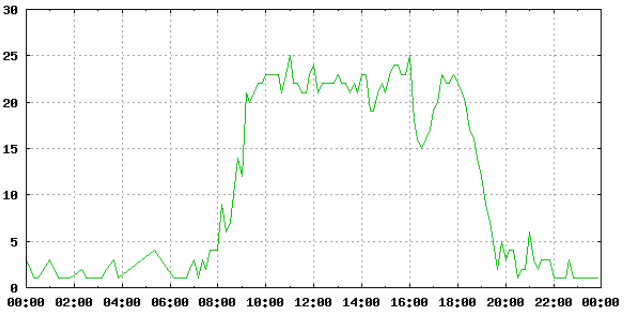


Figure 11.6: $\mathcal{A}(s_t)$

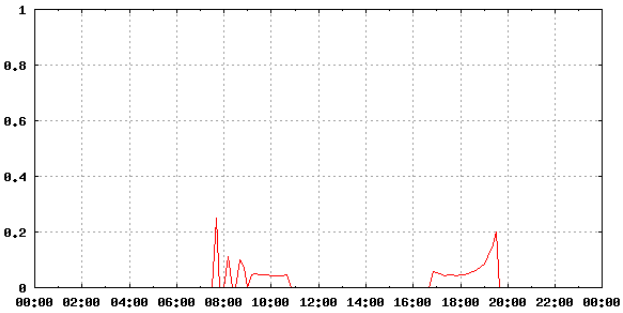


Figure 11.7: $\mathcal{P}(WINDOW\ LIGHT\ ON|s_t)$

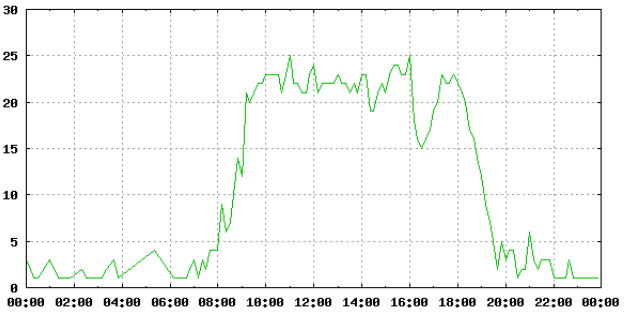


Figure 11.8: $\mathcal{A}(s_t)$

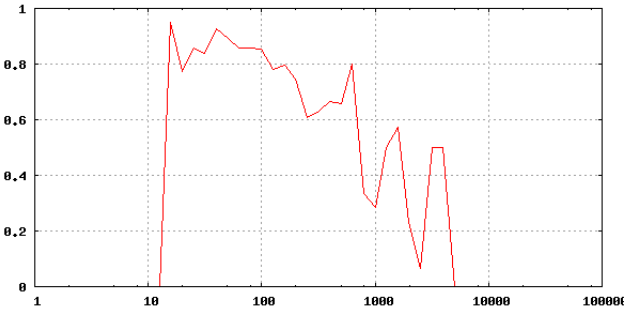


Figure 11.9: $\mathcal{P}(CORRIDOR\ LIGHT\ ON|s_d)$

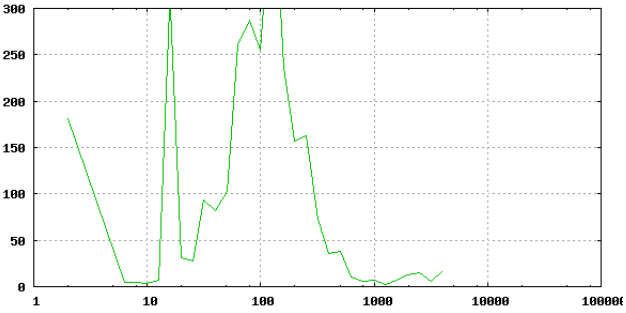


Figure 11.10: $\mathcal{A}(s_d)$

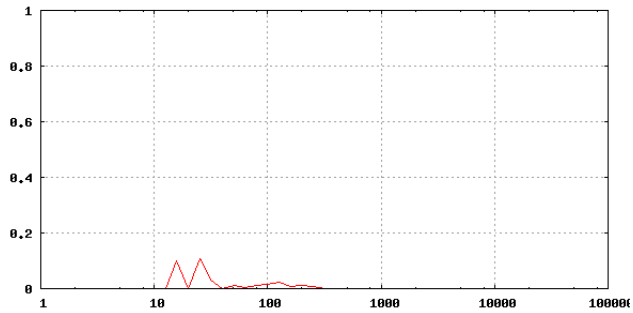


Figure 11.11: $\mathcal{P}(WINDOW\ LIGHT\ ON|s_d)$

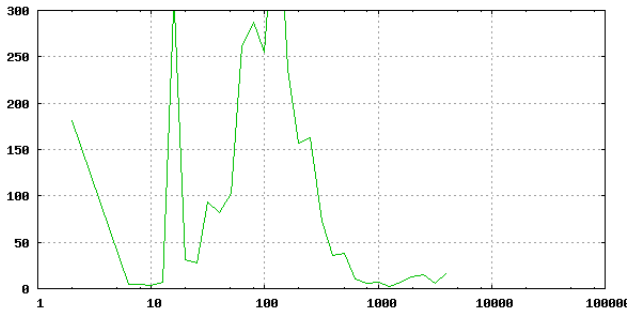


Figure 11.12: $\mathcal{A}(s_d)$

11.3.2 55.G.74

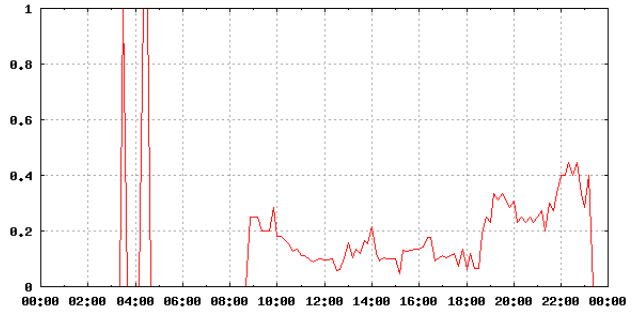


Figure 11.13: $\mathcal{P}(CORRIDOR\ LIGHT\ ON|s_t)$

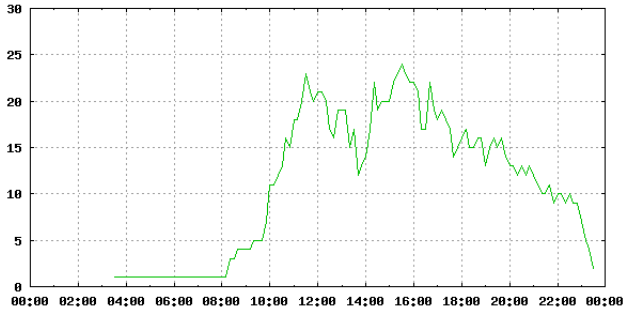


Figure 11.14: $\mathcal{A}(s_t)$ corridor

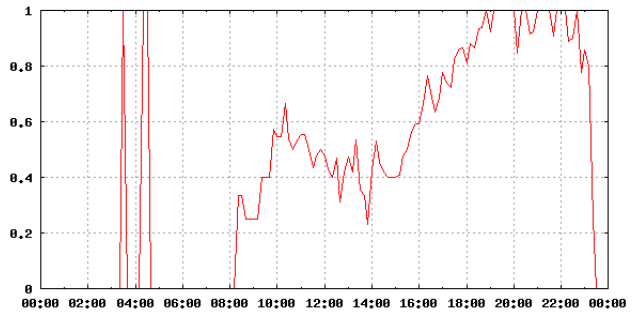


Figure 11.15: $\mathcal{P}(WINDOW\ LIGHT\ ON|s_t)$

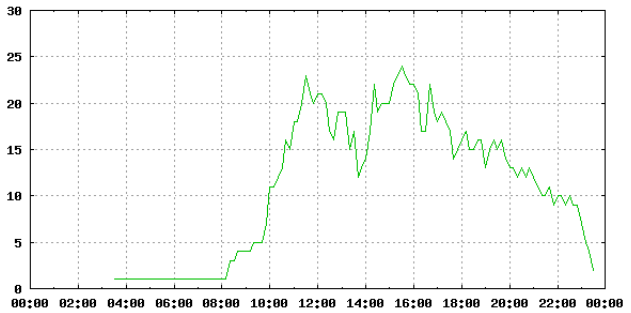


Figure 11.16: $\mathcal{A}(s_t)$ window

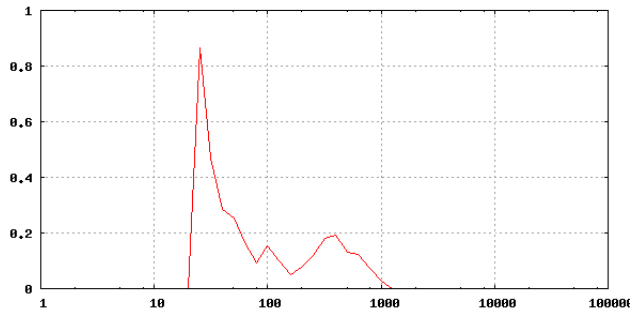


Figure 11.17: $\mathcal{P}(CORRIDOR\ LIGHT\ ON|s_d)$

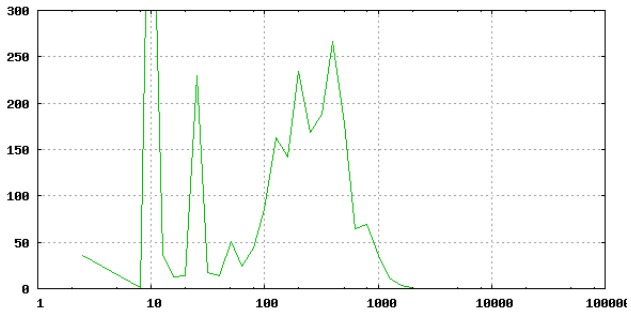


Figure 11.18: $\mathcal{A}(s_d)$ corridor

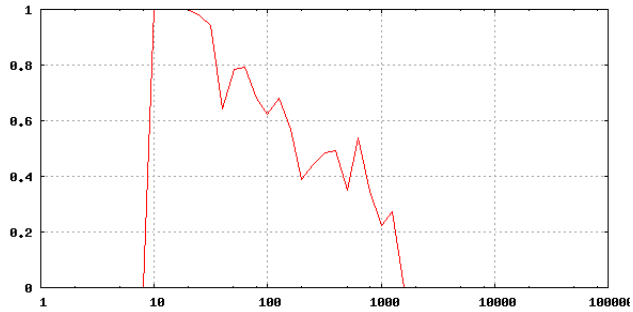


Figure 11.19: $\mathcal{P}(WINDOW\ LIGHT\ ON|s_d)$

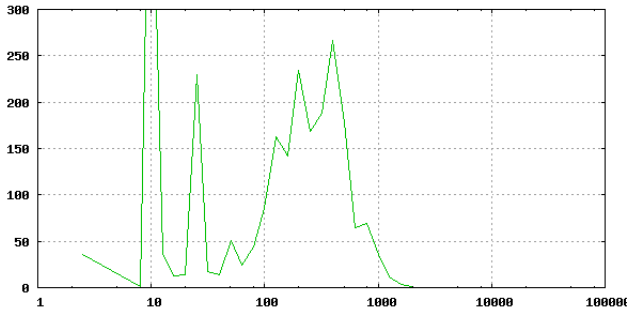


Figure 11.20: $\mathcal{A}(s_d)$ window

11.3.3 Discussion

The symbols used in the graphs:

- $\mathcal{P}(WINDOW\ LIGHT\ ON|s_t)$: The conditional probability when the light was on within a specific time-slot s_t.
- $\mathcal{A}(s_t)$ window | corridor : The number of data inputs (trust value) recorded within a specific time-slot s_t
- $\mathcal{P}(WINDOW\ LIGHT\ ON|s_d)$: The conditional probability when the light was on within a specific daylight-slot s_d.
- $\mathcal{A}(s_d)$ window | corridor: The number of data inputs (trust value) recorded within a specific daylight-slot s_d

Notice that both trust curves within each room are identical due to the fact that the presence sensor within each room remains the same. Figure: 11.5, 11.6, 11.7 and 11.7, depict the measurements which have been taken in Room 84.
As depict, we can clearly state that within the entire day the light was on. Whereas in figure: 11.6, the probability within this episodes even exhibited pretty good accuracies and thus definitely would allude that the corridor light ought to be switched on between 9 and 10 and switched on at 7.
In contrast to the corridor light, the window light astoundingly exhibited almost a reflection of latter light curve. Hence would indicate that the window light was practically on all day.

As expected, the appropriate daylight statistic also prove the latter results in providing trustable daylight values in a broad range which allude that whichever daylight we measure, the light was either on (See figure: 11.9) or on (See figure: 11.11).

Those results proved us that certain rooms exhibit a periodical set of recurring, initiated actions. In consideration of those results a possible statistical learning will definitely succeed, at least across a certain period

of time.

Unfortunately not all rooms can be seen as a periodical set of recurring actions within the daily cycle that can basically be replayed. This illustrated us room 74. In contrast to room 84, room 74 is a laboratory where many people comme and go whereas room 84 is a regular office with three occupants.
Especially the corridor light as depicted in figure: 11.13, practically doesn't give us any sufficient information if a light should be turned on or off.
Some frequency we may still be able to identify since one of the lights (window light) exhibited a continual coming and going at late nights (Compare figure: 11.15 and figure: 11.16).

On the whole though, the results actually showed us the correctness of our initial hypothesis that we stated in the beginning of this section (See: 11.3).
In summary we can freeze onto the fact that rooms are either *simple* or *complex* to learn. To address the problem that certain rooms can simply not successfully be predicted by applying just a regular statistical analysis with the provided input parameters such as the *daytime* or the *daylight* we need to think of something else to make a (better) hypothesis for controlling a regular.

It should be clear that the results gained from the performed analysis are only useful to certain degree since they were rather intended to get to know different environments.
Additionally it is quite hard to perform a correct statistical analysis since it is hard to discover which inputs have to be taken into consideration and which of them not. However the latter analysis showed us that the daylight and the daytime might gives us some powerful information about an environment. On the other hand we must clearly need to get down on paper that a statistical learning does not succeed in each environment (complex environments). This is because it is very difficult to pre-estimate which input variables are either depending on each other in form of conditional probabilities or which variables are insignificant and which of them not.
One way to circumvent the problem of conditional inputs may be the usage of *naive Bayes classifiers*.

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod P(a_i|v_j) \tag{11.2}$$

Where v_{NB} denotes the target value output by the naive Bayes classifier and v_j the prior probability within the finite instance space V and with the attribute values, each of which denoted by a_j.

For instance when establishing the *naive Bayes classifiers* into our context we would simply define the states $\{ON|OFF\}$ as the the finite instance space V. Hence the upper equation can be specified to:

$$v_{NB} = \arg\max_{v_j \in \{ON|OFF\}} P(v_j) \prod P(a_i|v_j) \tag{11.3}$$

When applying equation (11.3) we get something like this:

$$\begin{aligned} P(ON)P(daylight = bright|ON)\ P(humidity = low|ON)\ P(blindpos = 80|ON)... &= ... \\ P(ON)P(daylight = bright|OFF)\ P(humidity = low|OFF)\ P(blindpos = 80|OFF)... &= ... \end{aligned}$$

Hence depending on arg max we will get the probability for the new instance vector $\vec{x}$.
We see that this method involves a learning step in which the various $P(a_1)P(a_2)...P(a_n|v_j)$ terms are estimated, based on their frequencies over the training data. Thereby the estimates corresponds to the learned hypothesis. This hypothesis is then used to classify new instances, by applying the rule in the upper equation (11.2).

The major drawback by applying such an approach, is that it is quite hard to find a right set of conditional probabilities which do not strongly depend on others. However, this is frankly speaking quite often the case. In example when the daylight is bright then we presumably don't get humidity values above 90% which would regularly rather correspond to a fogy day.

An other problem is to find a decent method of *pre-processing* the attribute values or variables since the *naive Bayes classifiers* are classically rather meant to work with linguistic variables or classes that must in some way be pigeonholed without suffering from casualty.

In summary we conclude:

- Estimating $P(v_j)$ is simple. Compute the relative frequency of each target class in the training set.
- Estimating $P(a_1)P(a_2)...P(a_n|v_j)$ though is difficult since we typically don't have enough instances for each attribute combination in the training set. Which means that we have some sort of sparse data problem situation here.
- naive Bayes assumes that attributes are conditionally independent.

Throughout, we can say that a solution must be found to master *complex environments*, which probably will not succeed to be controlled by applying a statistical analysis on a daytime and neither on a daylight basis. Such rooms are just too agile or active to perform any such learning task on them since they just fail upon sudden changing customs.

More details about a possible statistical learning will be proceeded in the corresponding learning chapter. (See chapter: 18).

Chapter 12

Intelligent Building Framework (IBF)

12.1 Introduction

In regard to the gained results and its discussions, we can derive that must figure out another solution to approximate environment functions, for complex and maybe also for simple environments.
Apparently it is quite difficult to deal with both cases within one single algorithm or approach. Especially when considering an intelligence that should also incorporate long as well as short term memory data. Such as a one that is even capable of storing exceptional input data over a longer period of time.

By taking those summarized facts into account, we concluded that some kind of framework is useful where we have the ability to incorporate and test different AI approaches.
Additionally, by providing a set of standardized interfaces, we would facilitate any development of an urgently needed real-time environment software simulation that verifies and tests the developed AI algorithms. The software should hereby try to mimic the reality by providing a configurable simulation platform that integrates different aspects that the real ABI System should deal with.
This is necessary since any AI is heavily depending on user interactions which provide a form of feedback to the system.

In order to meet the goals stated in the beginning of this section (See: 12.1) we introduce a new algorithm that considers the changing strengths and weaknesses of different prediction algorithms.
The goal of this algorithm is similar to the already known weighted majority algorithm ([Mit97]) and hence can partly be seen as a derivative of the latter algorithm, with the crucial difference that weight alterations are strictly triggered by a function of the user input rather then their direct predictions (See figure: 12.1).

Before starting to describe any other component of the framework, we prefer to first illustrate how the framework algorithm (IBF algorithm) works in our context. The algorithm hereby undertakes the task of a **Decision Controller (DC)**, that can actually be considered to be part of the framework as well since it will emerge as one of the most important performance factors in the entire building intelligence.

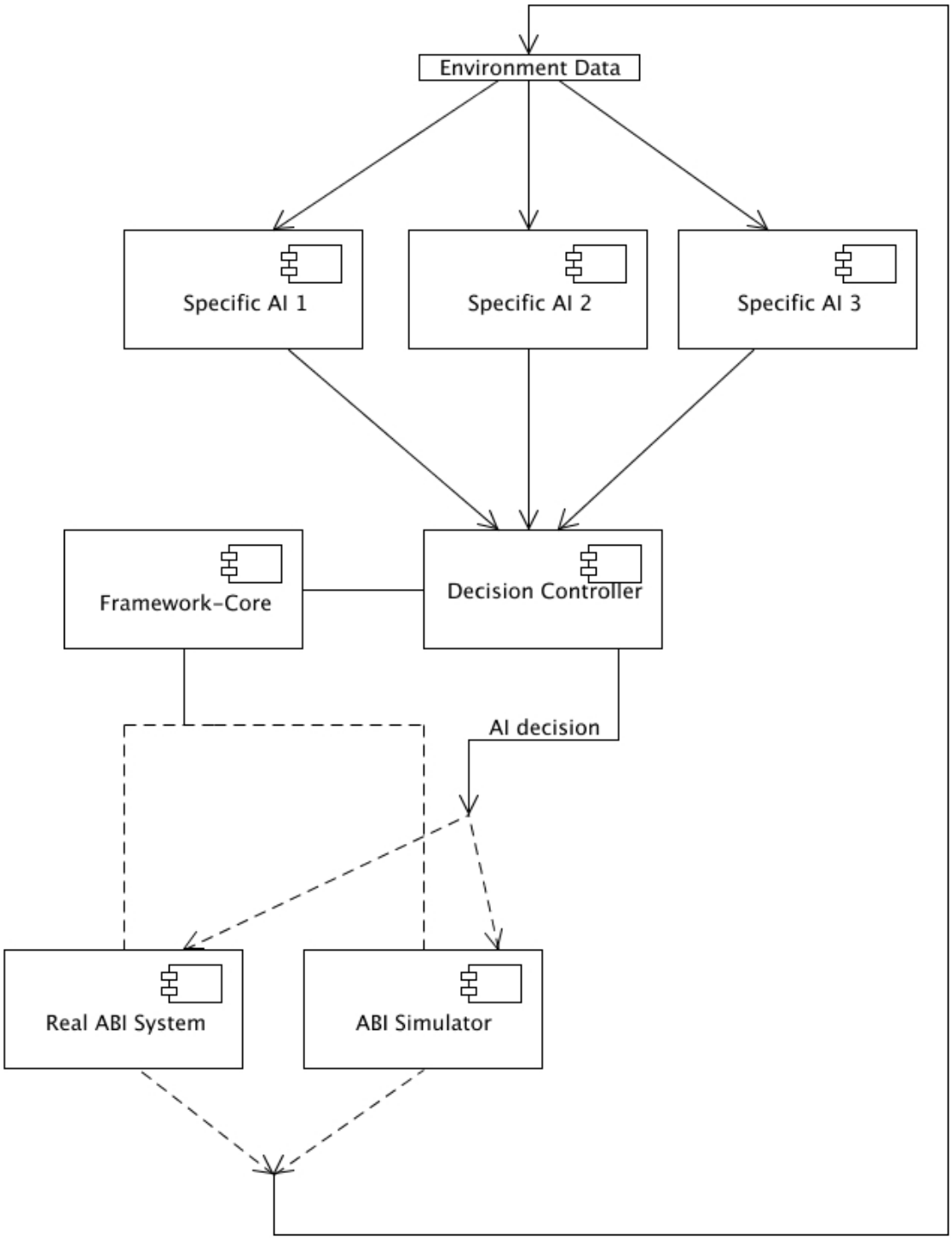


Figure 12.1: The Intelligent Building Framework (IBF))

12.2 IBF Algorithm

We propose a new algorithm that considers the changing strengths and weaknesses of different prediction algorithms. The algorithm is making a prediction by taking a weighted vote among a pool of prediction algorithms and learns by altering the weight associated with each of them. Accordingly they will start to compete among each other. Thus, algorithms which perform better will thereby be rewarded and others rather punished.
The rewards and punishments will hence be steered as a function of the user input (See figure: 12.2).

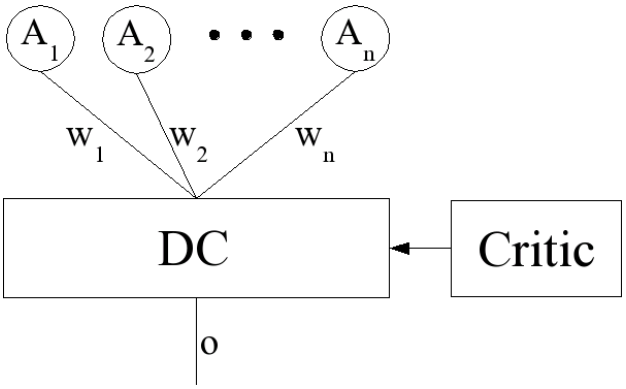


Figure 12.2: IBF Algorithm

The symbols used in this algorithm:

A_i:	Specific kind of prediction algorithm
o_i:	Prediction at time t that corresponds to the output of A_i
DC_t:	Decision Controller output at a distinct time t
w_i:	The accompanying weight w_i that is given by the decision controller DC to A_i at time t
O_t:	The resulting DC-output at time t
Δt_{user}:	The exceeded time since the last user output
α_{Max}:	Maximal learning rate
α_t:	The current learn factor

The prediction DC_t can hence be calculated as:

$$DC_t = \frac{\sum_{i=1}^{n} o_{i_t} \; w_{i_t}}{\sum_{i=1}^{n} w_{i_t}} \tag{12.1}$$

The resulting error that need to be corrected:

$$error_{i_{t+1}} = \left| O_{t+1} - o_{i_t} \right| \tag{12.2}$$

Let g be a function of the user input (Δt_{user}) whereas g [1] can be defined as:

$$g : \Delta t_{user} \longmapsto \alpha_{Max} e^{\chi \Delta t_{user}} \quad whereas \quad 0 \leq g(\Delta t_{user}) \leq \alpha_{Max} \wedge \chi < 0 \tag{12.3}$$

$$\alpha_t = g(\Delta t_{user}) \tag{12.4}$$

[1] Other functions such as a linear one, i.e. $f(x) = a - bx$ may also be possible

Ergo χ hereby corresponds to a negative value that predicates how strong user inputs need to be weighted. Hence if χ is chosen small, would result recent user inputs to be weighted stronger and therefore would cause a bigger learn factor (α) to be extracted. Vice versa, if χ is chosen big, will cause user inputs to be punished which are rather less recent then a small χ. To simplify any calculations we choose χ to be a function of the *hal-life period* t_{hl}:

$$\chi = \frac{\log 0.5}{t_{hl}} \tag{12.5}$$

Thereby t_{hl} refers to the time where $\frac{\alpha_{Max}}{2}$ is reached.

The weight will then be altered according to this formula:

$$w_{i_{t+1}} \leftarrow w_{i_t} \ \left(1 + \alpha_t - 2\ \alpha_t\ error_{i_{t+1}}\right) \tag{12.6}$$

To give a better insight how the size of t_{hl} effects the punishment or the rewarding, following graphs are given (See figure: 12.3 and figure: 12.4):

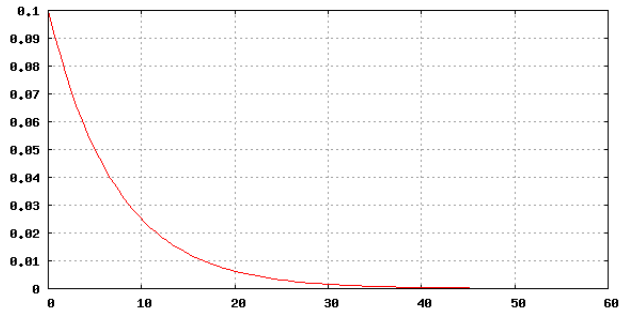


Figure 12.3: α_{Max}=0.1, t_{hl}=5 Minutes

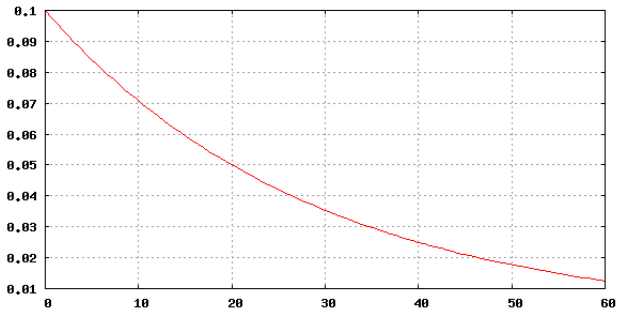


Figure 12.4: α_{Max}=0.1, t_{hl}=20 Minutes

However by applying the current solution, we've noticed that among a distinct "successful" set of algorithms, started to punish each other. As a result, the strongest algorithm continuously punished down the others, although they've been predicting almost as good.

To counteract this issue, we extend the current algorithm by introducing an additional function that basically artificially increases the $error_{i_{t+1}}$ for algorithms, that have been off base. Reciprocally the function lessens the $error_{i_{t+1}}$ produced by algorithms that exhibited predictions close to the last user input. Broadly speaking, we can consider this function as an amplifier which does either boost, or damp the weights in respect to user inputs.

The function that achieves such a behavior is nothing less then the classical sigmoid function [2].

$$error_{i_{t+1}} \leftarrow \frac{1}{1 + e^{-c\ (error_{i_{t+1}} - 0.5)}} \tag{12.7}$$

A little positive side-effect that we also recognized upon applying the *sigmoid* function is, that algorithms that were contributing a considerable amount of wrong, opposing-information to the overall decision making O_t, will herewith be reduced also. Hence the overall decision making O_t will be more agile to sudden changes.

Simplified the algorithm can be summarized to work like this (See algorithm: 2):

[2] The *sigmoid* function needs to be shifted-up appropriately since we're only interested in the positive side of it [0-1]

repeat

Fetch input vector $\vec{x_t}$

foreach A_i *in* A **do**

$A_{i_t} = getPrediction(A_i, \vec{x_t})$

end

$t \leftarrow t + 1$

foreach A_i *in* A **do**

$error_{i_t} \leftarrow \left| O_t - o_{i_{t-1}} \right|$

$error_{i_t} \leftarrow \frac{1}{(1+e^{-c\ (error_{i_t} - 0.5)}}$

$\alpha_t \leftarrow \alpha_{Max} e^{\chi \Delta t_{user}}$

$w_{i_{t+1}} \leftarrow w_{i_t}\ (1 + \alpha_t - 2\ \alpha_t\ error_{i_t})$

end

foreach A_i *in* A **do**

$w_{i_{t+1}} \leftarrow w_{i_{t+1}} / \arg\max_i w_{i_{t+1}}$

end

foreach A_i *in* A **do**

if $w_{i_{t+1}} < minWeight$ **then**

$w_{i_{t+1}} \leftarrow minWeight$

end

end

until *Stop condition reached* ;

Algorithm 2: IBF Algorithm within the DC Controller

12.3 IB Framework

According to section (See: 12.1), where we pointed out that a framework would enhance the development of different learning algorithms, we will now illustrate the major software components within our framework. To simplify any explanations, we rather prefer to discuss the class-diagram since all relations and dependencies can be easier visualized with it.

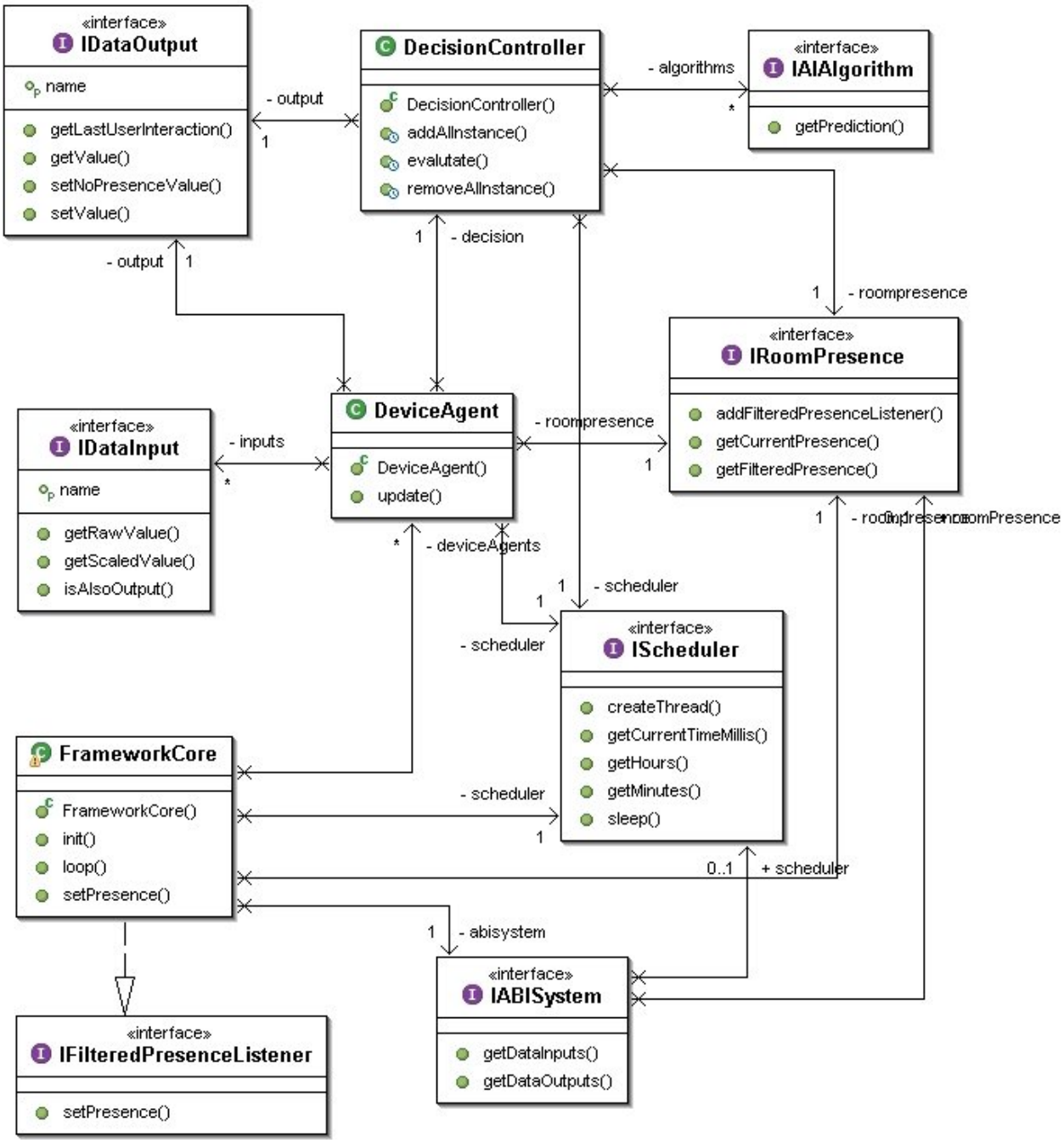


Figure 12.5: The Intelligent Building Framework (IBF))

As depicted in figure: (12.5, the heart of this framework is provided by the `FramworkCore`. In contrast to our predecessors ([RS02], [TZ03b], [TZ03a], [BG04a]), we rather follow a loop-based instead of an event-based approach to retrieve sensory inputs.
Hence, a heart-beat component is needed that triggers their pick-up. The heart-beat time should be triggered by an external concrete class that implements the `IScheduler` interface since the clock doesn't actually need to be provided by the framework but rather implemented by a concrete scheduler, i.e. a separate virtual

clock scheduler.
The loop of the `FrameworkCore` initiates the `update()` method of the `DeviceAgent`, that owns a set of concrete inputs which are basically forwarded to the `DecisionController`, that is responsible for retrieving the hypothesis, stated by each of the concrete, registered AI algorithms (See section: 12.3.5).

12.3.1 IDataInput

Each concrete input needs to implement this interface. In example when hooking-on the real ABISystem, a concrete class must for instance realize input pre-processing, i.e. scaling.

`getRawValue()`: This method retrieves the raw value as it has been provided by its dedicated sensor. According to the RBCAPI ([NB05b]), each input may underly a set of predefined types. Such as an `Enum` e.g, on or off.

`getScaledValue()`: By implementing this method each input must be brought into a double representation that fits a value between 1 and 0.

`isAlsoOutput()`: Some effectors may also appear as sensory inputs. In example: The blinds can indeed be moved, but at the same time also provide sensory inputs e.g. Current blind position or rotation.

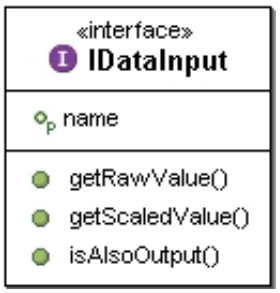


Figure 12.6: IDataInput interface

12.3.2 IDataOutput

Each concrete input need to implement this interface. In example when hooking-on the simulation ABISystem, a concrete class must take over the task of providing the set of methods provided by this interface 12.7.

`getLastUserInteraction()`: This method should retrieve the time (See section: 12.3) when the last user input has been received.

`getValue()`: Each output must keep track of its actuator status (Currently only one).

`setNoPresenceValue()`: In certain situations we might be glad of having a way to alter the current state. In the current usage we use this method to switch off the lights when nobody is present within a pre-specified time. This is necessary since we try to keep the energy consumption at a low level (See section: 2.1).

`setValue()`: The real as well as the simulation ABISystem, need to implement a way to physically or virtually alter an effector state change. For example, switching off the window light or moving the blinds to a certain position.

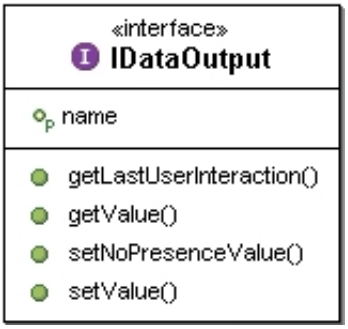


Figure 12.7: IDataOutput interface

12.3.3 IABISystem

As mentioned in the beginning of this section two local ABI System are to be implemented.

***Simulation ABISystem*:** In order to verify and test the developed AI algorithms, a real-time simulation software needs to be implemented that tries to mimic or replicate the environment the current ABI System should deal with. This was due to the fact that any AI is heavily depending on the user's interactions that are gradually taken and provide a form of feedback that can be considered as a measurable user satisfaction.

Real ABISystem: In contrast to the simulation, the real ABISystem solution basically is physically controlling the dedicated environment, respectively its effectors. Hereby each effector (momentary lights) is provided a dedicated device agent. Hereby we strike a familiar path that has also been implemented by our predecessors ([RS02], [TZ03b]). In order to communicate to the real ABI System the RBCAPI [NB05b] has been used.

By the way this interface can actually be considered as an *abstract factory* ([GHJV94]) since each concrete factory is responsible for creating its own compositions (See figure: 13.1).

Since each of the systems must provide a way to retrieve all currently supported in- and outputs, the following interface can be derived (See figure: 12.8):

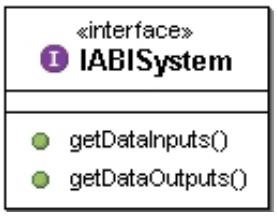


Figure 12.8: IABISystem interface

12.3.4 IScheduler

We already stated that the term *time* used in our context may not correspond to the system time. The reason is to allow a virtual time to be incorporated, that allows the time to be accelerated in contrast to the time being. Hence we need different ways of:

1. creating threads (`createThread()`)
2. get the current exceeded time (`getCurrentTimeMillis()`)

3. retrieving the currently exceeded hours (`getHours()`),
4. and minutes (`getMinutes()`)
5. and an own way to sleep since the regular sleep provided by Java ([JAV]) is impractical to use within a real-time simulation. (`sleep()`)

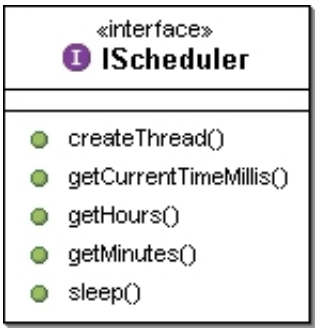


Figure 12.9: IScheduler interface

12.3.5 IAIAlgorithm

As introduced in the beginning of this section and partly also in the IBF section (See: 12.2), the whole idea of this framework is to figure out which algorithms (each of which must hereby implement the `IAIAlgorithm` interface (See figure: 12.10)), is the best suited among all algorithms. In example according to the conducted environmental analysis (See section: 11.3) we proved that certain environments have recurring behaviors. Unfortunately not all environments can be predicted that way. Hence we belief that each of the algorithm might feature a distinct particular strength, that will eventually be revealed within a period of time. In complex environments, it might even comes down to a multiple switching among certain algorithms.

In order to integrate each algorithm, each one must implement a solution to retrieve a prediction (`getPrediction()`). This methods will be frequently called by its corresponding `DeviceAgent`, or rather its dedicated `DecisionController` (See figure: 12.10).

Figure 12.10: IAIAlgorithm interface

12.3.6 IRoomPresence

In the same way as the environmental analysis has been accomplished, we only record environmental data and thus perform learning tasks, only when someone is present in that specified environment. With this condition we differentiate ourselves from our predecessors ([RS02], [TZ03b]), which were applying an anytime learning in the sense that they had been collecting data regardless of a room's presence status. We belief that it doesn't make sense to learn, when having prior knowledge on a room's presence status.

Accordingly we need to provide a way to retrieve the presence status since it isn't actually involved in any learning process directly, but rather a condition for any learning to take place. In reference to a real-time simulation environment we derived an interface, that needs to be implemented by both local ABI

Systems.
Hereby `getFilteredPresence()` is suppose to return a filtered presence status since the common presence sensors within our building, here at the Institute of Neuroinformatics (INI) more or less, semi-minutely announce their current presence status which was somewhat impractical for us to work with (Also compare: [BG04a]). The delay caused by the filtering has currently been set to 10 minutes since it was found to be adequate for most of the occupants.

By incorporating such a delay other problems are brought along. Problems that need to be circumvented by a rather unconventional way. One major problem is, that a light can manually be switched off by occupants before they actually leave their room. Therefore a second delay that we call *validation time* was needed. However the flip side to this coin is that we must enable events to pass trough without sustaining a delay. We face this situation when occupants enter a room and as a result, would be capable to respond with a probable lighting on. Hence, some boosting is required that temporary allows a device agent to forward such events to make an instant decision. See figure: 12.11 and figure: 12.12 for its proposed interfaces.

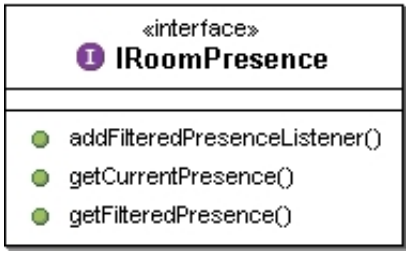


Figure 12.11: IRoomPresence

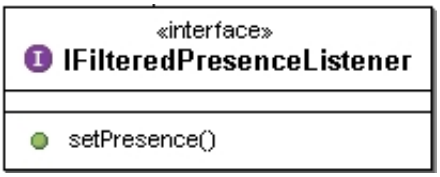


Figure 12.12: IFilteredPresenceListener

Part V

ABI Simulator

Chapter 13

Introduction

Part: IV already introduced the urgent need for a real-time simulation (RTS) environment platform where we have the ability to verify and test the developed AI algorithms, especially in their correct implementation.

13.1 Requirements

The simulation is one of the central components within this thesis since it allows any algorithm to be tested and judged to a certain point of degree.

In chapter 10, we already stated that just storing all sensory as well as user and IB inputs are not sufficient enough since the data evidentially can't simply be replayed. Hence some other solution must be found that should mimic or replicate a real environment that an IB should deal with. One of our predecessor ([TZ03b]) also mentioned that a real-time simulation platform would be necessary because any working AI should have been investigated and tested with different varying conditions.
In the introduction part (See: I), we already mentioned that for instance the daylight is a dynamic source of lighting where different skylight levels can be found even under the same sunlight condition. Furthermore one can identify that the usage of electrical lighting is mostly depending on the time of the year.
Considering those facts we find that the need of a simulation is compelling where we have the ability to simulate entire years, seasons and thereby even take instant weather changes into account.

In order to comply to the task of providing a convenient and configurable real-time simulation environment, that finally should try to mimic user inputs close to reality, we can extract following goals:

- Real test cases must be able to be created, configured and conducted.
- Test cases must be storable and loaded upon request again.
- The test cases must be conducted on a daily basis.
- Each day must therefore be able to be configured (Temperature, Humidity, Daylight as well as User preferences).
- Herewith, the user must be able to be swapped within a test. In example every month.
- The user must be configurable (Needs, preferences) i.e Daylight condition, Blind condition, blind or light lazy.
- The results should be able to be evaluated using graphs.
- The test data are to be brought into a textual representation as well.

- A test (with the same test data) should able to conducted several times, to avoid or at least lessen chance.
- The speed of the test should be maximized up to the pure consumption of each AI algorithm.

13.2 Overview

Although the simulator is one of the most interesting parts within this entire thesis, especially from a computer science point of view, we cannot dig into its detailed implementation constraints, in order to make room for other major subjects such as machine learning. Nevertheless it is crucial to have some insights on how the whole framework and within its real-time simulation platform is composed and designed. In the previous chapter we mentioned that the `interface, IABISystem` basically represents an abstract factory that creates the simulator as well as the real, local ABI System.

To make a long story short we give the classical example on how the simulation as well as the real ABI System needs to be created (See 13.1).

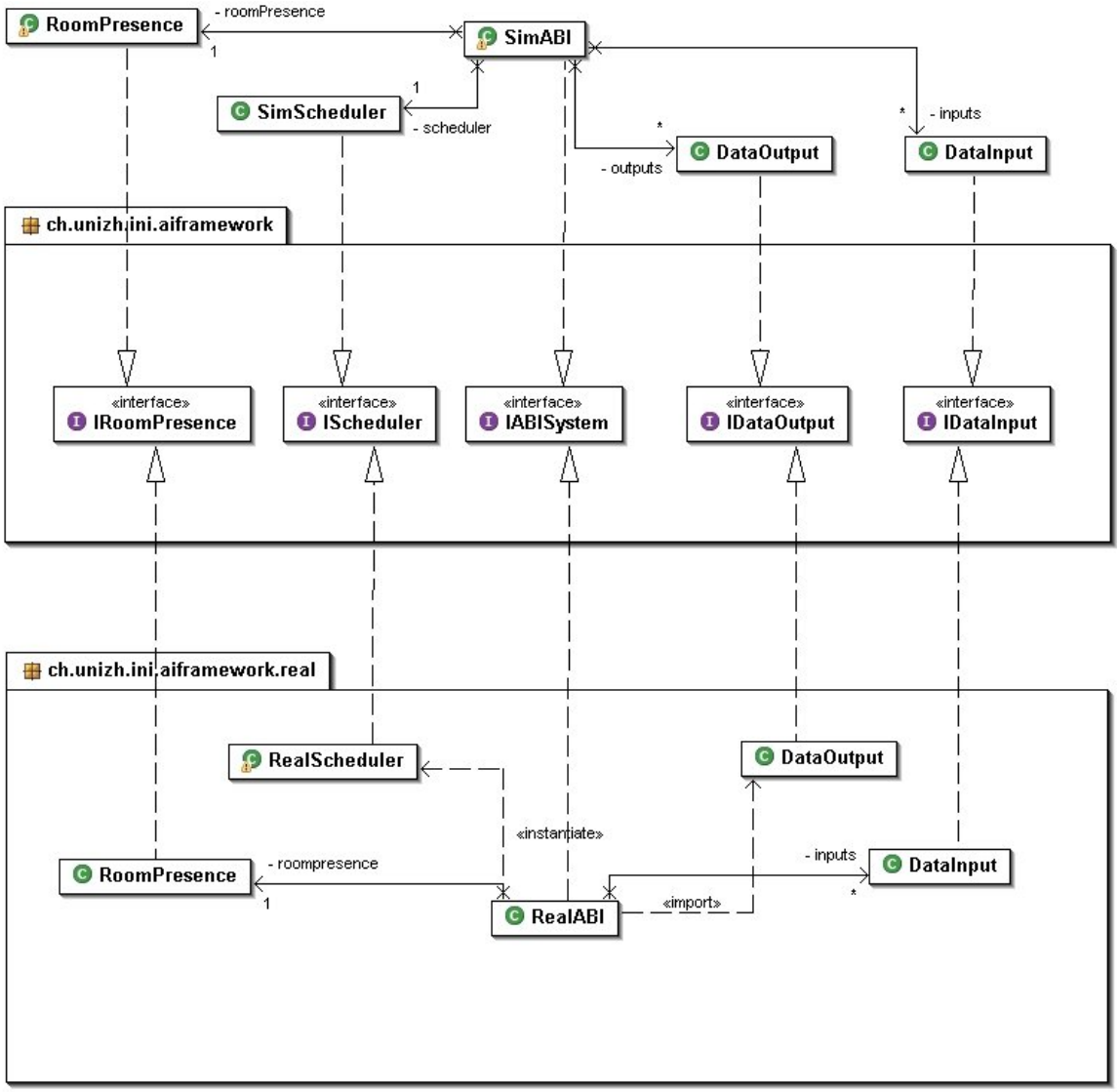


Figure 13.1: Abstract Factory

Hereby the classes `RealABI` and `SimABI` are undertaking the task of a concrete factory. Please note that this design has been simplified to point out some of the essential design decisions only.

13.3 Scheduler

One of the most obvious differences between the real and the simulation ABI is the so called scheduler. The real ABI system doesn't actually do anything spectacular. All it does is basically delegating its functionality to system related Java ([JAV]) classes (e.g. `System.currentTimeMillis()`, `Thread.sleep()`), whereas the simulation scheduler involves the implementation of an own scheduling system, in respect to the latter multi threaded "task distribution".
In contrast to classical scheduling algorithms such as preemptive, priority based round robin scheduling, we don't actually offer such a general task model. We rather stick to only one simple task model that is enough sufficient for our purpose.
The scheduling discipline that we applied is a non-preemptive scheduling without considering any priority.
In reference to the requirements stated in the section 13.1, one of the major goals of this RTS is to maximize its speed by all means possible. In general we can motivate that the actual processing time should only be consumed by the algorithms whereas the time being is accelerated to infin

Chapter 14

Simulator Usage

Instead of providing a detailed design and implementation documentation we would rather found the usage of the RTS more important since the performance of each learning algorithm will later be tested and illustrated herewith. Especially its derived graphs profoundly illustrate how they have been reacting upon different inputs. Hence it is crucial to understand how to utilize the simulator. Above all, it is fundamental to know how different configurations influence the behavior of the user's multi sensory environment (exterior daylight, temperature and humidity).

14.1 Startup

When starting the application one will see the little startup screen (See figure: 14.1) that currently offers two options to choose from.

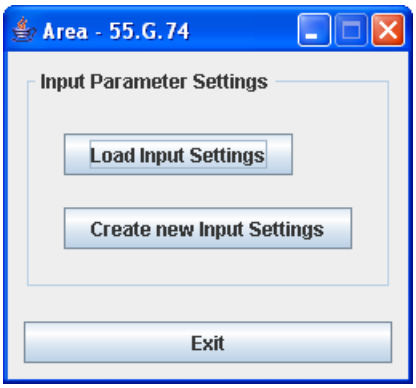


Figure 14.1: Startup dialog

We first explain how tests are to be configured and launched and at a later stage we demonstrate how tests are loaded into the simulator and how the gained results can be illustrated from raw data again.

14.2 Create new Input Settings

By selecting this option the user will be prompted to prescribe the length of the test period.
Commonly the test will be held between 30 and 120 days and hence have already been pre-configured to be loaded up (See section: 14.4).
When the number of days have been specified, the main settings dialog will show up (See figure: 14.2).

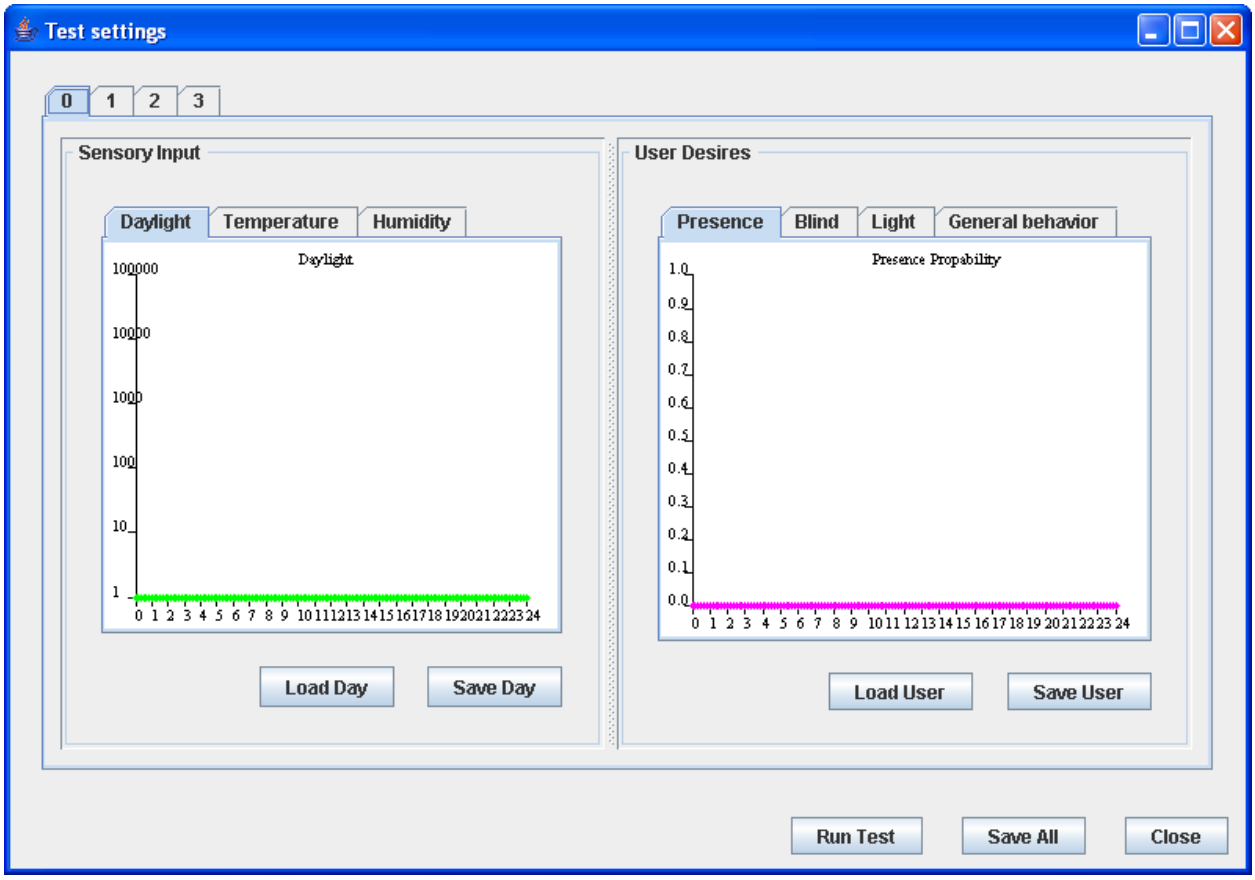


Figure 14.2: Main Screen

Thereby the entire main screen has been split into two setting categories. On the left side sensory settings are conducted, whereas on the right side user preferences are to be set.
Each configuration takes place on a daily basis and is represented by a tab pane. Hereby each day must be configured separately, in order to enable different sensory inputs to be configured and also to setup different user profiles that facilitate user preference switching among tests. Herewith we're capable to emulate long as well as short term behavior tests.

14.2.1 Sensory input settings

Generally tests are being configured by starting to setup the daily curves which are momentary composed of three different sensory inputs.
The exterior daylight (See figure: 14.3), the temperature (See figure: 14.4) and the humidity (See figure: 14.3).
Each of the graphs can be customized to suit any real environmental lighting. In example a fogy coolish day can be emulated by providing the necessary plots such as high humidity or low temperature. When considering an entire test period, we might even take instance weather changes into account but also seasons, etc. Herewith we potentiate any learning algorithm to gather and keep such a knowledge to improve the prediction making by incorporating a long and short term memory that try to rehearse or generalize any

such kind of recurring patterns.

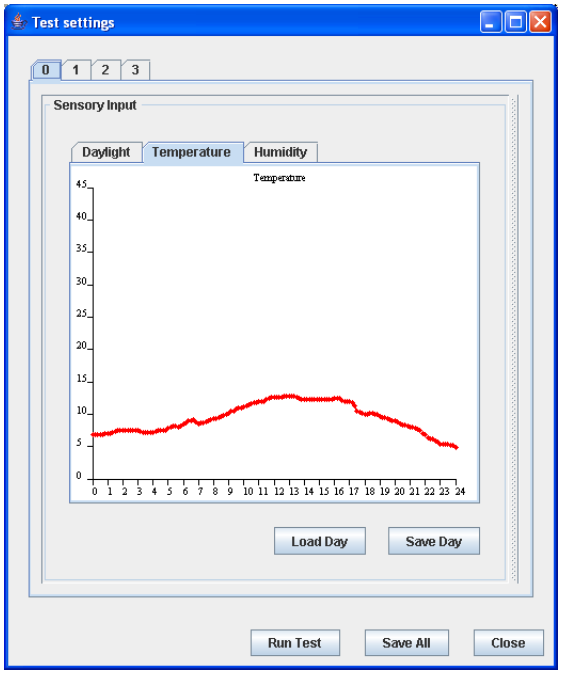


Figure 14.3: Exterior temperature settings

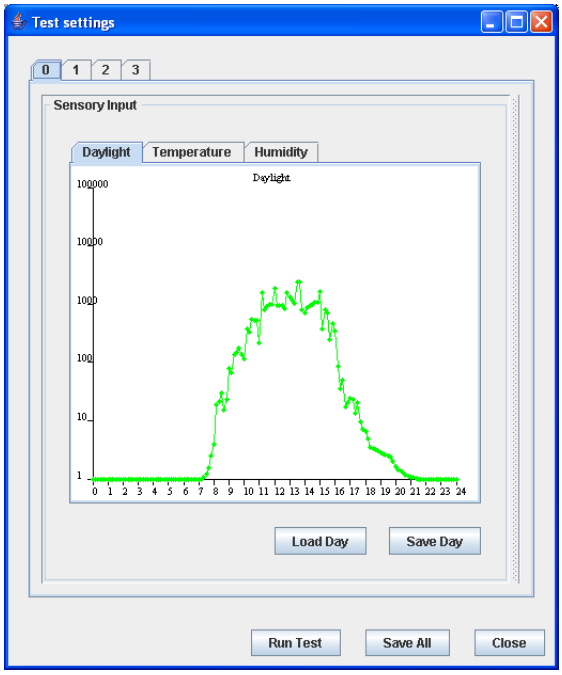


Figure 14.4: Exterior daylight settings

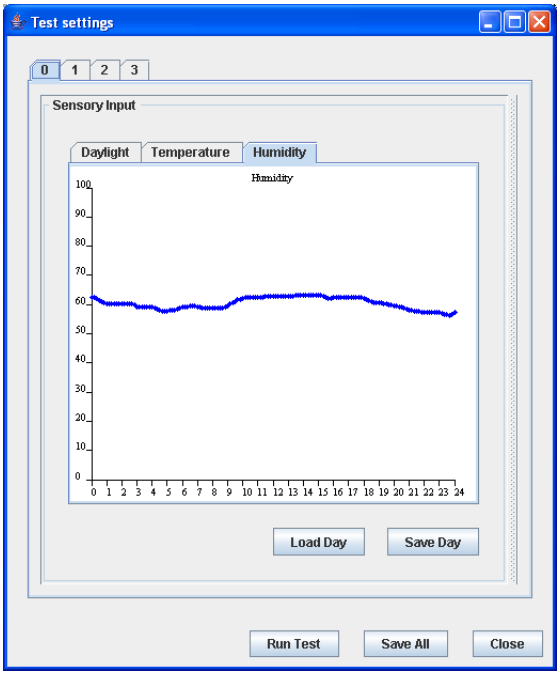


Figure 14.5: Humidity settings

14.2.2 User preferences settings

The configuration of user preferences is probably one of the most substantial settings which need to be configured and will definitely have a huge impact on the agility or the laziness of all user inputs. Upon changing such customs, any building intelligence must eventually react to such changes by incorporating the new preferences that we also refer to as *knowledge*.
Therefore care must be taken when configuring the synthetic users since each of them should try to mimic real users and not to impossible IB controllers to conduct wise decisions.

Again, we would like to point out that every day must be configured with an individual user behavior. Hence, different user behaviors within the entire test period are possible and even quite important to consider in a test-row, especially when modeling a *Complex Environment* such as a laboratory. Of course it doesn't make sense to provide huge changes within user profiles solely because as for most users, they frankly hardly have strongly distinguishable preferences, except their presence status. Accordingly, any tester should think about that when setting up such a test row.
In order to bring in some randomness within a commonly fictional user's presence status, we provide presence probabilities instead of straight values that sharply indicate a user's presence.

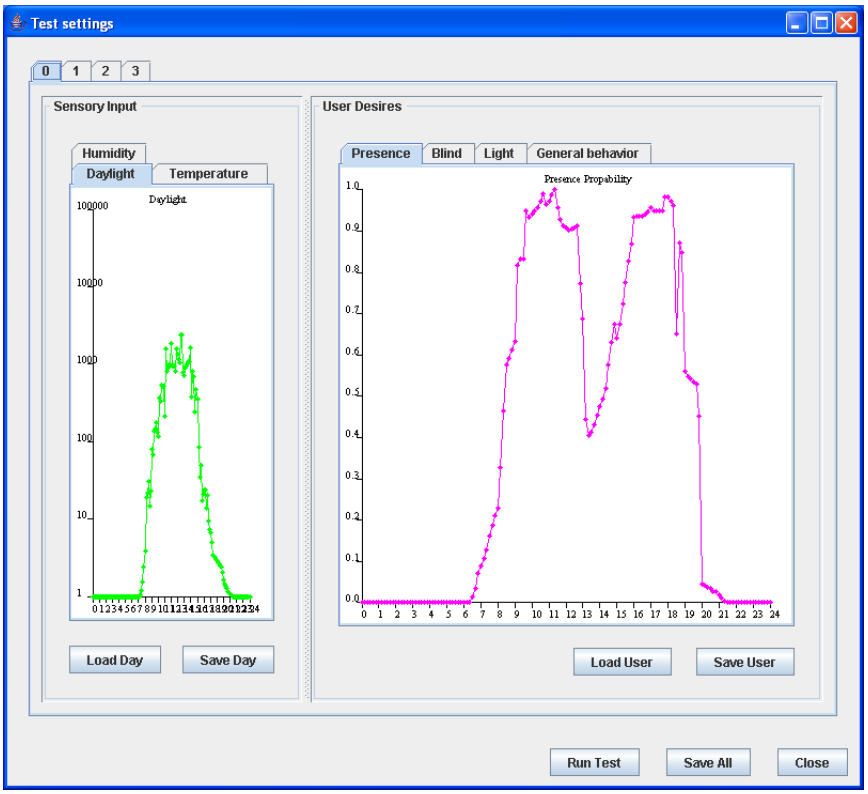


Figure 14.6: Presence status probability settings

The variability of the presence status can hence be considered and derived with the following simple algorithm: 3.

Result: Variable presence status $P_{presence_t}$ with respect to the proposed presence $Prob_{presence_t}$
input: Presence Probability $Prob_{presence_t}$

$\delta_t = 0.5 \cdot \delta_{t-1} + randomDouble() \cdot SMOOTH$

$\vartheta_t = \vartheta_{t-1} + \delta_t$

if $\vartheta_t < 0$ **then**
    $\vartheta_t \leftarrow \vartheta_t + 1$
end
if $\vartheta_t \geq 1$ **then**
    $\vartheta_t \leftarrow \vartheta_t - 1$
end

$P_{presence_t} \leftarrow Prob_{presence_t} \geq \vartheta_t$

Algorithm 3: Variable presence status

The symbols used in this algorithm:

δ_t:	Corresponds to the smoothed delta of the presence status
ϑ_t:	The calculated dynamic presence status with respect to the proposed presence probability, $Prob_{presence_t}$
$SMOOTH$:	A constant that indicates how strong past presence probabilities $Prob_{presence_{t-n}}$ are to be considered. It has been proven that values in the range of 0.001 were quite adequate when considering an AI framework frequency of $600^{-1}Hz$.
$P_{presence_t}$:	The final dynamic presence status $P_{presence_t} \in \{0, 1\}$
$Prob_{presence_t}$:	The proposed presence status given as a probability
$randomDouble()$:	Pseudo random value between 0 and 1

The blind preferences is presumably one of the most crucial factors to look out for since it has a huge impact on each user behavior since the interior daylight has been calculated considering:

a) the blind position

b) the blind rotation

c) the blind laziness

d) and the exterior daylight on a day d

It is very important to know how these factors have been used to continuously calculate the interior daylight (See: 14.1).
Furthermore as depicted in figure: 14.8, the calculation is even depending on whether the given fictional user rather has a favor for lights or blinds. We refer this specific behavior as *blind or light laziness* and accordingly consider them within our calculations.
As depicted in figure: 14.7, each row should correspond to a configuration, that the current user should feel comfortable with.
Hereby, the first component refers to the current blind position and the second and the third complies to the exterior daylight spectrum. Hence, each input will eventually need to be categorized to one such row. If the current configuration isn't suitable anymore we might rather tend to move the blinds to a better suited position. In order to lessen any blind movements we introduced a variable that keeps track of a *user satisfaction* level, that is continuously calculated as a function of the *blind laziness* and the error azimuth. Specifically speaking this means that every time when a configuration has been hit that wasn't practical at that current moment, the *user satisfaction level* will subsequently be decreased by a distinct size which has been calculated through the severeness of the change and whether the user tends to be blind lazy or not.

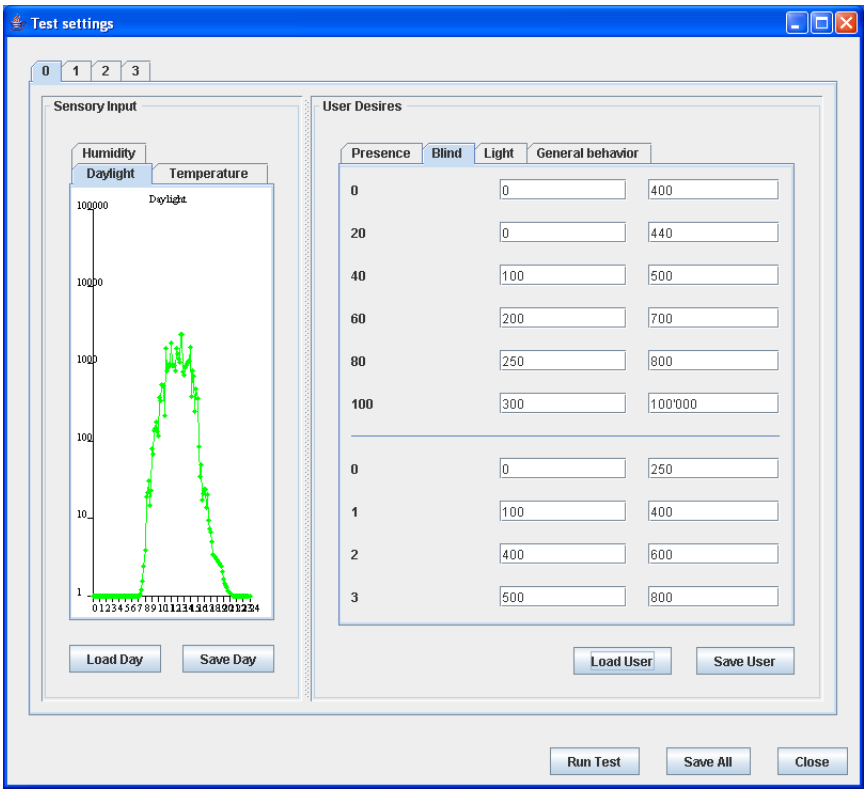


Figure 14.7: Blind preferences settings

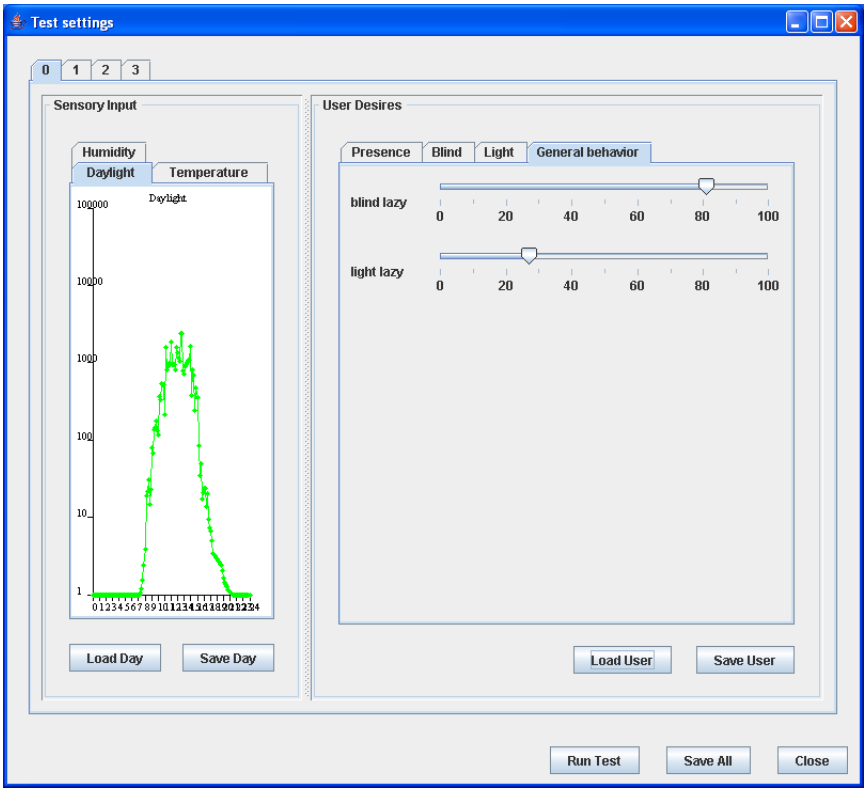


Figure 14.8: General behavior settings

If the *user satisfaction level* falls below zero, we take a opposing action that will fulfill the current user desire. On the other hand we increase the *user satisfaction level* upon hitting the current configuration being, by a minuscule reward like amount.
We omit on illustrating this algorithm in a formal way since its discussion part should have been enough sufficient.

The blind rotation are similarly treated and hence should be clear also.

Since the interior daylight is depending the exterior daylight that is deflected upon the blind position and rotation we simplified the calculations by applying this formula (See: 14.1).

$$Lux_{interior_t} = Lux_{exterior_t} \cdot \frac{1}{n} \cdot \sum_{i=1}^{n} \left(1 - blindposition_i + \frac{6 - blindrotatation_i}{8} \cdot blindposition_i\right) \quad (14.1)$$

Whereas n denotes the number of blinds within a room and $Lux_{exterior_t}$ corresponds to the configured value at time t (See section: 14.2.1).

The following picture should provide the necessary reasons why such a simplification is adequate in our context. For instance when the blinds are down, only the blind rotation can further influence the interior daylight (See figure: 14.9).

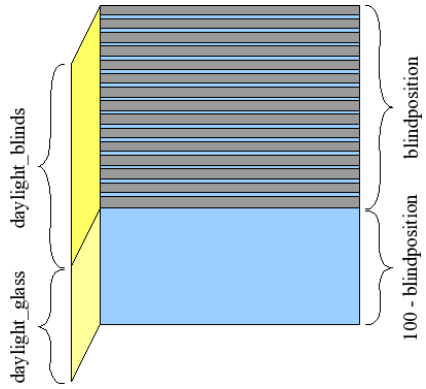


Figure 14.9: Interior daylight calculation illustration

Having once calculated the interior daylight, we can start to setup a specific preference that signalizes how sensitive the user should react, in respect to different interior daylight values (See figure: 14.11). A fictional user perception is model Additional to the latter settings though, are the meaning of *lower and upper bound* (For an illustration see figure: 14.10).

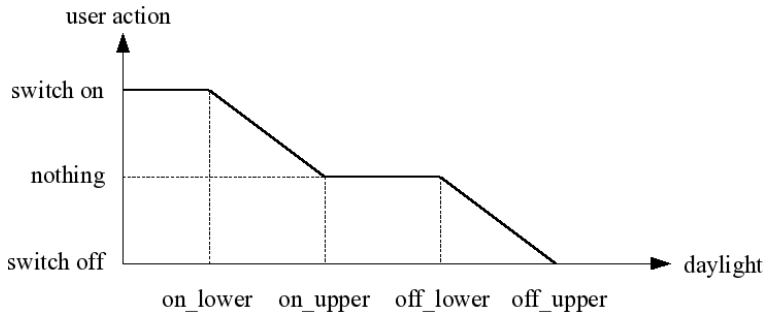


Figure 14.10: Upper and Lower bounds for the interior daylight configuration

If the current interior daylight falls below the lower bound value of the *switch on daylight*, the light status will be changed without even affecting or considering the *user satisfaction level*, introduced previously. Likewise

we deal with the upper bound value of the *switch off daylight*.
If the interior daylight falls between the the upper bound border of the *switch on daylight* and below the lower bound border of *switch off daylight*, no specific action is taken since the all settings seemed to be adequate for the user. Similar to the blind *user satisfaction level*, we only undertake an opposing action upon reaching a maximal *user dissatisfaction level*.

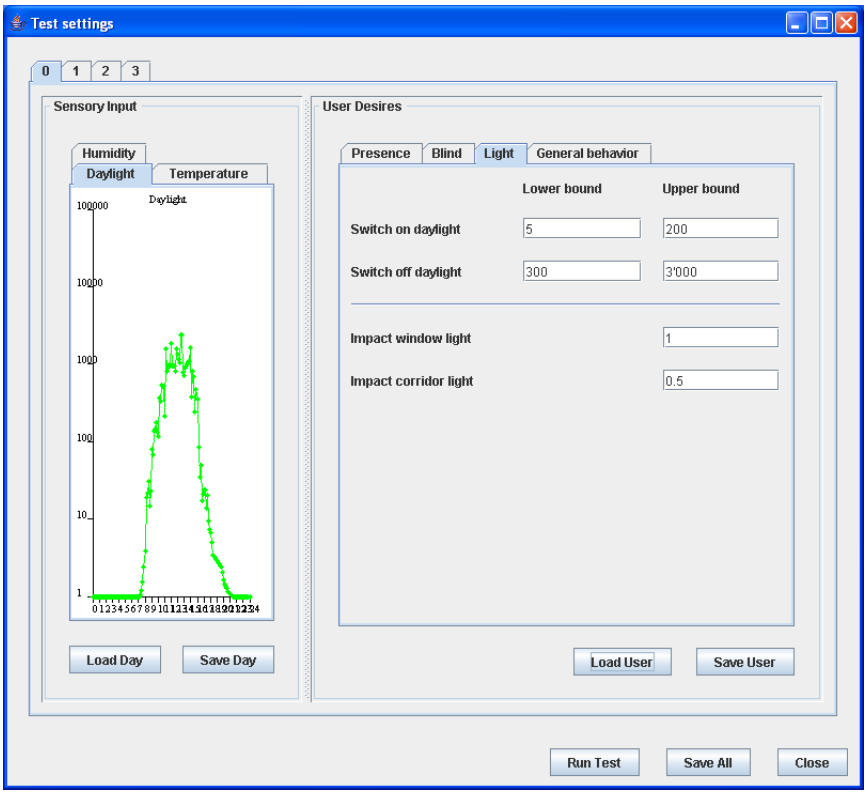


Figure 14.11: Light-preferences settings

14.3 Saving a test

We suggest to store each configured test, day or user within the application subdirectory called *data/** where the wild card stands for either days, tests or users.

The file that is being stored isn't textually readable since object serialization has been used to load and store the data.

It might be very inconvenient to configure each day row by row. Hence, to improve the usability for setting up a test, we allow to copy and past days among different tests (See figure: 14.12). By focusing the source window and pressing **Strg+C** the current day and within its entire configurations (sensory as well as user configurations) are being copied. The copy will be carried out by popping-up the target window and simply pressing the well known **Strg+V** for pasting (See figure: 14.12).

14.4 Loading a test

By following the suggestions of the latter section (storing tests), we can anytime carry out a previously conducted test by loading it back into the simulator (See figure: 14.1). This is accomplished by pressing the button called *Load Input Settings* and selecting a desired test ending with **.simtest*.

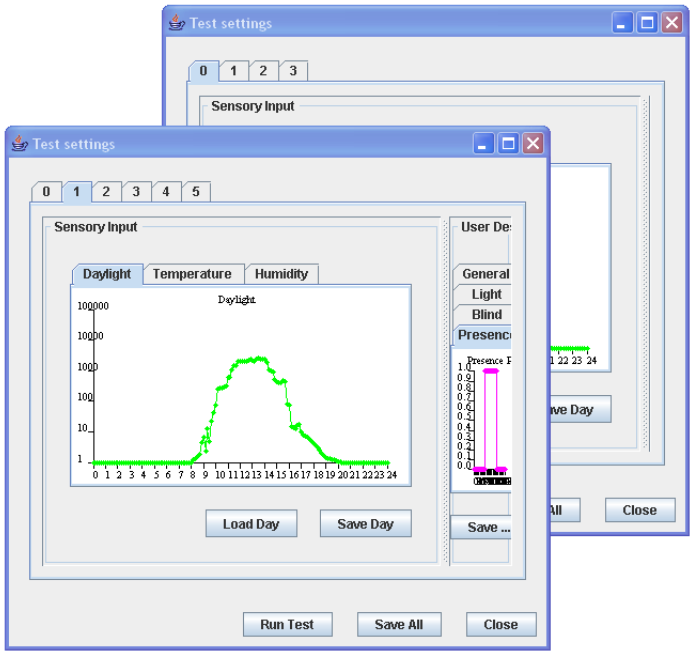


Figure 14.12: Copy and past days

14.5 Launching a test

When all configurations have been carried out, the test can be performed by invoking the run test button that is located at the bottom of the main window (See figure: 14.2 and figure: 14.13).

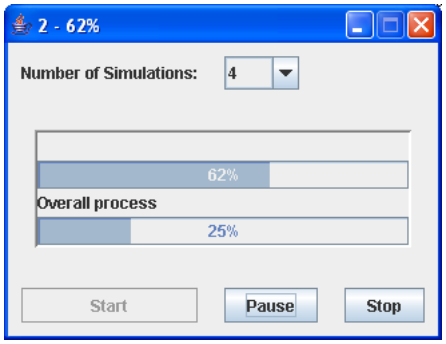


Figure 14.13: Test progress

The user might gradually need to make some modification on the test-configuration settings in order to attain a decent replica of a specific environment.
In a later stage, real sensory data are to be integrated to facilitate and improve the sensory data representation. Nevertheless the gained results are in some extent quite representative since user behaviors can quite good be simulated due to incorporating a *user satisfaction level* that is frequently calculated by considering different parameters of user *stimuli*.

Before a test can be executed, the user will be prompted to provide a file that allows all the test data to be stored in. Care must be taken upon running huge simulations though since sufficient hard disc space is beneficial. Incidentally, the resulting files might easily exceed sizes far beyond a 100 MByte, i.e. 100 MByte/per test lap is quite normal.

14.6 Test evaluation

While performing a test, a progress bar is provided that gives helpful feedback (See figure: 14.13). Particularly upon performing multiple tests (e.g, 10 in one row with the same data).
Although it is a real-time simulation, it might still take time to conduct all algorithms. Hence it is quite possible that the test might endure several hours (e.g by conducting a one year simulation and this several times).
When all test have been performed, the results can be analyzed and evaluated by an internal graph viewer (See figure: 14.14).
It is a quite convenient feature to have the ability to correlate each of the graphs among each other. In particular, we're interested in how the algorithms have been performing and also to point out their strengths and weaknesses. In reality as well as in simulation, *unforced user inputs* might occur. Sometimes it is quite helpful to get to the bottom of why for instance suspicious user input stairs have happened within the simulation (More on that in the learning part (VI)).
It's very important to understand how to read the graphs. In order to avoid any misleading conflicts the major input plots are illustrated.

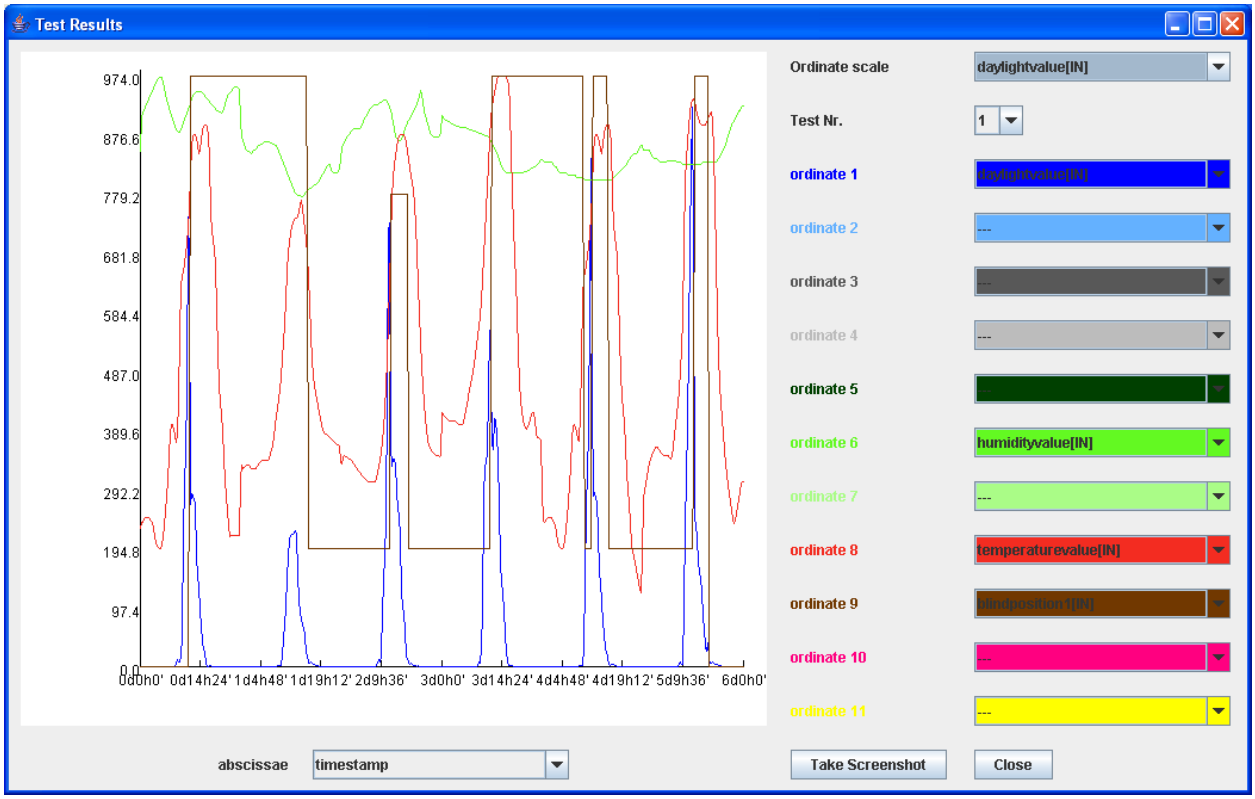


Figure 14.14: The blue graph shows the interior daylight, the red the temperature, the green the humidity and the gray graph corresponds to the blind position. The ordinate is currently set to fit the exterior daylight ordinate.

One important measure for any device agent is the ratio of all unforced decisions taken by the system divided by all feedback signals (corrections) received by the building intelligence controller ([RJD04]).
To illustrate this ratio consider the following figure: 14.15.

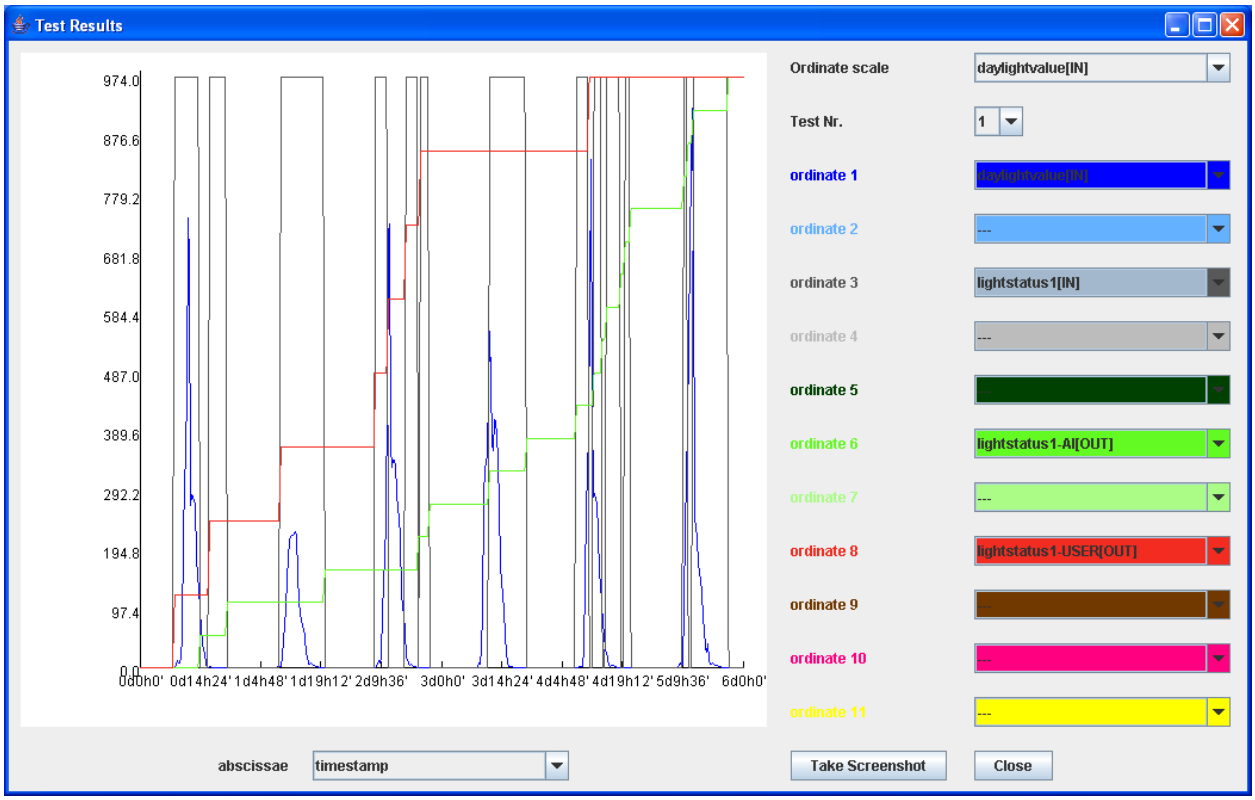


Figure 14.15: The red curve hereby depicts the accumulated number of forced user inputs whereas the blue curve illustrates the accumulated AI inputs. Commonly, most algorithms won't state a prediction in the first couple of days and hence rather exhibit a creepy behavior.

By either scrolling or by right clicking and dragging the mouse vertically upward the graph, a more accurate view is given to evaluate the data. With growing size of the test and by zooming out far, some performance optimizations must have been taken to speed up the evaluation process. Inevitably close data points have been merged to fit one data point and thus might leak on information, especially when trying to provide an over-viewing screenshot that can be captured upon clicking the corresponding button at the bottom (See figure: 14.14 and figure: 14.16). Sometimes we're interested in finding out different correlations among the plots. Hence the ordinate as well as the abscissae can be set to the appropriate measure anytime.

Quite often one might notice that some of the inputs were actually caused by so called *unforced* user inputs. In general we can distinguish between two different major cases of *unforced* user inputs where possible punishments actually would't be appropriate.

1. Every input (whether or not it was an user or an IB input) will cause a "security" delay to be initiated that basically should actually prevent a device agent from taking any subsequent action, i.e upon any user input, no opposing action should be carried out in the first place within this delay.
   The consequences that need to be paid upon incorporating a security delay (that was found to be one of the major issues, according to the results presented by our predecessors ([BG04a], [TZ03b])), is that certain IB decisions will get suppressed or disabled, although the overall prediction would be correct.

2. An other form of *unforced* user interactions we define as user inputs that otherwise would have been taken also within a short period of time by a device agent. Thus, depending on the situation, we can only sometimes refer certain user interactions as *unforced* since up to this time, we can hardly give a global tolerance that should be adhered, to comply to the definition of an *unforced* user input.

It should be noted that presumably most *unforced* user interactions only appear within this simulation since it is quite hard to model behavior to comply to such a particular problem. Hence, we recommend to deeply

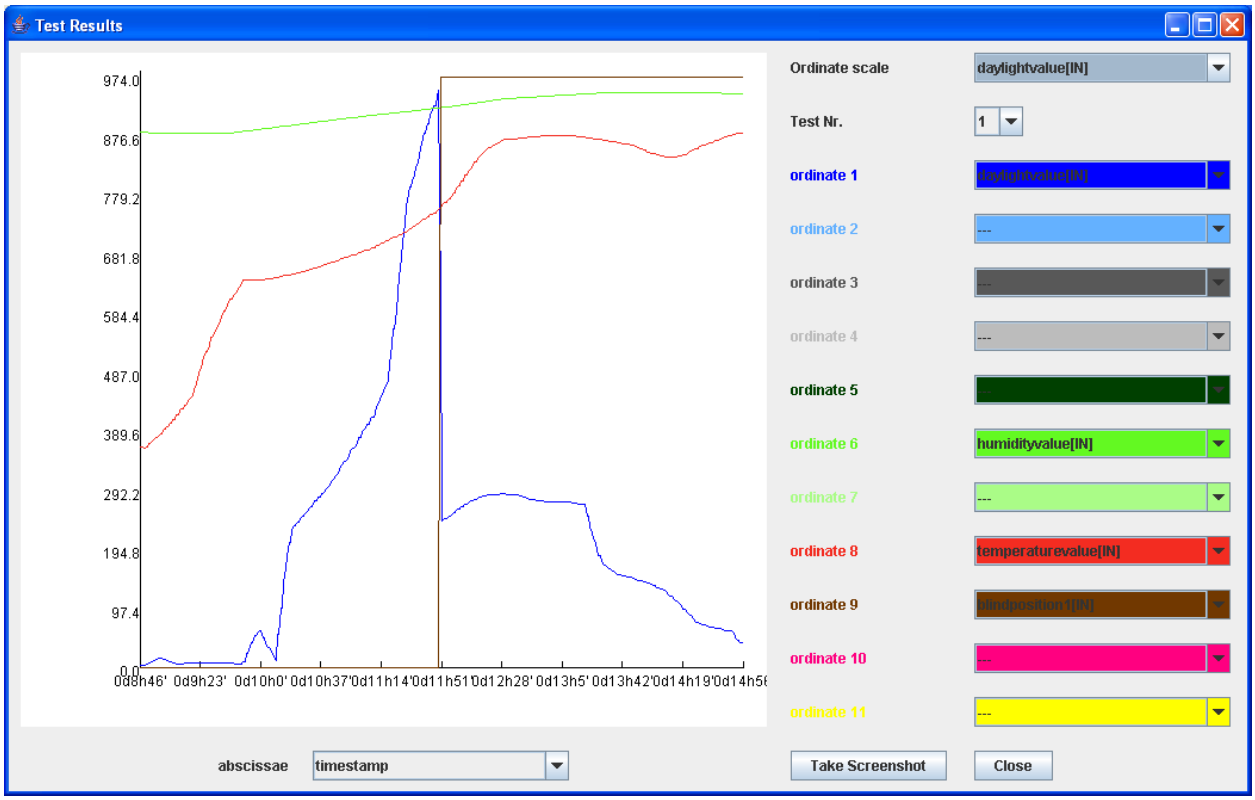


Figure 14.16: When zooming into the graph we can identify that that interior daylight is effected upon altering the blind position. Hereby blinds are that are up are shown on the abscissa. Inversely blinds that are down are show above.

analyzed each of the graph. Generally though such slip-ups should be more or less negligible. Concrete unforced user inputs will be given within the perimeter of later chapters.

Chapter 15

Result Viewer

In order to re-illustrate test results from past series, a new application has been developed that is capable of visualizing such data that has textually been stored by the framework.
Hereby any data can be plotted that complies to those simple rules:

- The data must be represented using columns
- The first value of each column should normally correspond to the column-name
- Each column must be separated by a tabulator

Herewith any data can conveniently be read and illustrated using graphs and accompanying legends, etc. (See figure: 15.1). Its quite advantageous since from an illustrative point of view quite simple in contrast to advanced tools such as Matlab or Maple.
One special benefit is that all data can be visualized since they have been scaled to meet a conjoint measure. Thus correlations among different inputs can easily be depicted and compared. Furthermore it is quite practical to perform any evaluation on the data since they can easily be swapped and re-plotted. Additionally data can be analyzed by zooming into distinct spots where further reasoning may take place. Not to mention that any documentation surely benefits from such concrete, generated figures since we no longer need to mess around with foreign tools anymore. In particular we found that it isn't quite easy to provide multiple variables with different scales to be displayed within one graph.

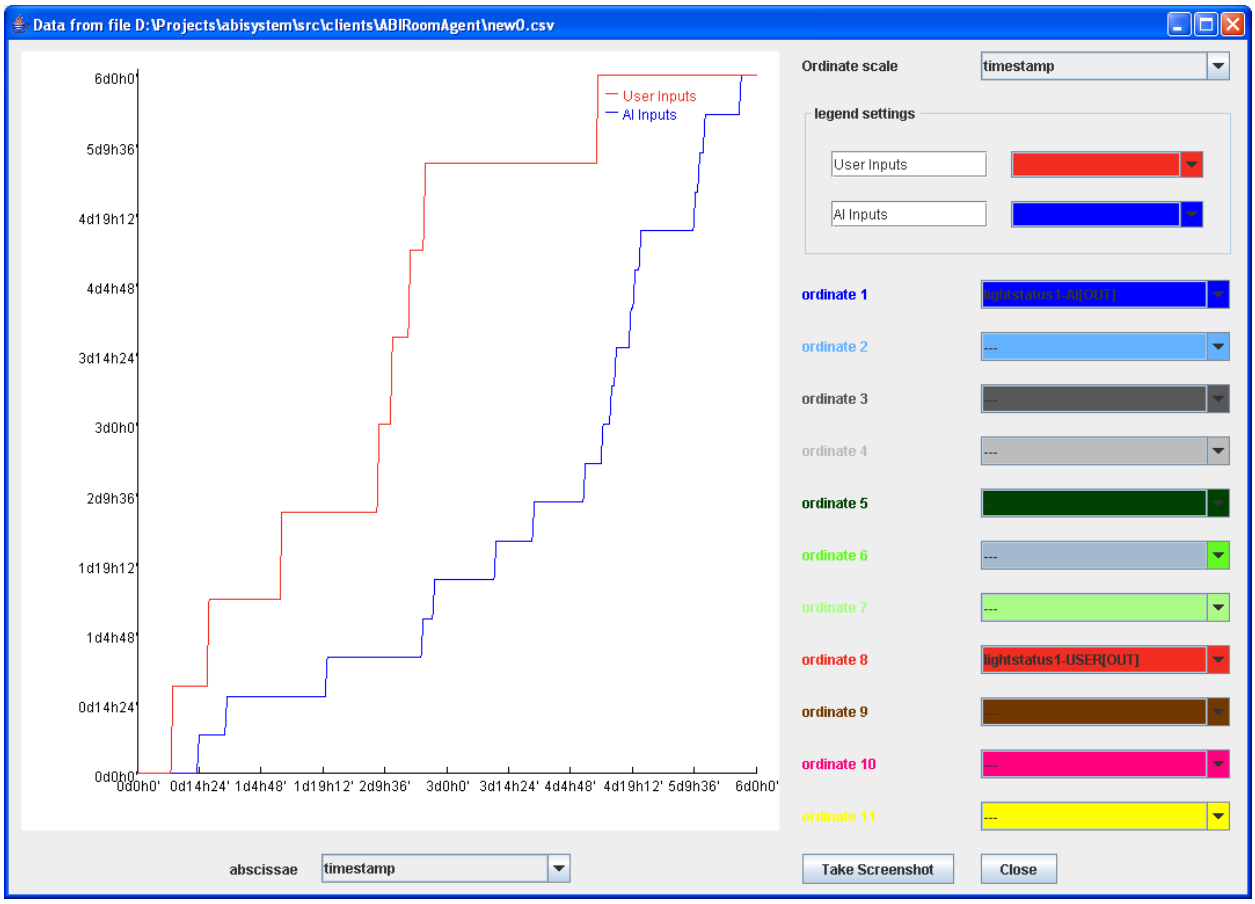


Figure 15.1: Additionally legends are provided which improve the screenshots quality

15.1 Result Viewer Usage

The utilization of this tool does more or less correspond to the simulator and will therefore not be explained anymore.
In contrast to the simulator though, a file must be provided at startup that must adhere to the rules stated above (See section: 15).

Part VI

Learning

Chapter 16

Introduction

Tom Mitchel [Mit97] mentioned in his book back in 1997 that:

> *When studying machine learning it is natural to wonder what general laws may govern machine (and nonmachine) learners. Is it possible to identify classes of learning problems that ar inherently difficult or easy, independent of the learning algorithm? Can one characterize the number of training examples necessary or sufficient to assure successful learning? Can one characterize the number of mistakes that a learner will make before learning the target function? Can one characterize the inherent computational complexity of classes of learning problems?*

We agreed upon that a general answers to all these questions are not yet known.

In regard to building intelligence, this part examines among other things a row of different AI algorithms. Chiefly we're interested in their strength and weaknesses as well as under which conditions or circumstances they have been proven to be successful in learning dynamic space behaviors. Herewith we try to compose a set of algorithms that we finally intend to incorporate into the IBF algorithm. The task of the IBF algorithm is then to compete them against each other and will filter out those algorithms which were rather wrong in their prediction and boosts algorithms that rather tended to be right. This is reasonable since each environment is different. - And likewise each algorithm!

According to the introduction chapter (See: 2) we illustrated what issues learning from a dynamic fuzzy rules set brings along. As announced we strike a totally different path on trying to make a building act intelligently.

We already discussed the term *building intelligence* as well as its major issues that have chiefly been addressed, identified and discussed back in chapter 11.
However when picking out some of the crucial inputs again that were stated, we bump into a conclusion that revealed that when being once competent of distinguishing between relevant from insignificant inputs as well as when being capable of keeping exceptional data over long periods we would potentiate any reproduction of such *"clustered"* data that might have been gathered even through across decades.
Thus improving any conventional prediction algorithms such as neural networks or other regression techniques to maximize their hypothesis since a lot of ambient noise should have already been filtered by the use of clustering.

Herewith we would even possible a solution that would master any possible dynamic structure as well as user behavior changes, in an already introduced concept that is known as *short* and *long-term memory* ([TZ03b]).

Hence the core approach of our learning technique, involves the examination and the development of algorithms that are capable of representing such knowledge in form of clusters. Each of which should hereby represent a variety of configurations that were encountered within different situations. Hence the first problem that we eventually need to tackle down is an automated clustering algorithm that doesn't presume any

premature quantity of such clusters.
It has been long known that many unsupervised learning techniques such as the k-means (See: 21.1.1) suffer from estimating the right number of k's and Kohonen networks (See chapter: 20) from providing the right number of neurons as well as to keep knowledge over long periods.

Before we introduce the developed cluster algorithms, we start by covering some of the conventional methods since astoundingly some of them achieved quite good results as well. Further, all advanced algorithms will be using both (clustering as well as conventional methods) and therewith necessary anyhow.

Practically each of the algorithms complies at least to some of the stated goals that we proposed to solve by an IB (See chapter: 1).
In respect to the framework algorithm (IBF algorithm) (See chapter: 12.2), we further prove that algorithms, which continuously succeed will contribute a major chunk to the overall prediction and thereby will improve the succession of each device agent.

16.1 Terms and definitions

For the sake of clarity we use the term space and environment to refer to a set of controlled entities that are located in a room of a building. Also note that the term neuron as well as unit is being used in same context.
Other terms and definitions are to be looked-up in the glossary (See chapter: 28).

16.2 Preconditions

This part presumes that the reader is already familiar with the following chapters: 1, 11, 13 and the following paper: [RJD04].

We also assume that the reader is a bit aware of the following diploma theses and term projects: ([NB05a], [TZ03b],[TZ03a], [RS02]).

Other papers that might be of interest are: [CCSZ21], [CZ16], [CZ65].

Other related term projects and diploma theses that have been conducted in reference to ABI but not really required: [Fen04], [NB05c], [NB05b], [BG04a], [BG04b].

16.3 Part structure

This part is structured into six chapters: An introduction chapter (See: 16), a Test-Settings and Data Pre-Processing chapter (See: 17), followed by a row of algorithm chapters (See: 18, 19, 20, 21, 22)

The Test-Settings and Data Pre-Processing chapter as the name implies, holds down the general test-settings that each of the algorithm will be using for conducting a couple of simulations. Further, we cover some fundamental aspects, how certain input variables have been prepared for the algorithms and most importantly we discuss which of the inputs have been skipped and which *"artificially"* created and added.
We then start to introduce each of the realized algorithm by adhering to the following sequence:

Overview: Within this section, we commonly give a rough introduction, how the algorithm principally works.

Algorithm-name: Within this section, we give a rather more accurate or formal description about the concrete algorithm. Commonly expressed on a mathematical basis.

Realization: This section usually engages, how the algorithm has been realized to be useful for ABI. Additional applications are usually enclosed that give a rather practical idea on how the algorithm behaves to meet our needs.

Evaluation and discussion: This section presents all relevant test results that are needed to discuss a strength or a distinct weakness about an algorithm.

Its important to note that each algorithm is plugged alone without competing with other ones. Overall results have been postponed to a later part since within this part we're only interested in their general principles, strengths, weaknesses and most importantly on their correct implementation.
Supplementary, we might even be capable of making a preliminary decision on not using an algorithm upon poor performance.

Please note that chapter: 18, 19 and 20 rather cover-up more or less standard algorithms and hence will provide a small repetition of some sort whereas chapter: 21 and 22 will be introducing the previously mentioned cluster algorithms that provide some sort of hybrid solution to the IB problem. Hybrid in the sense that unsupervised learning as well as supervised learning is applied. Since we perform clustering we don't actually need to specify what we are trying to learn. The gained clusters will then commonly be used to perform a *task evaluation*. *Task-based evaluation*, will then commonly be conducted by some standard supervised algorithm such as neural network or other regression technique that can be improved by using the output of the gained clusters.

This thesis assumes different type of readers. Hence, one or the other might already be familiar with some of the presented standard algorithms and thus will most likely rather tend to be repetitive.
Nevertheless we recommend to skimm through each of the result sections.
Please also take note that the goals of this thesis are rather to compare different learning algorithms then trying to optimize each of them to perform at their best. Hence we skipped on doing any detailed analysis to figure out their optimal parameters.
Anyhow for those which already possess a solid knowledge in machine learning might rather skip to the major chapters such as: 21 and 22.

Chapter 17

Test-Settings and Data Pre-Processing

17.1 Test-Configuration

All algorithms will be conducted using following global settings when not explicitly stated otherwise.

17.1.1 User profiles

Hereby we abuse the term *personas* to refer to fictional users that comply to the following description that has also been suggested by Cooper:

> *Personas are archetypal users that represent the needs of larger groups of users, in terms of their goals and personal characteristics. They act as 'stand-ins' for real users and help guide decisions about functionality and design*([Per]).

In order to bring in some behavioral changes along the tests we defined two different personas that will be used by the algorithm tests.

Dave: is a 29 years old software engineer that is working in a company which provide flextime. Dave hardly makes use of the flextime though since he's rather accustomed of keeping a fixed number of working hours. However he doesn't like to get up early and rather tends to working late. He's more of night person. Twice a week he's having a session with his team and thus won't be sitting at his desk during this time. Despite of that his office is rather located at the east side and thus is sparsely facing any direct sun light, he barely switches on the lights since he likes to work with less synthetic lighting. Therefore his office is almost quit dark in contrast to some other adjacent offices.

Helen: is a 46 years old secretary and possesses a huge antipathy for darkness. Hence during cold, fogy days she always has both lights on. But since she's sharing the office with a co-worker which rather tends to let some sun-light into their office they eventually must arrange to a common configuration. Generally though they agreed upon moving up the blinds and subsequently switching off the lights as well as around the summer time when temperatures above 20 degrees can be observed and the sky is almost free of any clouds.

Superficially speaking both users can basically be categorized into a dark and bright a persona. However they still have some minor distinguishable differences such as their work habits which ultimately will also impact a rooms occupancy status.
According to the latter defined personas we can derive following configurations.

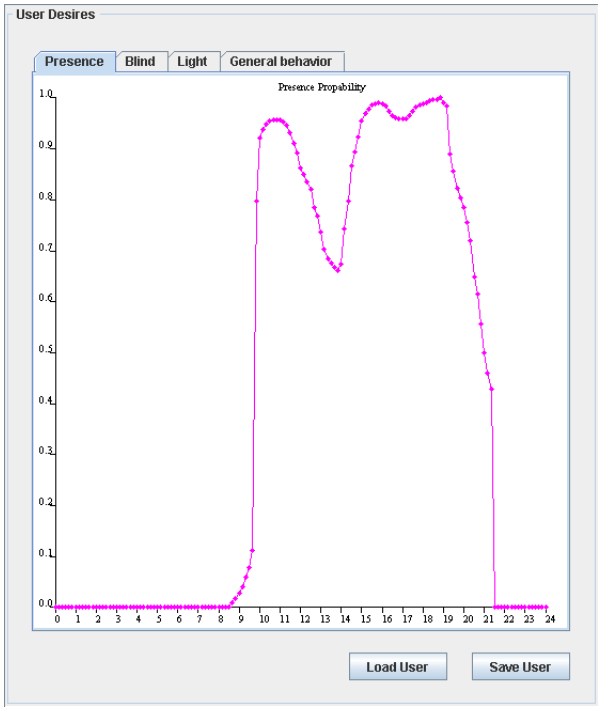


Figure 17.1: Dave: Presence preferences

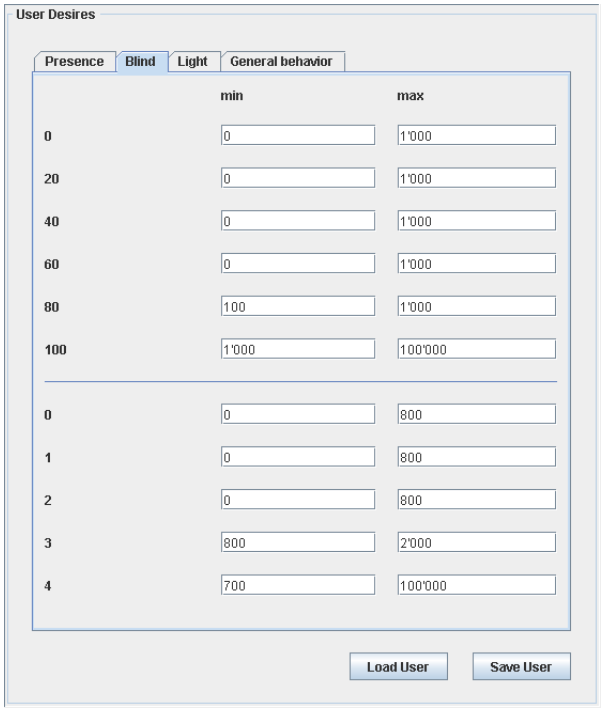


Figure 17.2: Dave: Blind preferences

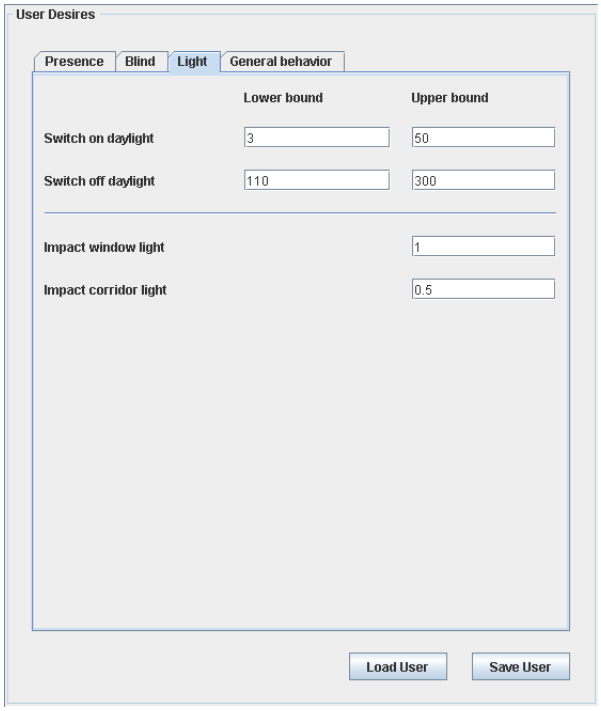


Figure 17.3: Dave: Light preferences

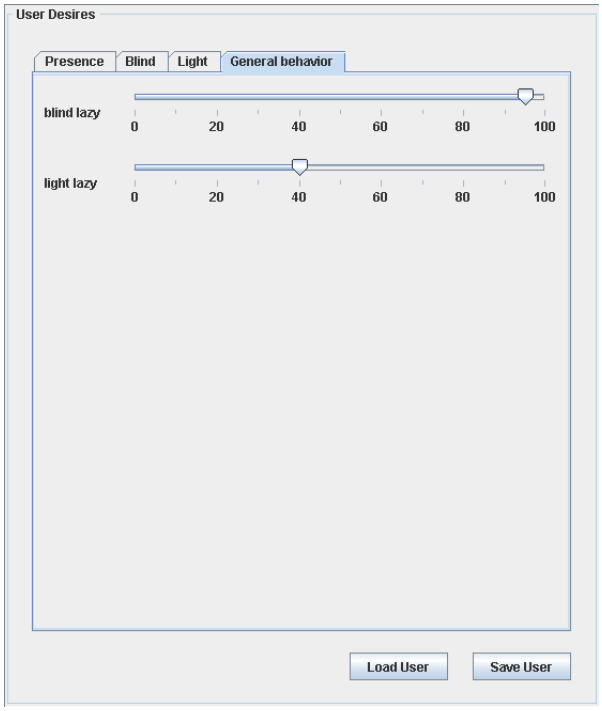


Figure 17.4: Dave: General preferences

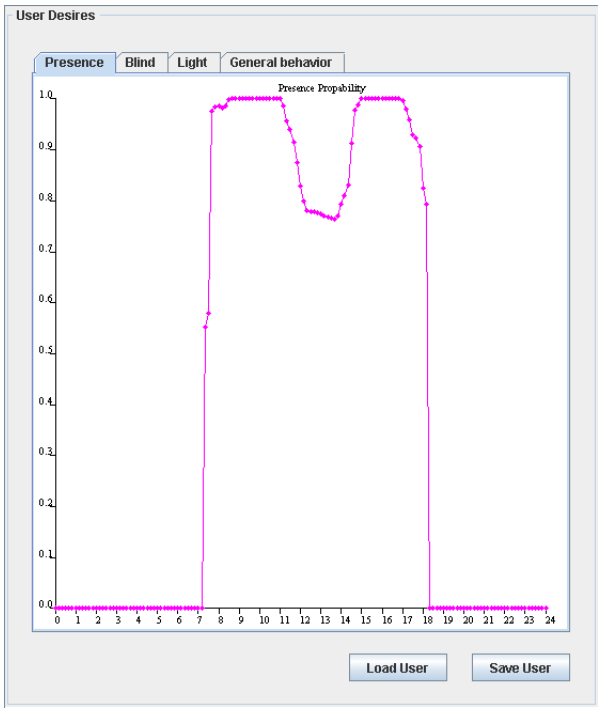


Figure 17.5: Helen: Presence preferences

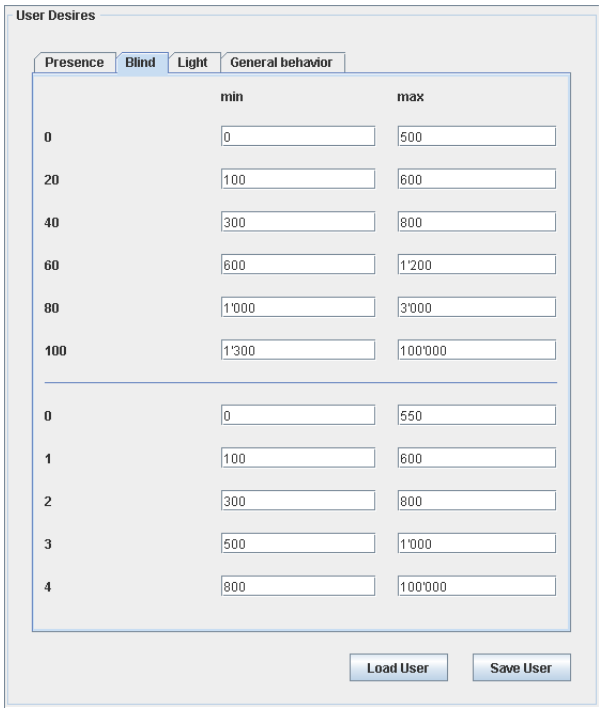


Figure 17.6: Helen: Blind preferences

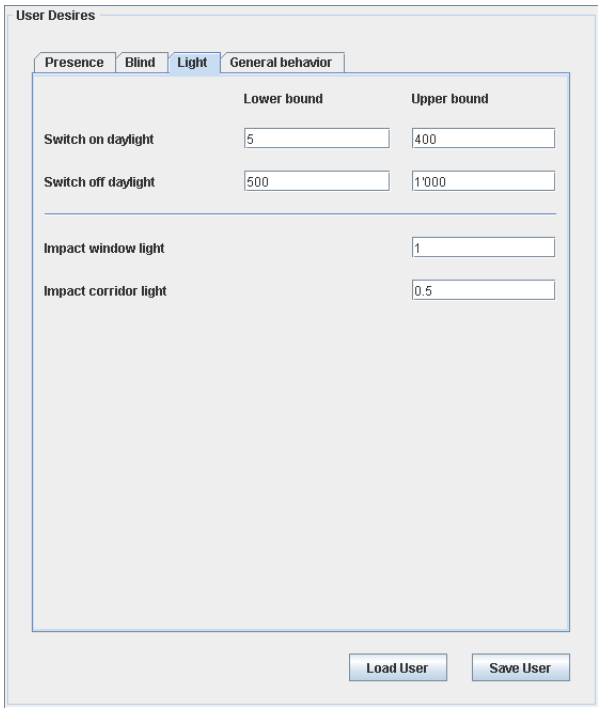


Figure 17.7: Helen: Light preferences

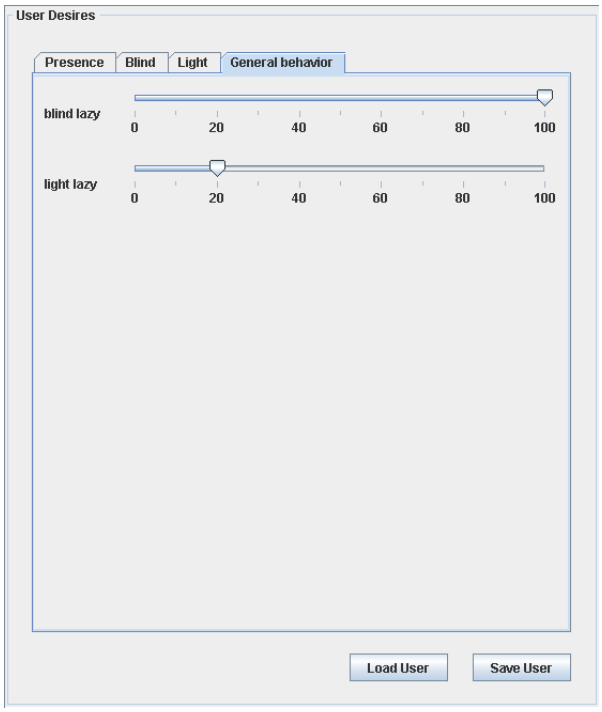


Figure 17.8: Helen: General preferences

17.1.2 Simulation Tests

The algorithms will be tested with one major standard test over a 60 day period. Environmental changes have been tried to take into account as far as possible. However its quite difficult and most of all cumbersome to plot each sensory input manually, day by day. In the future we will be using real life data from the INI

to set a least the sensory inputs. Currently though we haven't been able to collect enough data to do so. Not to mention that we wouldn't have found extra time to implement such an extension to the simulator anyway.

The first test row that synthetically has been created with the simulator is illustrated in figure: 17.9.

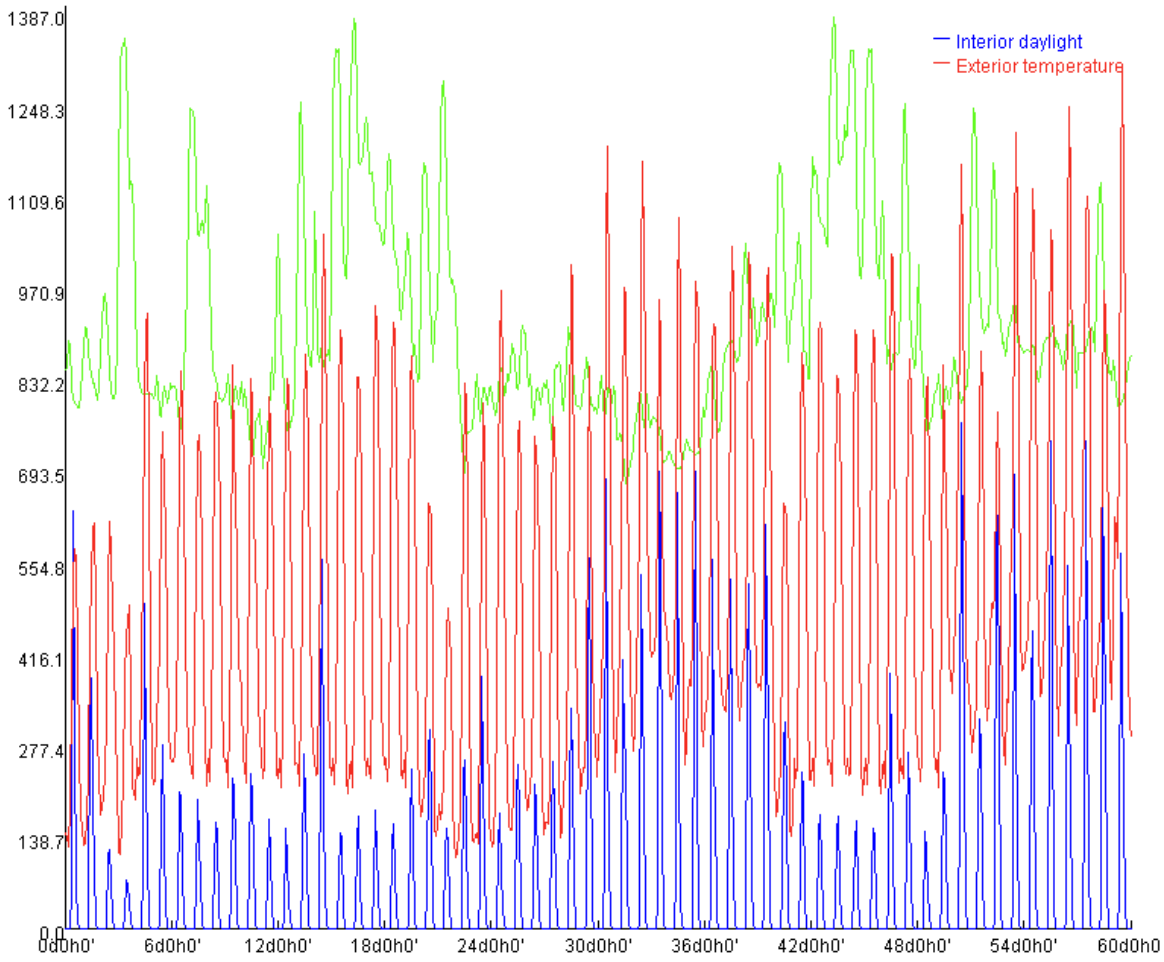


Figure 17.9: 60 day standard test. As we can see two bright day rows have been included to make it a little chunkier. Hereby the blue curve corresponds to the interior daylight, the red to the exterior temperature and the green curve depicts the humidity. The ordinate is set to the scale of the interior daylight (lux).

Due that **Dave** exhibits less interesting characteristics, we'll be taking **Helen** as our main test persona, in the test illustrated in figure: 17.9.

In some of the algorithms we're particularly interested in how they react to sudden changing customs. For this reason we setup a simple test that features a constant change of the sensory inputs rather then incorporating natural looks. This is more adequate for us since we're only interested in their un-learn and re-learn strengths or weaknesses. Hence, we commonly swap users (**Helen** and **Dave**) occasionally in order to visualize this particular capability.
The test row is depicted in figure: 17.10

It may be worth to explain why the blind laziness has been chosen so high. The reason is that blind behaviors are very difficult to determine since no real intelligence is behind all this. In the future we might consider an entire integrated intelligence that can be set with rules or so. However for the sake of simplicity we didn't

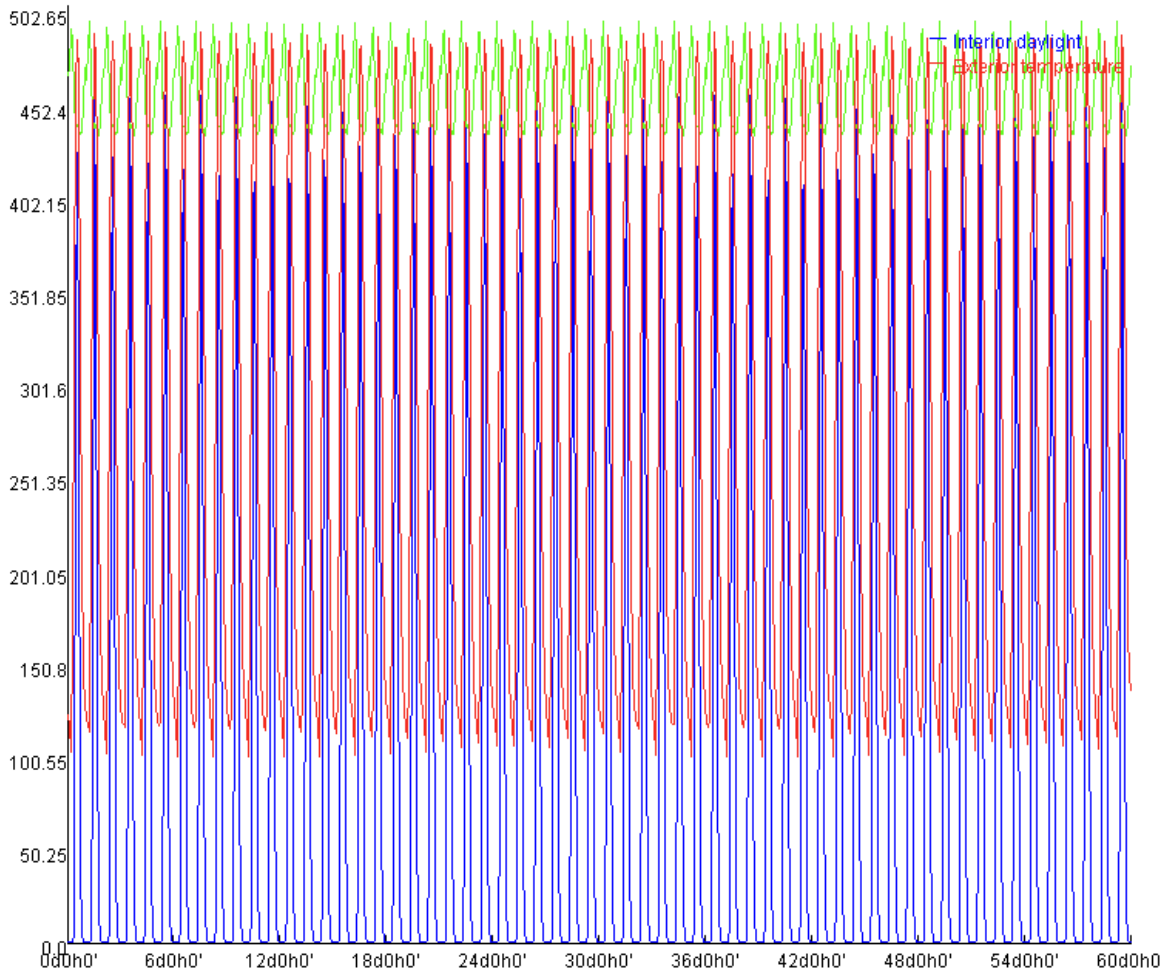


Figure 17.10: 60 day standard test. Repetitive days to facilitate the recognition of changing user behaviors. The first 20 days we start with Helen, the next 20 days with Dave and for the last 20 day Helen again.

do that. It has been proven that blind motions within the INI are rather sparse in nature. Especially at nights the blinds are not moved up or so. A proper imitation is practically impossible right now within the simulator. Nevertheless to minimize the movements we set it as lazy as possible. However wrong mimicked blind movements may cause wrong patterns to be produced which might cause gradient descent algorithms such as the backpropagation network (See chapter: 19) or even cluster algorithms such as the k-means (See section: 21.1.1) or the g-means (See chapter: 21) to get trapped in local minimas.
Additionally, it should be noted that the simulator is actually one of the best suited platforms for algorithms which only consider the interior daylight and the blind position or even the blind rotation as the major measure.
This is because each synthetic user, that is mimicked with the simulator, only considers the interior daylight value, which is again (according to section: 14.2.2) calculated through the dynamic blind position and rotation and of course the configured exterior daylight.
Hence, the humidity as well as temperature don't play any crucial role for the synthetic users since each decision is depending on only the interior daylight. Therefore we expect that conventional algorithms such as the ordinary backpropagation artificial neural network as well as the statistical learning to perform at their best. Hence, such algorithms will basically benefit from such a setup since they will either don't consider other inputs in the first place, such as the statistical learning or will pretty soon learn that only the interior daylight will be of crucial importance (sort of overfitting).
The major backlash in the simulator will therefore need to be taken by other algorithms such as the cluster

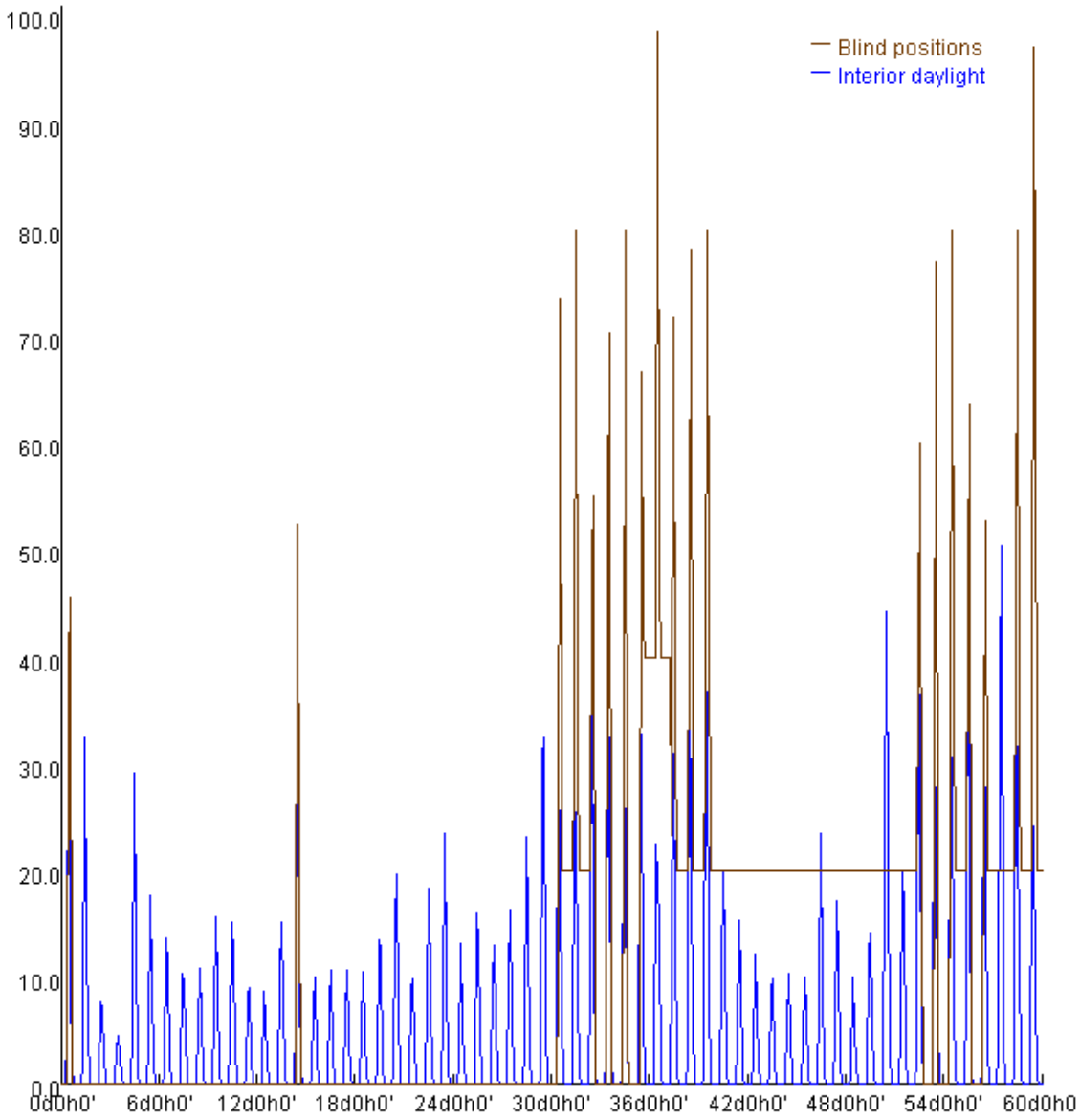


Figure 17.11: The brown spikes depicted here correspond to the blind position. Hereby 100 means down and 0 up.
We notice that although the blind laziness has been set high, the movements are still to frequent which might impact the generalizability for certain learning approaches to get stuck in local minimas.

algorithms. They will surely suffer from this fact since clustering is usually performed, considering all parameters and will therefore not cause good results to be achieved within this simulator since crucial dependencies heavily depends on each test.

Please note that any simulator constraints e.g about its particular usage such as how those values have been configured, should be looked-up in the corresponding simulator sections: 13.

17.1.3 Data Pre-Processing

Most of it has already been mentioned in part: IV. Hence we make it short. When not stated otherwise we'll be using following inputs for the learning tasks:

- All interior daylight sensors
- All available blind positions
- All available blind rotations
- Temperature
- Humidity
- Filtered interior daylight

Output from other lights as well as the exterior daylight have been skipped but may be appropriate at a later stage.

The filtered interior daylight provides a softened or filtered version of the regular interior daylight. This will help some algorithms to zip past small fluctuations within the error surface or also to just lessen sudden AI interactions, caused by sharp spikes.
Hereby every minute we poll for interior daylight values which we then use to calculate a form of filtered daylight values by applying the simple formula introduced in one of the past chapters (See: 11.1).

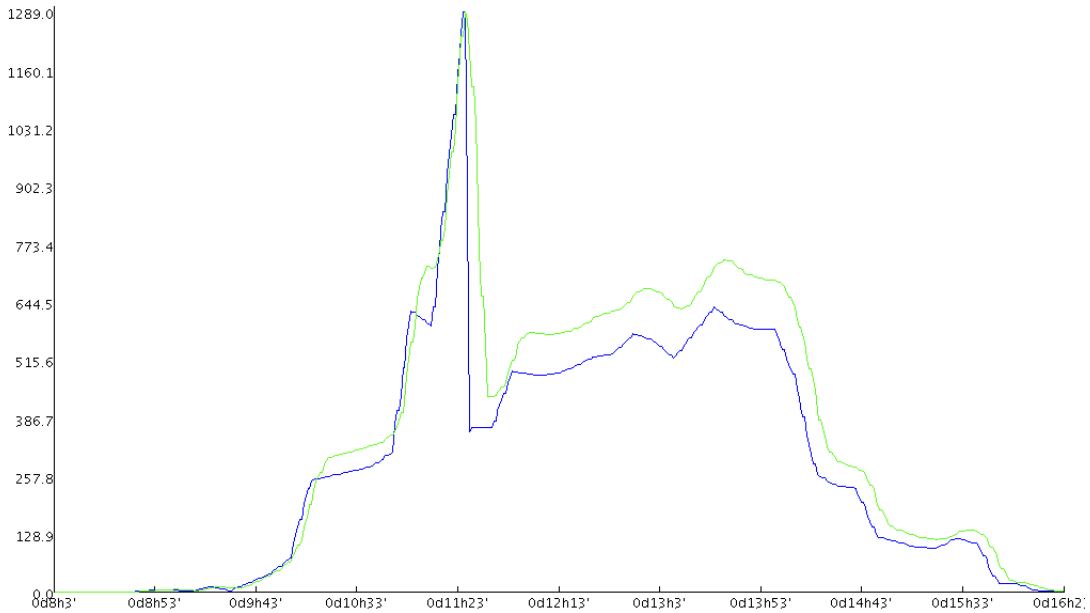


Figure 17.12: Filtered daylight using a history size of 10 within the time period (now - 10 minutes)

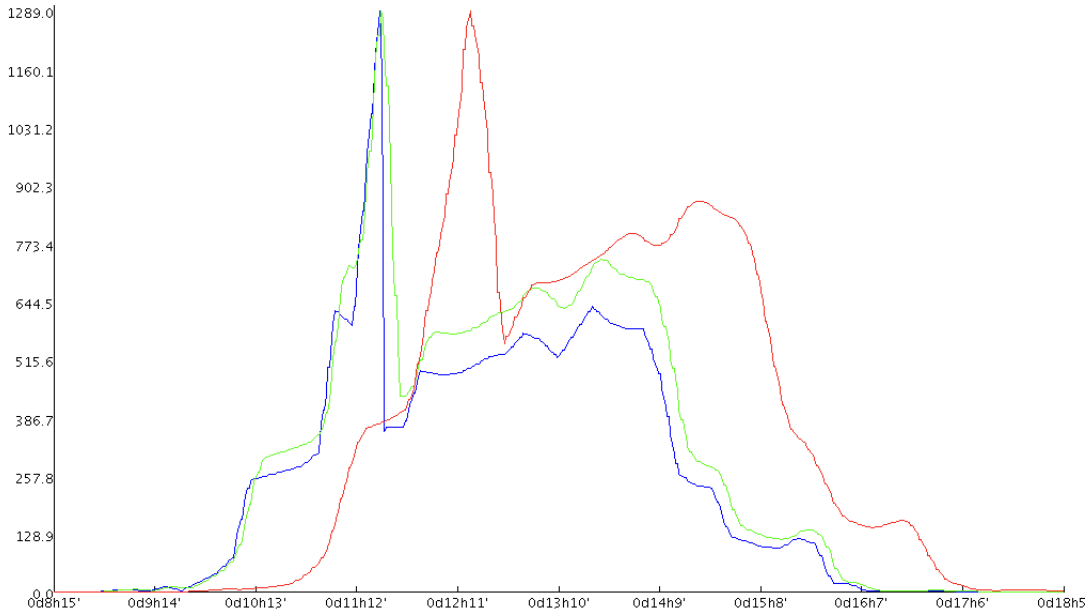


Figure 17.13: The red curve corresponds to the filtered daylight, using a history size of 70, within the time period (now - 70) and (now - 50). The blue curve denotes the one that has already been illustrated in figure: 17.12.

Chapter 18

Statistical Learning

In this first simple approach we extend the initially conducted statistical analysis (See section: 11.3) in the form of applying a well know curve fitting procedure: *A polynomial n-th degree*. Before discussing the curve fitting we present the adapted algorithm already discussed in the section: 11.3.

Result: Light ON/OFF statistic per slot s
input: A set of continuous input-pairs $\Psi\ \{x, y\}, x \in \mathrm{S}, y \in \{true, false\}$
foreach *slot s in S* **do**
    set probability $\sigma_s \leftarrow 0.5$
end
set learn factor (momentum) α to a value between 0 and 1
repeat
    fetch next inputpair
    if $y = true$ **then**
        $\sigma_x \leftarrow \sigma_x * \alpha + (1 - \alpha)$
    else
        $\sigma_x \leftarrow \sigma_x * \alpha$
    end
until *Stop condition reached*

Algorithm 4: Simple statistical learning

We see that the algorithm depicted in algorithm: 4, shows that a certain "momentum value", α has been applied in the probability calculation. This value determines how agile we need to react upon trend changes. For instance when a corridor light was on all the time and all the sudden needs to be switched off. Hence the value α will help us to either consider past or recent values before affecting the probability σ of a timeslot s. In the future we might consider this value to be computed as a function of the user input (Example see section: 12.2).

18.1 Overview

One of the most common procedures when it comes down to curve fitting is the standard polynomial curve fitting.
Curve fitting is one of the simplest example of supervised learning of a function. In practice we usually want to approximate curves in regard to the raw field data. However as for most given data-sets, a standard polynomial curve fitting approach isn't sufficient enough since we're actually interested in a rather generalized form that cardinally doesn't overfit. Overfitting is one of the most common problems in regression whether or not applying one of the best curve fitting procedures such as the method of least squares.

Nevertheless the method of least squares increases the generalizability remarkably since it surely suppresses

a huge chunk of overfitting. Although care must be taken when applying the *least-squares m^{th} degree polynomials*. Since upon increasing the number of degrees of freedom we indeed obtain a better fit but on the other hand we might rather tend to overfit and hereby violate the capability of the polynomial to generalize new data. As a result the approximated function will turn into some dramatic oscillations and hence provides a poor representation of the required hypothesis function $h(x)$.

18.2 Polynomial Regression

Consider the *m^{th}-order polynomial* given by:

$$y(x) = a_0 + a_1 x + a_2 x^2 + ... + a_m x^m = \sum_{j=0}^{m} a_j x^j \tag{18.1}$$

Equation: 18.1 can be regarded as a non-linear mapping which takes x as input and produces y as output. Hereby the precise form of the function $y(x)$ is determined by the values of the coefficients $a_0, ...a_m$ which is also commonly represented as a vector, $\vec{a}$. Hence, the polynomial can be written as a functional mapping in the form of $y = y(x; \vec{a})$. The target data point that we wish to approximate we denote as t like target value. According to the latter section we are interested in finding suitable values for the coefficients in the polynomial and therefore consider the error between the desired output t_n for a particular input x_n, and the corresponding value predicted by the polynomial $y = y(x; \vec{a})$.

$$E = \sum_{n=0}^{N} \{y(x_n; \vec{a}) - t_n\}^2 \tag{18.2}$$

In other words when applying the m^{th} degree polynomial to approximate a given data-set the best fitting curve $y(x; \vec{a})$ has the least square error:

$$\Pi = \sum_{n=1}^{N} \{y(x_n; \vec{a}) - t_n\}^2 = \sum_{n=1}^{N} \left[\sum_{j=0}^{m} a_j x_n^j - t_n \right]^2 = min \tag{18.3}$$

To obtain the least square error, the unknown coefficients $a_0, ...a_m$ must yield zero. We receive each of the desired coefficients by taking the first partial derivatives (See section: 18.3).

18.3 Realization

The applications of the method of least squares curve fitting using polynomials have been applied to both statistics (See figures: 18.1 and 18.2):

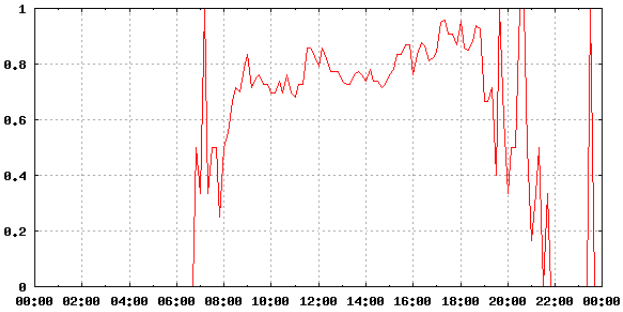


Figure 18.1: $\mathcal{P}(CORRIDOR\ LIGHT\ ON|s_t)$

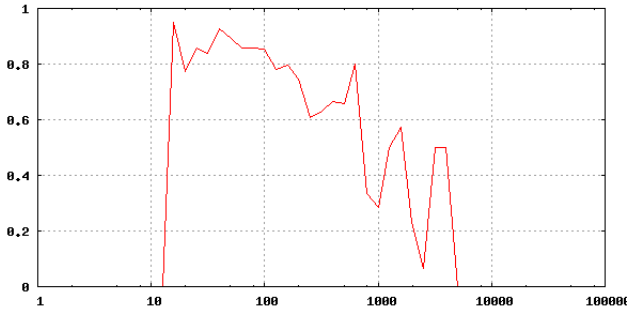


Figure 18.2: $\mathcal{P}(CORRIDOR\ LIGHT\ ON|s_d)$

Since a curve with a minimal deviation from all data points is desired, following equation can be extracted that we could actually use to implement the curve fitting.

$$\begin{aligned}
\frac{\partial \Pi}{\partial a_0} &= 2\sum_{n=1}^{N}\left[\sum_{j=0}^{m} a_j x_n^j - t_n\right] = 0 \\
\frac{\partial \Pi}{\partial a_1} &= 2\sum_{n=1}^{N} x_n\left[\sum_{j=0}^{m} a_j x_n^j - t_n\right] = 0 \\
&\vdots \\
\frac{\partial \Pi}{\partial a_m} &= 2\sum_{n=1}^{N} x_n^m\left[\sum_{j=0}^{m} a_j x_n^j - t_n\right] = 0
\end{aligned} \tag{18.4}$$

By solving the equations above, we will receive a set of linear equations that can then easily be calculated using the standard gaussian-jordan method.

$$\begin{aligned}
\sum_{n=1}^{N} y_n &= \sum_{i=0}^{m} a_i\left[\sum_{n=1}^{N} x_n^i\right] \\
\sum_{n=1}^{N} x_n y_n &= \sum_{i=0}^{m} a_i\left[\sum_{n=1}^{N} x_n^{i+1}\right] \\
&\vdots \\
\sum_{n=1}^{N} x_n^m y_n &= \sum_{i=0}^{m} a_i\left[\sum_{n=1}^{N} x_n^{i+m}\right]
\end{aligned} \tag{18.5}$$

It might be worth to mention that we've developed a library that implements the method of the least-squares m^{th} degree polynomials as well as a gaussian-jordan algorithm that solves the linear equations in our own since at that time we didn't find any suitable open source math tool library to utilize.
In order to test our library we implemented a small test-bed application in which we've fed the data used to display the graphs above: 18.1 and 18.2.

The new graphs which have been fitted following the least-squares m^{th} degree polynomials method are illustrated below (See figures: 18.3 and 18.4).

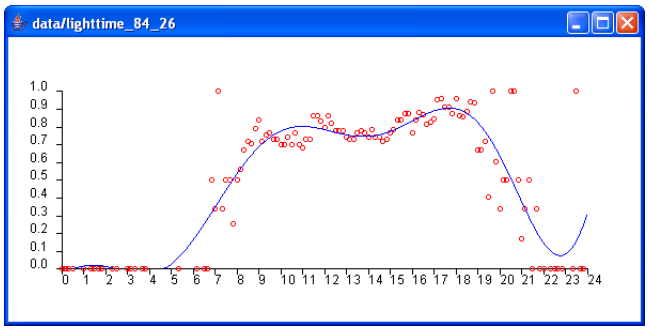


Figure 18.3: LS 10^{th} degree polynomials: $\mathcal{P}(CORRIDOR\ LIGHT\ ON|s_t)$

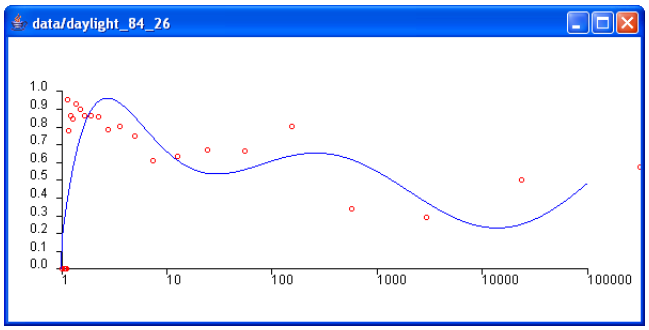


Figure 18.4: LS 10^{th} degree polynomials: $\mathcal{P}(CORRIDOR\ LIGHT\ ON|s_d)$

There are several methods to improve the method of the *least-squares m^{th} degree polynomials*.
Since we're already tracking some sort of trust value with us we might benefit from it by altering the *least*

squares (LS) method a bit in hope to gain some positive influence to improve our regression.
The improved method is known by the name: *weighted least squares (WLS)* regression. Unlike the *least squares*, however, each term in the *weighted least squares* criterion includes an additional weight, w_n that determines how much each observation in the data-set influences the final parameter estimates. The weighted least squares criterion that is minimized to obtain the parameter estimates is:

$$\Pi = \sum_{n=1}^{N} w_n \left\{ y(x_n; \vec{a}) - t_n \right\}^2 = \sum_{n=1}^{N} w_n \left[\sum_{j=0}^{m} a_j x_n^j - t_n \right]^2 = min \tag{18.6}$$

We've been testing and comparing both methods but the achieved improvements were rather moderate then what we've been expecting. (See figure:18.5, and figure:18.6).

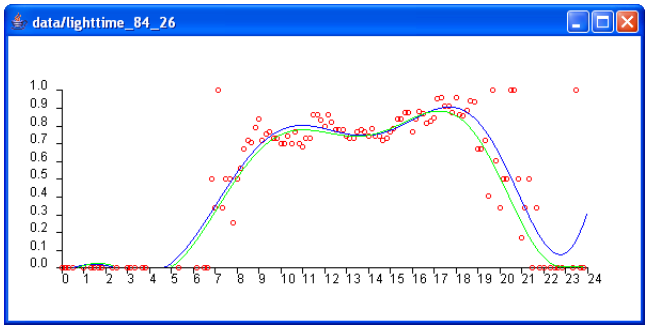


Figure 18.5: Green=WLS vs. Blue=LS, 10^{th} degree polynomials: $\mathcal{P}(CORRIDOR\ LIGHT\ ON|s_t)$

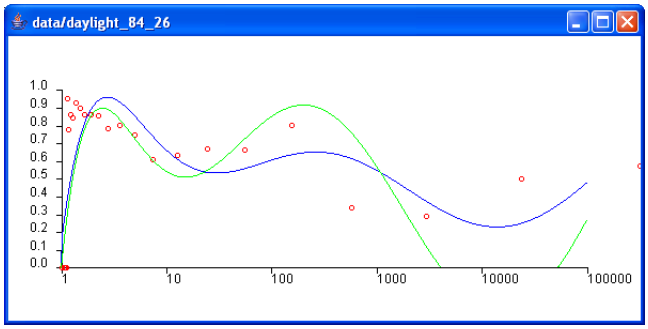


Figure 18.6: Green=WLS vs. Blue=LS, 10^{th} degree polynomials: $\mathcal{P}(CORRIDOR\ LIGHT\ ON|s_d)$

Another problem that has already been mentioned in the beginning of this chapter is the problem of overfitting and generalization.
Currently we deal with 10^{th} degree polynomials since we achieved pretty good results with them. Nevertheless in order to figure out a decent number of freedoms which do not tend to overfit, one would need to consider the root-mean-square (RMS) (See equation: 18.7) that provides a better way to calculate the effectiveness of a polynomial since the strong dependence on the number of data points will be removed. Furthermore when we would consider two different sets to be used typically one for training and another one for testing then we could detect a possible overfit.

$$E_{rms} = \sqrt{\frac{1}{N} \sum_{n=1}^{N} \left\{ y(x_n; \vec{a}) - t_n \right\}^2} \tag{18.7}$$

Now we've been discussing how to basically curve fit each of the statistic. Now we simply tie them together at one point by applying:

$$P_{ON} = \frac{\mathcal{P}(LIGHT_{ON}|s_t) \cdot \mathcal{P}(LIGHT_{ON}|s_d)}{(1 - \mathcal{P}(LIGHT_{ON}|s_t)) \cdot (1 - \mathcal{P}(LIGHT_{ON}|s_d)) + \mathcal{P}(LIGHT_{ON}|s_t) \cdot \mathcal{P}(LIGHT_{ON}|s_d)} \tag{18.8}$$

It should be noted again that this algorithm only considered the inputs: *daytime* and *daylight* as well as the effector output that momentarily corresponds to one light. Hence other statistics would have been possible but due to prove a common hypothesis, namely that certain rooms can simply be predicted by only taking these two variables into account. Nevertheless for other environments which are more complex in nature such a simple statistic simply fails to approximate a decent environment function by using just a few variables and must therefore be approximated otherwise.

18.4 Evaluation and Discussion

In the beginning of this chapter we discussed how strong the predictions can be acclimatized by configuring the momentum value α.
Hence we conducted four different tests to see how a statistical learning performs in our real-time simulation test-bed.
Possible future prediction might consider other recorded and fitted probabilities. However within this thesis we forgo other possible statistics such as blind position vs. daylight or such.

The first test row has been conducted using the previously defined standard test over 60 days.

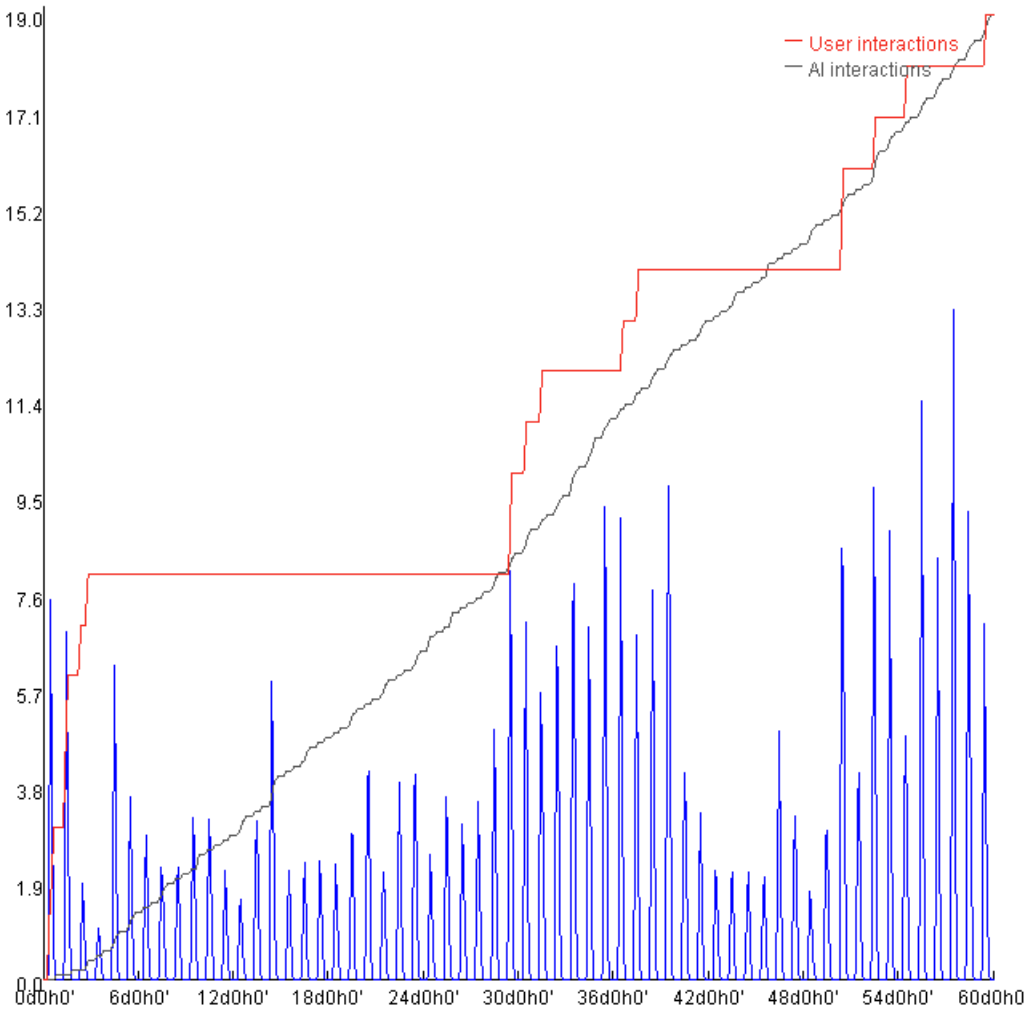


Figure 18.7: The performance of this algorithm in the simulator is quite remarkable. We notice that a low α, (α=0.6) will get accustomed to environmental changes quite fast but will result in an unprecise learning, that ultimately will have a drawback in a making proper predictions later on. The ordinate is set to the number of user interactions which currently would correspond to 11 within a period of 60 days. Ordinate: User interactions

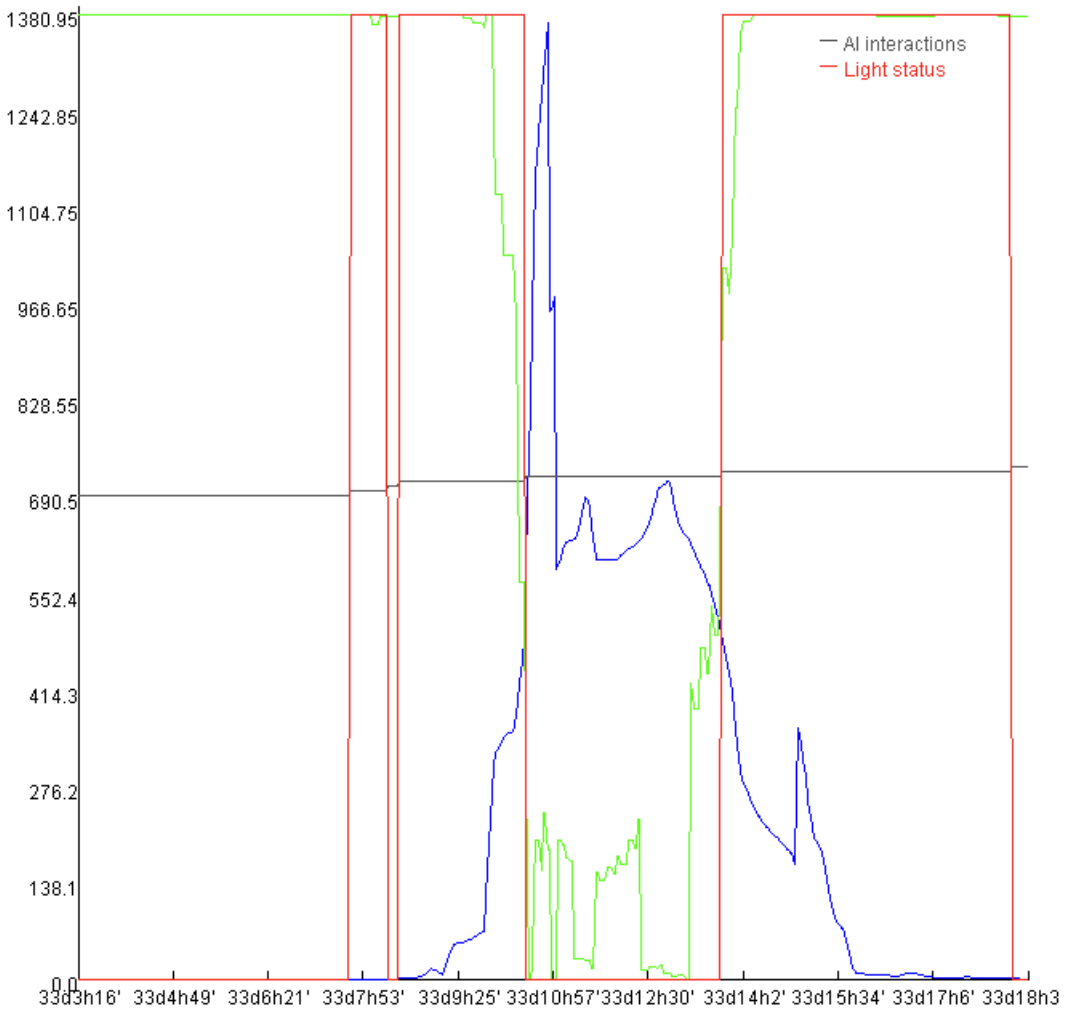


Figure 18.8: A perfectly predicted day. Hereby the blue curve depicts the interior daylight value whereas the green curve illustrates the prediction which turned out to be very successful. Take notice of the first light on/off hit that indicates that all occupants have left the room and thus energy has been saved by turning the lights off. We can further depict that the threshold is approximately around 550 lux which is quite normal, when comparing to the user profile illustrated in section: 17.1.1. Further we can denote that the steep spike is an indicative for blind movements. Probably down in order to block direct sun light. Parameters: α=0.6, Ordinate: Interior daylight

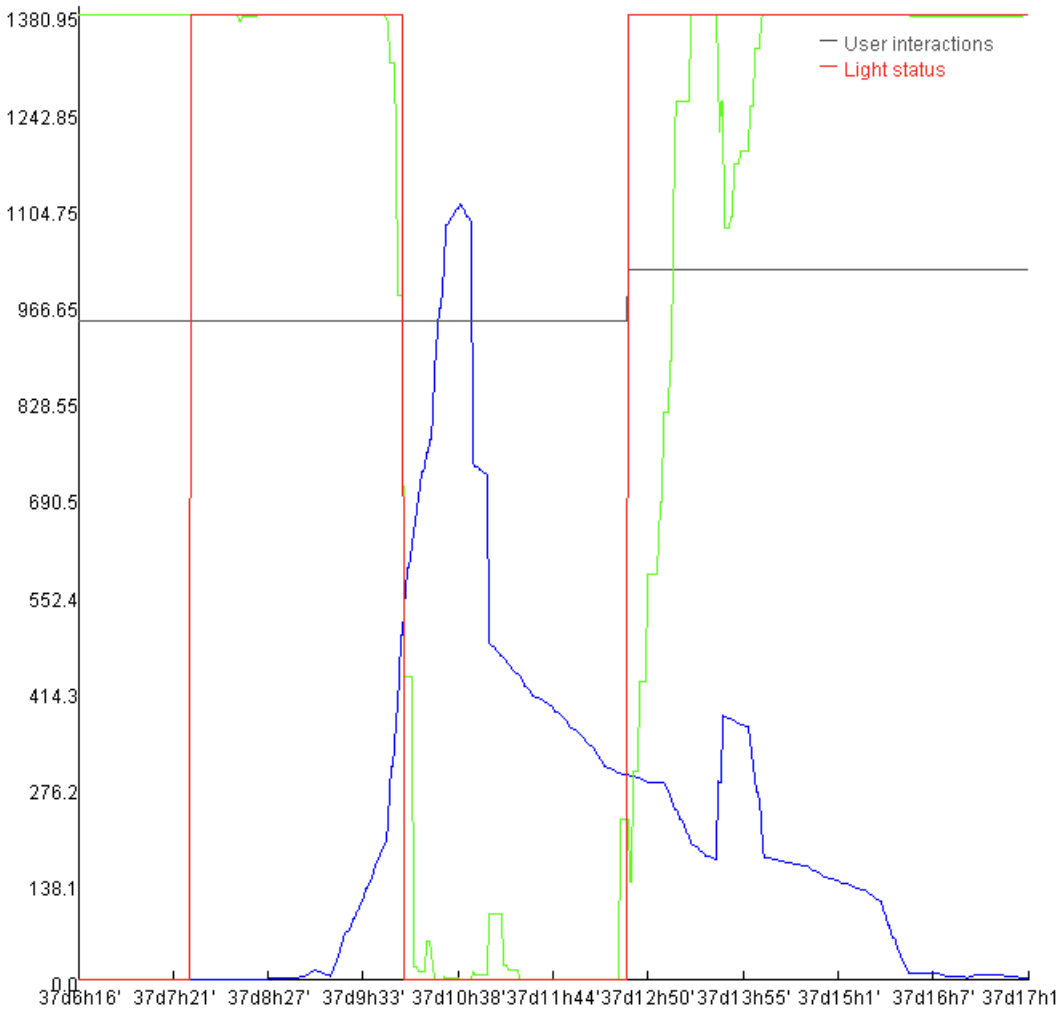


Figure 18.9: In contrast to the latter graph this graph illustrates a rather bad predicted day. This might be the cause of the low chosen momentum value α=0.6 (α=0.6) since the start-up threshold has been predicted way too low (Roughly around 270 lux). Hence the light should have been turned on at the latest around 300 lux. The off prediction though was just fine and reasonable. Ordinate: Interior daylight

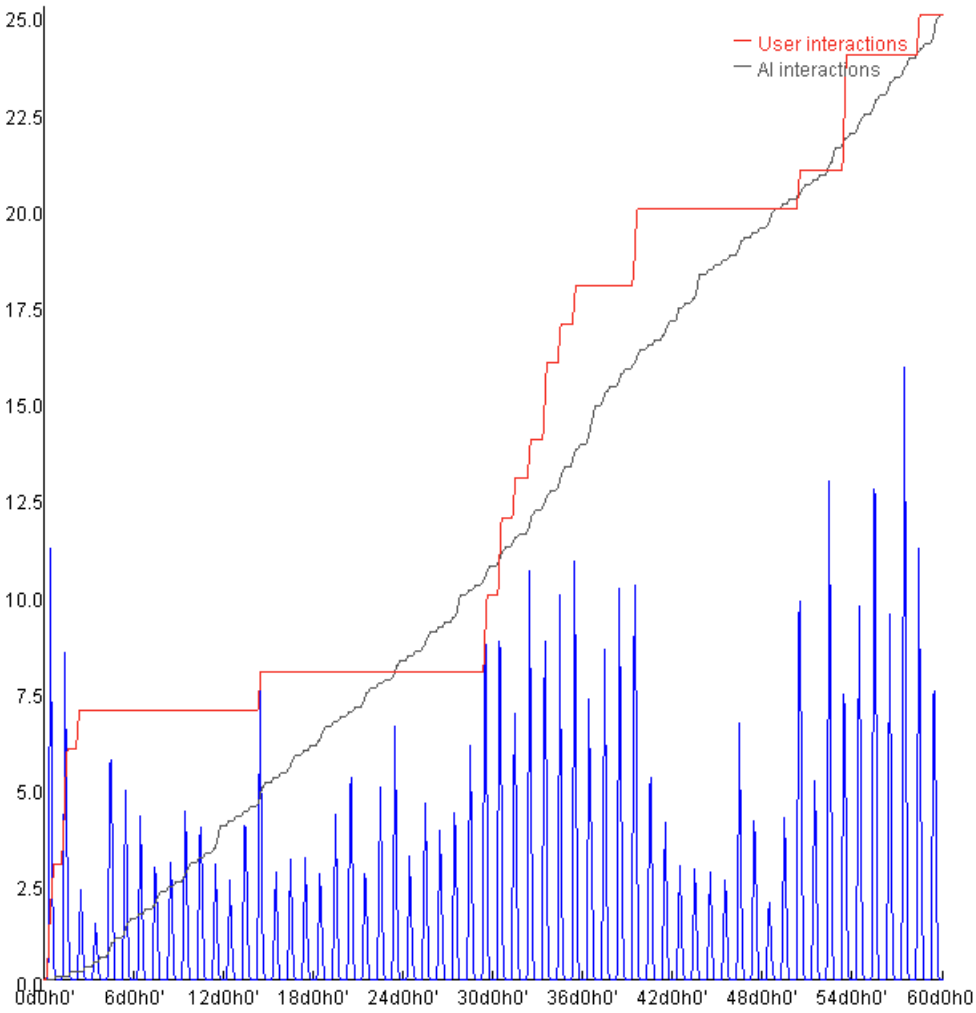


Figure 18.10: As expected we notice that a high α (α=0.9) in fact result in laziness but will eventually benefit from it by providing a more steady prediction later on. Ordinate: User interactions

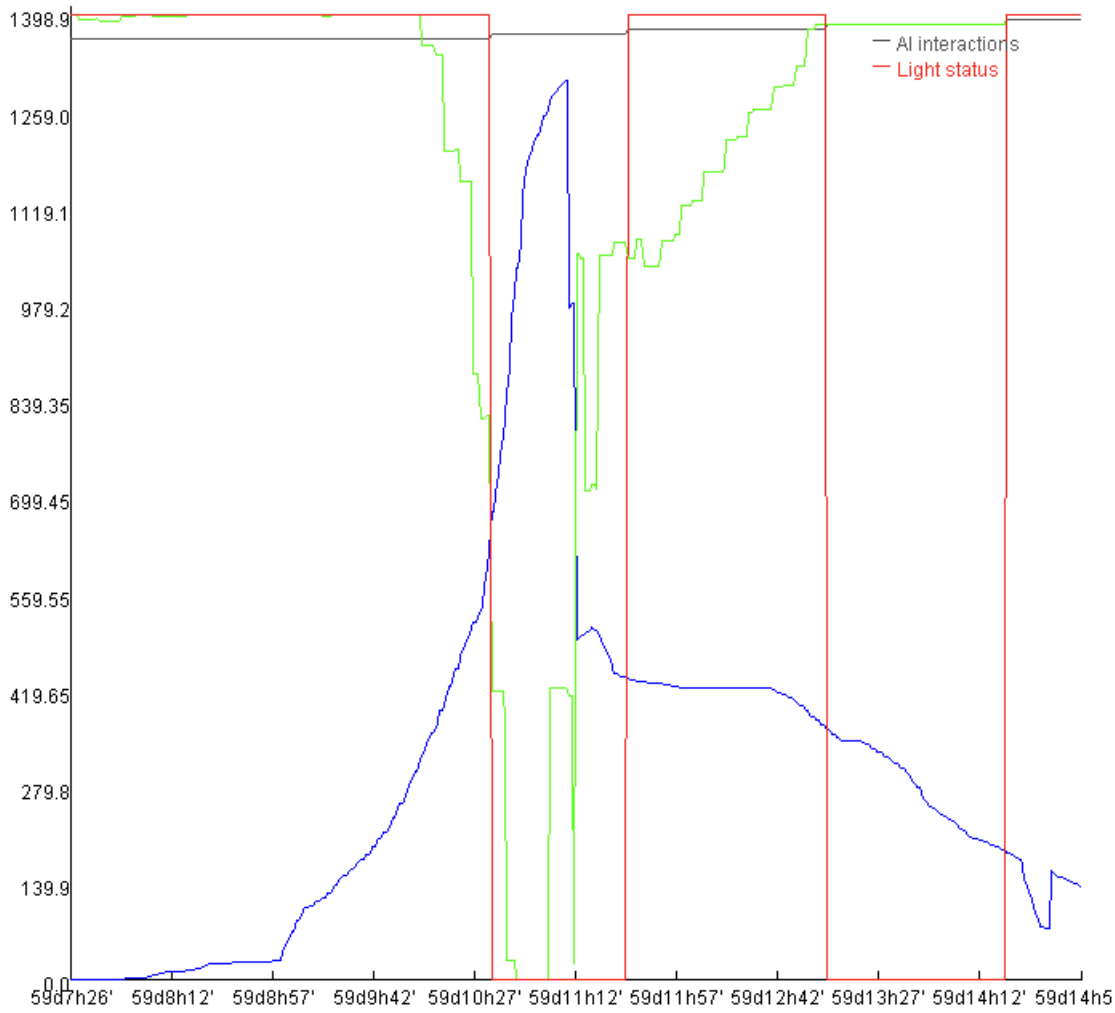


Figure 18.11: Also a quite good hypothesis for this day. The prediction is a bit more shaky but still fine and correct. We estimate the threshold to be around 650 lux which is still adequate for Helen. The second off switching is the cause of having lost the presence of this environment. Parameters: α=0.9, Ordinate: Interior daylight

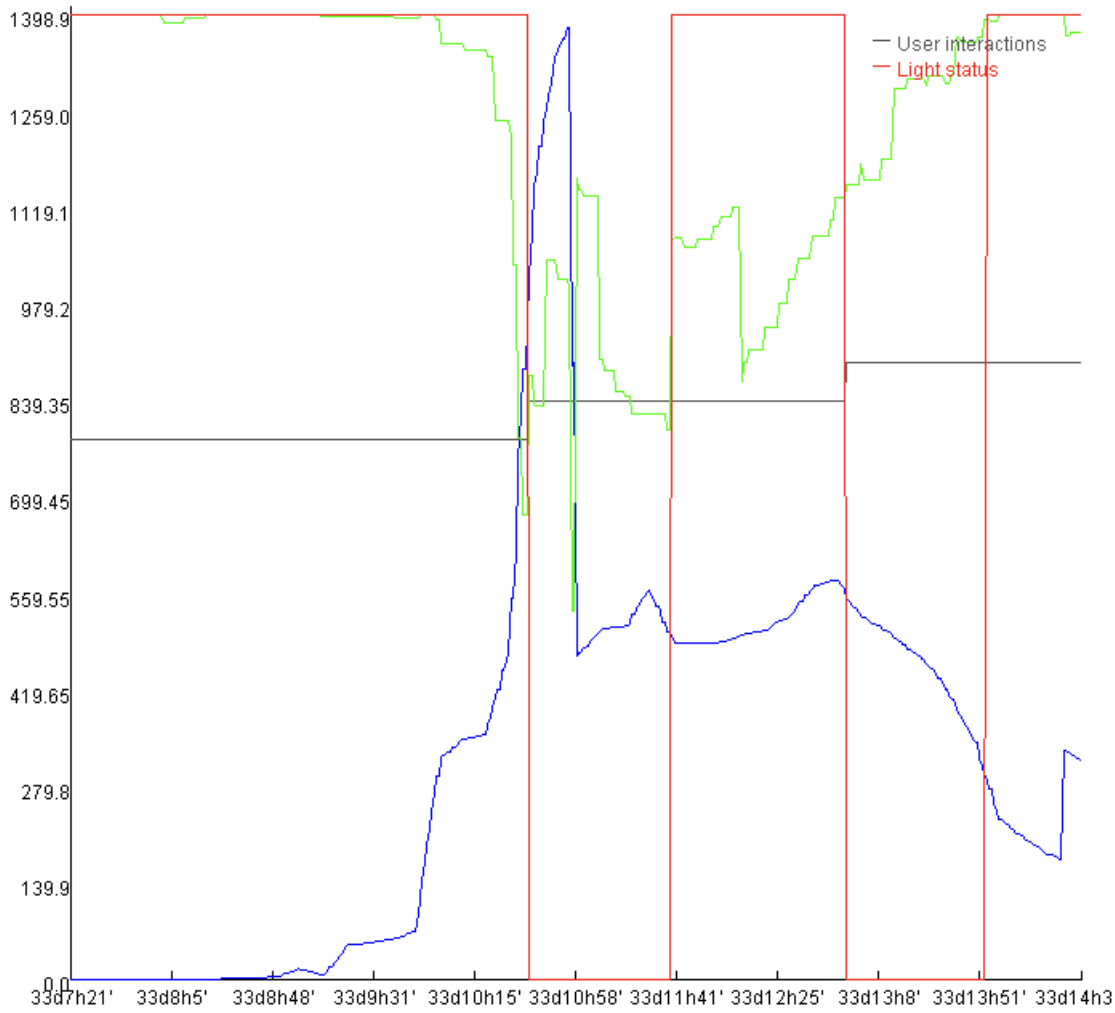


Figure 18.12: This graph depicts some interesting artifacts since it shows how new environmental changes needed to be learnt. We can convince ourselves by comparing the day with figure: 18.10. This day corresponds to the 33^thd day, which in contrast to the latter days, starts to get more clearer and thus causes the algorithm to need to get to know the new environment. Hence, no clear prediction could be made yet and consequently ended-up in user interactions (twice). The second on switching was wrong and thus resulted in dissatisfaction which started to grow until effect has been taken by turning the light back off. α =0.9, Ordinate: Interior daylight

As we presumed, both statistical predictions ended-up with no major big differences. Nevertheless we can conclude that if the user remains the same the momentum α doesn't impact the performance that strong. However if α is chosen too small, one or two user inputs which might have even been taken by mistake can impact the future prediction. Vice versa, if α is chosen too big, the algorithm rather tends to get lazy in its predictions and consequentially all environmental as well as user behavior changes are less taken into account. Instead of providing a static α we determine α heuristically by considering recent user inputs every once in a while. Hence upon receiving a couple of user inputs, α will sample out small. Likewise if no user inputs have been given recently, α will be increased. With this approach we rather take user behavior changes more strongly into consideration then by just providing a static defined α. Thus a commonly lazy acting statistical algorithm can be improved to react to sudden trend changes also to a certain degree.

We now illustrate the same test but this time with a dynamically determined α (See figure: 18.13).

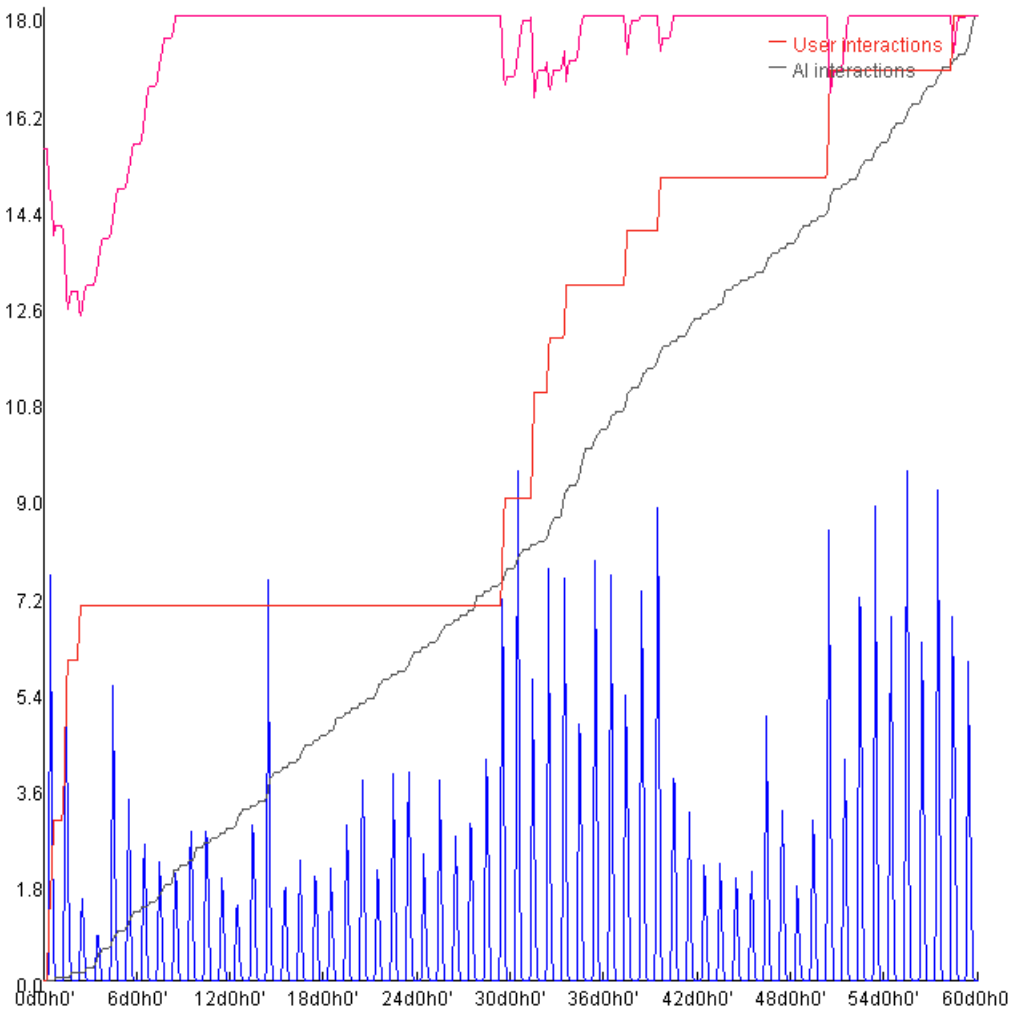


Figure 18.13: Evidentially, no major big difference can be noticed since the user remained the same across the entire period. Upon receiving many user inputs the alpha shrinks. When no recent user input can be registered, the α grows. The dynamic α is illustrated by the pink colored curve. Ordinate: User interactions

More informative was the result of the second test row where users are swapped by a 20 day shift (See figure: 18.14). Reacting to changing customs is still difficult although a dynamic α has been applied. Hereby minor

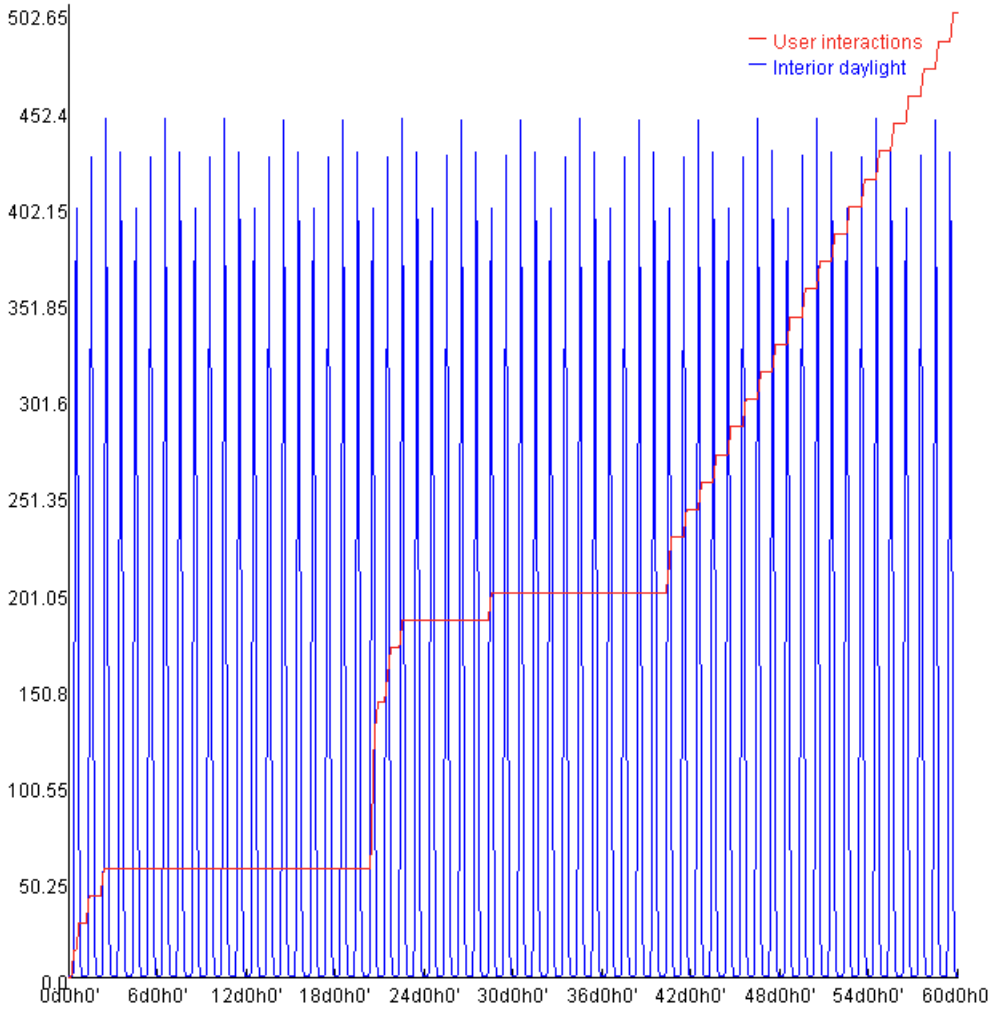


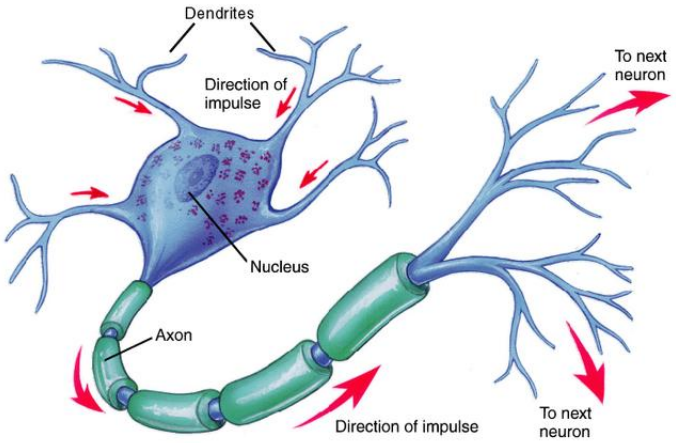
Figure 18.14: We notice that the transition to the dark user went quite well whereas the other transition caused major troubles. Ordinate: Interior daylight

transition changes from bright to dark seem to get passed easier, then the other way around. This is actually reasonable since the users have different characteristics and therefore also different dynamic thresholds.

We conclude that it takes time to get accustomed to strongly different user habits, due to the usage of timeslots. On the other hand, the timeslots could have been enlarged but this would have brought other problems along, such as an inadequate mapping of the transitions since the daylight can rise within a few minutes.
Obviously it is very difficult to fulfill or satisfy both tasks. But frankly speaking for some environments, this algorithm may score some good results. At least for a certain period. For transitions, other algorithms may be more appropriate and will according to the IBF algorithm (See section: 12.2) dominate the overall prediction eventually.

Chapter 19

Artificial Neural Networks



Another famous regression model which is quite often applied when it comes down to learning desired target functions are: *Artificial Neural networks (ANNs)*.

> Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples. Algorithms such as backpropagation use gradient descent to tune network parameters to best fit a training set of input-output pairs. ANN learning is robust to errors in the training data and has been successfully applied to problems such as interpreting visual scenes, speech recognition, and learning robot control strategies ([Mit97]).

It should be noted that we're not going to cover most of the ANN basics since this would go far beyond the scope of this thesis and should be the material of books such as: [Mit97], [Bis95], [UL03], etc.

Nevertheless it may be quite useful to provide a short summary of multi-layered artificial neural networks since they are from a computational point of view quite interesting.

What we're going to illustrate however is one of the most common algorithms used to train multi-layered artificial neural networks and further discuss some of the used functions and associated parameters that have been chosen in it as well as in what context neural networks are useful or applicable for us. In one of the latter chapters we've already mentioned artificial neural networks and their common problems. Remember that we've discussed that certain algorithms started to learn stuff we haven't really wanted them to? Tasks such as learning to switch off the window light upon switching on the corridor light or so? - To tell the truth those where artificial neural networks (ANNs).

19.1 Artificial Neural Networks

19.1.1 Overview

In this section we will only give a short introduction to the artificial neural networks and then illustrate one of the most common training algorithms called *backpropagation* used in multi-layered artificial neural networks. In subsequent sections some of the the equations are being explained and described on a rather mathematical basis. Further we cover some of the common problems that ANNs bring along.

Basically a neural network can almost approximate anything that could be represented by a vector of any dimension. Although it doesn't actually make any sense to use a neural network to approximate a simple function that could be approximated using the method described in the latter chapter (See chapter: 18) where the input and output is one-dimensional.

Since this algorithm plays a central role in the majority of training algorithms, especially for multi-layered networks such as a one that we've been using in ABI, we will dig in quite deep into its mathematical constitution (See section: 19.1.2).

The *backpropagation algorithm* is one of the most powerful and computationally efficient methods for finding the derivatives of an error function with respect to the weights and biases in the network. The key feature of this algorithm hereby underlies the strength of learning the **weights** for a *multi-layered network*. When

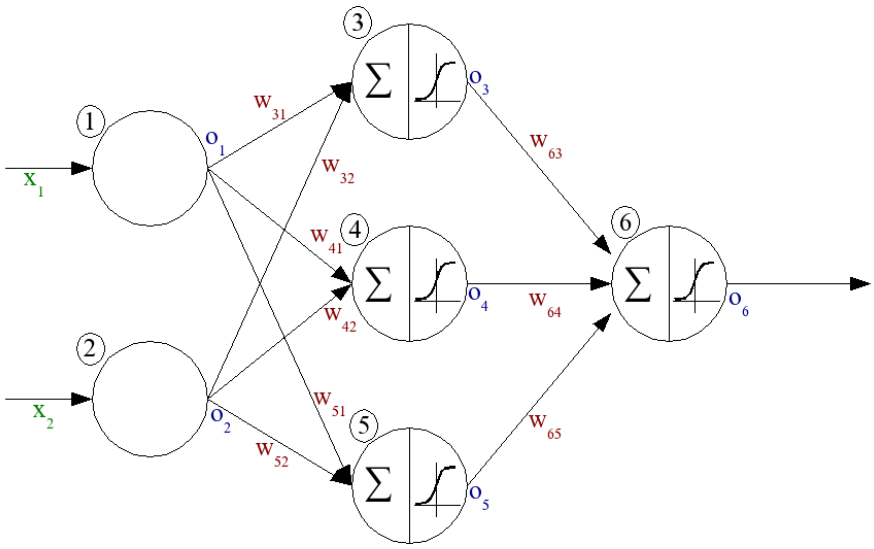


Figure 19.1: Sample ANN

training such a network (See figure: 19.1), the output error produced from a neuron j can be considered as a function of its weights from all incoming connections. Hence, the goal of the gradient descent is to find those weights, in trying to minimize the squared error between the network output values and the target values for these outputs.

The learning problem faced by the backpropagation algorithm is therefore to search through a large hypothesis space defined by all possible weight values, for all the units in the network (See figure: 19.2).

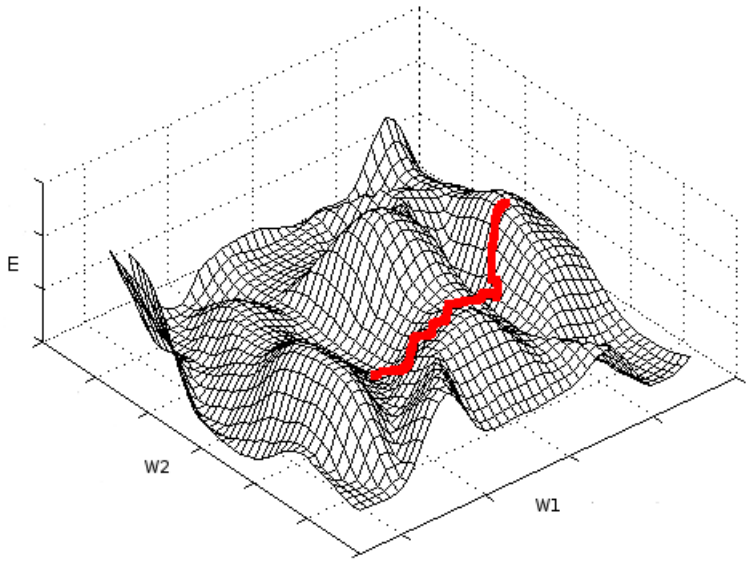


Figure 19.2: Error surface for two weights. Hereby the error is denoted by E.

Having once discussed the essential goal of the *backpropagation algorithm*, we then illustrate the entire algorithm completely since from a applied point of view more useful, then the plain theory part that is given in a subsection (See: 19.1.2)

We now consider the following stochastic gradient descent version of the backpropagation algorithm. It might be worth to mention that the true error gradient of E would actually be obtained by summing over all the $\delta_j x_{ji}$ values over **all** training examples before altering the weights. But commonly the stochastic is being used.

Stochastic Backpropagation(training examples, η, n_i, n_h, n_o)

Each training example is of the form $\langle \vec{x}, \vec{t} \rangle$ *where* $\vec{x}$ *is the input vector and* $\vec{t}$ *is the target vector.* η *is the learning rate (e.g. 0.05).* n_i*,* n_h *and* n_o *are the number of input, hidden and output nodes respectively. Input from unit* i *to unit* j *is denoted* x_{ji} *and its weight is denoted by* w_{ji}*.*

Create a multi-layered network with n_i *inputs,* n_h *hidden units and* n_o *output units.*

Initialize all network weights to small random number (e.g. between -0.05 and 0.05)

repeat

foreach $\langle \vec{x}, \vec{t} \rangle$ *in trainingexamples* **do**

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u.

$$o_u = \sigma(\sum_{i \in U} w_{ui} o_i) \tag{19.1}$$

Propagate the errors backward through the network:

2. For each output unit k, calculate its error term δ_k:

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{19.2}$$

3. For each hidden unit h, calculate its error term δ_h:

$$\delta_h \leftarrow o_h(1 - o_h) \cdot \sum_{k \in outputs} w_{kh} \delta_k \tag{19.3}$$

4. Update each network weight w_{ji}:

$$\Delta w_{ji} = \eta \delta_j x_{ji} \tag{19.4}$$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji} \tag{19.5}$$

end

until *Stop condition reached*

Algorithm 5: Stochastic Backpropagation

19.1.2 Details

Since we will be using quite some symbols we prefer to list them up:

E_d:	Error in the training example d
σ:	The sigmoid function
o_j:	The output by unit j
t_j:	The target output from unit j
w_{ji}:	The associated weight from the i^{th} input unit to j
outputs:	Units in the final layer (Outputlayer)
x_{ji}:	The i^{th} input to unit j
net_j:	corresponds to $\sum_i w_{ji} x_{ji}$
δ:	Error term
η:	Learnrate

In contrast to simple 2-layer networks, multi-layered networks learned by the backpropagation algorithm are capable of expressing a rich variety of complex non-linear decision surfaces. Hence in order to classify patterns in non-linear separable training sets requires separating surfaces more complex than hyperplanes. (See figure: 19.3)

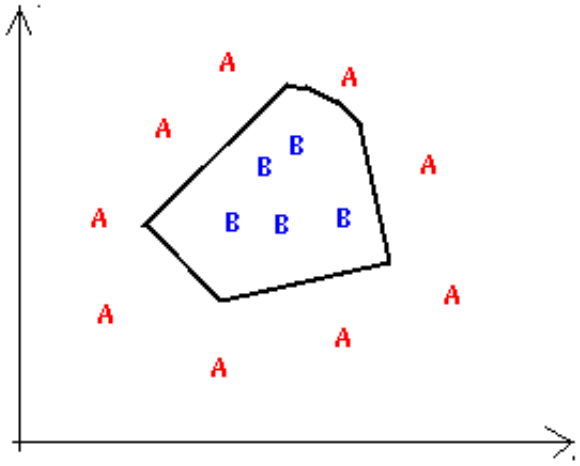


Figure 19.3: Complex, non-linear separable surface

In order to be capable to perform *backpropagation* in the first place each of the units within the network must provide a differentiable activation function. Commonly functions such as the *sigmoid (logistic)* or the *tan* function are being used since their're differentiable.
The sigmoid function $\sigma(y)$:

$$\sigma(y) = \frac{1}{1 + e^{-y}} \tag{19.6}$$

And its simple derivative:

$$\frac{\partial \sigma}{\partial y} = \sigma(y) \cdot (1 - \sigma(y)) \tag{19.7}$$

We will now explain how the gradient descent is calculated in a multi-layered network using the latter illustrated backpropagation algorithm. We think that it is quite important to get the essence on how the gradient descent really works especially on a mathematical basis. Most of the theory that we illustrate here have been inspired by two sources ([Mit97] and [Bis95]).

Since we need to iterate across all the training examples D, for each such training example d the gradient of the error E_d must be decreased. Hence according to equation: 19.4 and 19.5 which is calculated for every training example d, every weight w_{ji} from each unit u_j needs to be updated.

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} \tag{19.8}$$

E_d hereby corresponds to the squared error mentioned above and is traditionally calculated as:

$$E_d(\vec{w}) = \frac{1}{2} \sum_{k \in outpus} (t_k - o_k)^2 \tag{19.9}$$

We can extend $\frac{\partial E_d}{\partial w_{ji}}$ to:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji} \tag{19.10}$$

Since we need to distinguish whether we need to update output or hidden unit weights we illustrate both variants:

Output unit weight updates

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j} \tag{19.11}$$

$\frac{\partial E_d}{\partial o_j}$ correspond to the squared error function and can be derived to:

$$\frac{\partial E_d}{\partial o_j} = -(t_j - o_j) \tag{19.12}$$

$\frac{\partial o_j}{\partial net_j}$ hereby correspond to the activation function σ and hence
we can take the upper defined derivative of the sigmoid function (19.6)

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j} = o_j(1 - o_j) \tag{19.13}$$

Hence the error term illustrated back in the algorithm can be denoted by taking: 19.11 and substituting 19.12 and 19.13:

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)o_j(1 - o_j) \tag{19.14}$$

Hidden unit weight update For the hidden units we need to consider those units in the set whose immediate inputs include the output of unit j. The procedure to get the formula for the hidden units are generally quite similar calculated as for the output units:

$$\frac{\partial o_j}{\partial net_j} = \sum_{k \in outputs} \frac{\partial E_d}{\partial net_k} \cdot \frac{\partial net_k}{\partial net_j} \tag{19.15}$$

$$= \sum_{k \in outputs} \frac{\partial E_d}{\partial net_k} \cdot \frac{\partial net_k}{\partial net_j} \tag{19.16}$$

$$= \sum_{k \in outputs} \frac{\partial E_d}{\partial net_k} \cdot \frac{\partial net_k}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in outputs} \frac{\partial E_d}{\partial net_k} w_{kj} \cdot \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in outputs} \frac{\partial E_d}{\partial net_k} w_{kj} o_j(1 - o_j)$$

$$\frac{\partial E_d}{\partial net_j} = -o_j(1 - o_j) \sum_{k \in outputs} \delta_k w_{kj} \tag{19.17}$$

Considering equation: 19.17 and the latter calculated formula 19.13 and substituting each of them into equation: 19.10 we can write the final weight changing rules which now corresponds to the upper defined equations in the algorithm.

For output units:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta(t_j - o_j)o_j(1 - o_j)x_{ji} \tag{19.18}$$

For the hidden units:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta \left(o_j(1 - o_j) \sum_{k \in outputs} \delta_k w_{kj} \right) x_{ji} \tag{19.19}$$

19.1.2.1 Problems

The drawback when using a multi-layered backpropagation network is that the algorithm can easily get stuck in a local minima since it is not guaranteed to converge to the global minimum error. One way of suppressing this is by adding momentum to the weight update. To do this one simply add a fraction of the previous time-step's weight update to the current weight update. This will help the algorithm zip past any small fluctuations in the error surface (See figure: 19.2) and thereby giving a much better chance of finding the global minimum.

$$\Delta w_{ji} = \eta \delta_j x_{ji} + \alpha \Delta w_{ji}(n-1), \quad whereas\ 0 \leq \alpha < 1 \tag{19.20}$$

Another issue upon using backpropagation is again *overfitting*. Commonly there're several ways to counteract

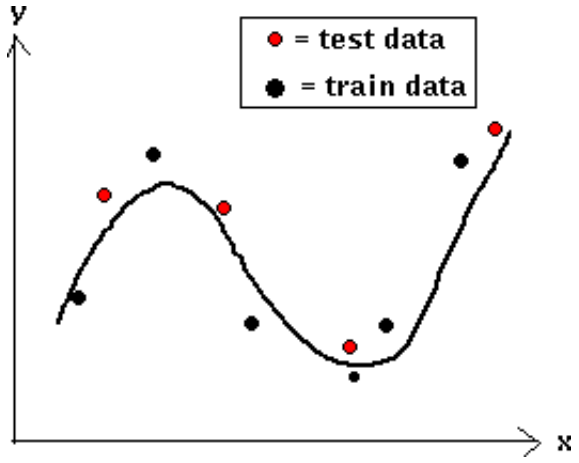


Figure 19.4: Good Fit, ([MUL])

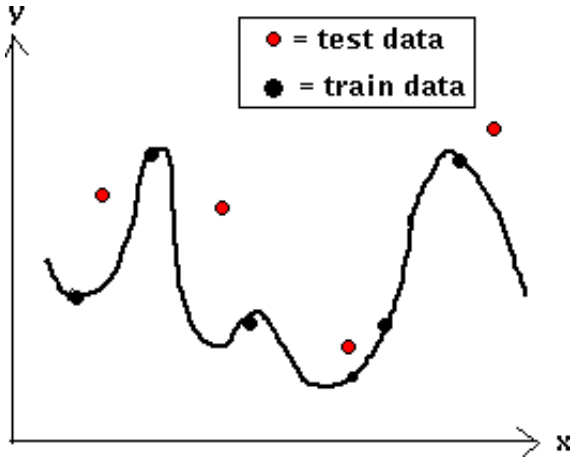


Figure 19.5: Overfit, ([MUL])

overfitting:

- *Minimizing the number of Neurons:* Reduce the number of hidden neurons to a minimum to improve the generalizability. Commonly determined by good trial and error.

- *Adding jitter:* Adding noisy data is a very important aspect for improving the generalizability.

- *Early stopping:* Applying two sets: Split the training data into a training and a validation set. The idea is then to stop the training when error from testing against the validation set begins to increase as opposed to reducing the squared error below a predefined value.

Since artificial neural network is a huge chapter in machine learning, we only tried to cover-up some of the topics which were either essential or just interesting for us to know. Especially for us since we've implemented the algorithm and hence needed to get acquainted, how certain things really worked e.g. testing the network under different conditions.

19.2 Realization

19.2.1 Network structure

We assume that a simple 2-layer network such as perceptrons, will not be sufficient since we think that more complex functions need to be approximated in our situation here. Hence we applied a 3-layer network that we trained using the backpropagation algorithm.

According to chapter: 17, we're dealing with quite a few input variables and thus it might be worth to lose some words about how the network structure is composed.
Depending on the usage we set a different number of hidden neurons. However in a standalone regression configuration (no combination with clustering or such) 4 hidden neurons have been used. Since our goal is to control a light only one output neuron is being used. The input neurons regularly corresponds to the number of all available inputs.

19.2.2 ANN discussion

Upon using an ANN pretty fast we need to address the issue of when to train the network. Because most ANNs are traditionally applied offline we need to adapt the ANN to work online. Hence, since we continuously receive data, we eventually need to decide when to evict old and when to incorporate new data. This is a very entitled questions since there is no generic solution to this.

In order to launch the discussion we will initiate some questions that eventually need to be answered.

1. *How large should the training-set be?*
2. *When and how to train the network?*
3. *Can we affect the current prediction by considering LTM data?*

A training-set corresponds to a set of vectors. Each of which contains some sort of *"snapshot"* of the environment (momentary sensory inputs as well as effector outputs).
Evidentially a lot of such questions must first be answered to finally implement a solution to those problems.

There are actually many ways to configure and to train a network that might suite our purpose. The first problem can, at least in some extent be counteracted. Namely the problem of the training-set size. According to [Mit97] the problem of overfitting is quite small when the training-set is large enough or else, we would need to think of applying a k-fold cross-validation approach ([Mit97]). However the problem when using small training-sets still remains since the the collected data might be full of noise. A little calculation might prove that: Let us considering an framework frequency of $600^{-1}Hz$ (10 minutes). Hence, this would result in 6 samples within one hour to be collected. Spread over lets say a training-set size of 30 would be completed within 5 hours.
At the latest in the morning we will realize that we would only be aware of the late yesterday afternoon or in the worst case even the night e.g. when someone has been working late.
Specifically speaking we would start to forget stuff that we had been learning 5 hours ago! Evidentially we need a training-set size that is much larger then 30.
To conclude, when the knowledge space is to small we can hardly predict changing behaviors. A further improvement can be achieved by keeping the old weights from the previously trained networks and rather perform a continuous re-training procedure (online) with the adapted weights with a lowered epoch size.

Hence, if the training-set size is large enough and additional help is provided by applying an online training that would even lessen network computation, we might achieve better results. Herewith old knowledge will still have an impact in future predictions by incorporating a couple of old input data into the new knowledge base e.g. by taking a ring buffer that basically thrust aside old knowledge.
The flip side by applying an online approach is that the network will be reacting lazier to sudden changes.

Especially we might face this problem when a rainy week is about to end. Accordingly the network might have been trained with corresponding values. Especially on fogy, rainy days occupants might interact differently with their environment. Since the weather can always change pretty fast in nature and within the sensor outputs, we might really need a faster way to suit the new needs since the thresholds have surely been shifted and above all, the network doesn't have the slightest idea what good weather conditions might bring along. Conditions such as blinds that are all the sudden moved down to block direct sun light which consequently start to turn into interior daylight spikes which maybe quite hard to be (re-)learned. Not to mention that by just collecting a bunch of data and start to perform a prediction wouldn't be a practical solution either since we need to consider the fact that in any point of time users may change and hence also the knowledge base.

Finally we need to make up our minds about the optimal storage length. Since at one side we need to deal with changing behaviors and on the other side we need to be agile in re-learning things. Hence its a kind of a Catch-22 that we find us in.

Evidentially by just applying a regular artificial neural network, we probably won't achieve sufficient good results in reality since our objectives is seeking for a solution that provides a fast reliable re-prediction to changing situations and behaviors.

Another solution that we've been testing is a method of trying to incorporate some LTM and STM behavior on the basis of neural networks (See subsection: 19.2.3).

19.2.3 LTM-STM Network

With this approach we try to comply to the statement made above.
The intention was based on the fact that LTM in human brains have a strong influence on the perception through top-down processing. Our prior knowledge affects how we perceive sensory information. Our expectations regarding a particular sensory experience influence how we interpret it. This is how we develop bias. For instance: Most optical illusions take advantage of this fact ([LTM]).

Hence we derived following three different network structures out of that fact, which generally can be seen as simple dual networks (See figure: 19.6, 19.7 and 19.8):

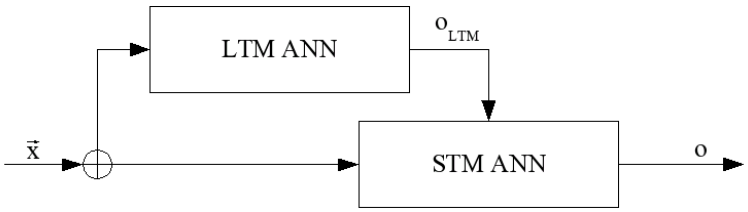


Figure 19.6: LTM-STM Network decomposition with no user input control and nor does the LTM affects the final prediction.

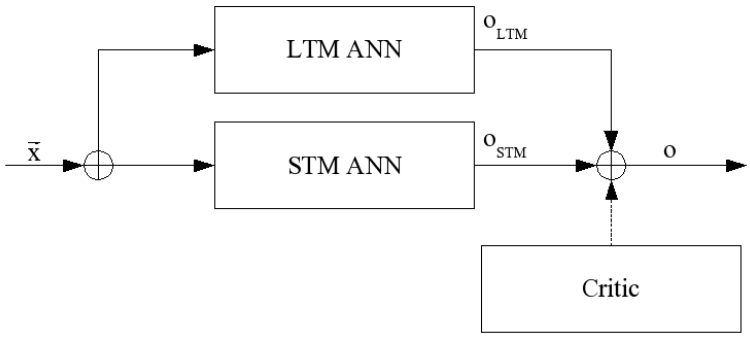


Figure 19.7: LTM-STM Network decomposition with a user controlled LTM consideration.

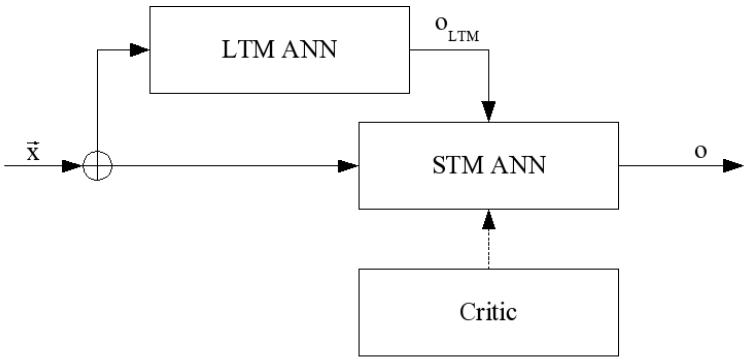


Figure 19.8: LTM-STM Network decomposition with a user controlled STM size.

As depicted all figures an additional input is provided from the LTM network in order to consider long term knowledge (LTM) influences.
Figure: 19.6 illustrates an other variant of this approach where in contrast to the latter one the overall prediction is calculated using a function of user inputs.
By taking user inputs also into account we hope for a more accurate environment function to be approximated. LTM hereby corresponds to a much larger set of training data whereas the STM contains $\frac{1}{10}^{th}$ of

the LTM size. Hence upon receiving many user inputs the STM should govern the prediction. Vice versa, if user inputs are sparse the LTM will dominate. The prediction is then calculated using:

$$w_{ltm} = 1 - \frac{w_{ltm_{max}} - w_{ltm_{min}}}{1 + UserInputTime/ltm^{c}_{ThresholdTime}} + w_{ltm_{min}} o = o_{ltm} \cdot w_{ltm} + (1 - w_{ltm}) \cdot o_{stm} \quad (19.21)$$

Where w_{ltm} correspond to the weight that determines how strong the LTM should be considered. The w_{ltm} is calculated using the threshold curve (See figure: 19.9). The used symbols:

$w_{ltm_{max}}$:	Maximal influence of LTM
$w_{ltm_{min}}$:	Minimal influence of LTM
$UserInputTime$:	Exceeded time since last user interaction
$ltm_{ThresholdTime}$:	Critical threshold value
c:	The slope of the reaction
o:	Overall prediction

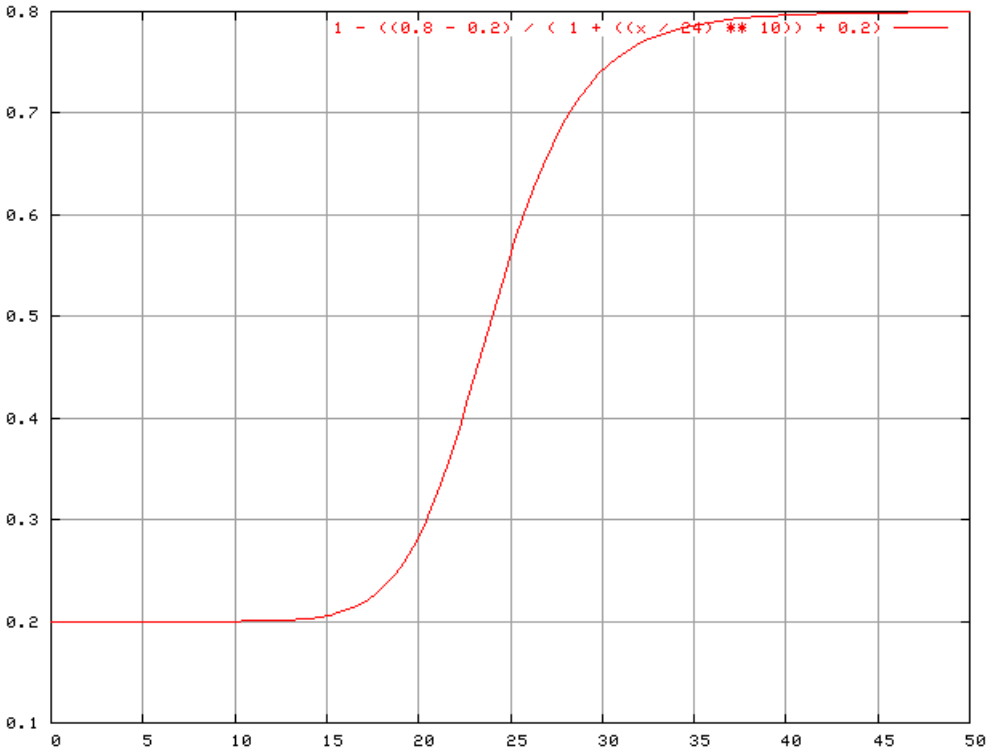


Figure 19.9: Threshold-curve

This approach is actually one of the algorithms that we had been testing for two weeks in real environments here at the INI (See chapter: 25).

The other LTM-STM network (See figure: 19.6) doesn't consider the LTM prediction in the final prediction and instead returns the STM prediction directly without considering user interactions at all.

A third approach that we came up with later on, was a solution that is some sort of mixture between the other two solutions. A mixture since the network structure remained the same as depicted in figure: 19.8, but with the difference that the LTM knowledge is also not directly considered in the final prediction. Instead, we dynamically adapt the size of the STM as a function of the user input.

There would be a lot of other ways to experiment around with STM-LTM but we had to stop to make room for other approaches.

For all solutions we can construct an algorithm that accommodates the basic STM-LTM behavior. The algorithm is simple and straight forward and thus won't need any further explanation.

Result: trained LTM- and STM ANN
input: A set of continuous input-pairs $\Psi\{\vec{x}, y\}, y \in [0;1]$
repeat
  fetch next input $in_t \leftarrow \{\vec{x}_t, y_t\}$
  initialize traininqueues
  if *somebody present* **then**
    put $in_t \rightarrow traininqueue_{STM}$
    put $in_t \rightarrow traininqueue_{LTM}$
    if $trainingueue_{STM}$ *is full* **then**
      train ANN_{LTM} with all available tuple in $traininqueue_{LTM}$
      create empty list $learndata_{STM}$
      foreach $\xi \in traininqueue_{STM}$ **do**
        $o_{LTM} \leftarrow result\,(ANN_{LTM}, x_\xi)$
        create in_{STM} from x_ξ with additional o_{LTM}
        put $\{in_{STM}, y_\xi\} \rightarrow learndata_{STM}$
      end
      train ANN_{STM} with all tuple in $learndata_{STM}$
    end
  end
  $t \leftarrow t + 1$
until *Stop condition reached*

Line 26 shows how the 2^{nd} solution would be incorporated and 27 shows the momentary solution according to figure: 19.7. Also note that the network training happens to be online also, as discussed in the latter section.

```
1  public synchronized double getPrediction()
2  {
3      IDataInput[] inputs = this.deviceagent.getInputs();
4      double scaledInputs[] = new double[inputs.length];
5
6      for (int i = 0; i < inputs.length; i++)
7      {
8          scaledInputs[i] = inputs[i].getScaledValue();
9      }
10
11     if (stmAnn != null)
12     {
13         double ltmAnnOutput = 0;
14         if (ltmAnn != null)
15         {
16             ltmAnnOutput = ltmAnn.use(scaledInputs);
17         }
18
19         double[] tmp = new double[scaledInputs.length + 1];
20
21         System.arraycopy(scaledInputs, 0, tmp, 0, scaledInputs.length);
22         tmp[scaledInputs.length] = ltmAnnOutput;
23
24         double stmAnnOutput = this.stmAnn.use(tmp);
25
26         // return ltmAnnOutput * ltmWeight + (1 - ltmWeight) * stmAnnOutput;
27         return stmAnnOutput;
```

```
28    }
29    else
30    {
31        return -1;
32    }
33 }
```

19.3 Evaluation and Discussion

We've been testing both solutions. A plain neural network and LTM-STM solutions. As mentioned in section: 17, the results should be treated with care and should only demonstrate the correctness of the implementation as well as the general learning process. Thus the tests are only applicable for us to correct possible severe slips or even to prematurely take out entire algorithms due to poor performance.

Regular Artificial Neural Network test:

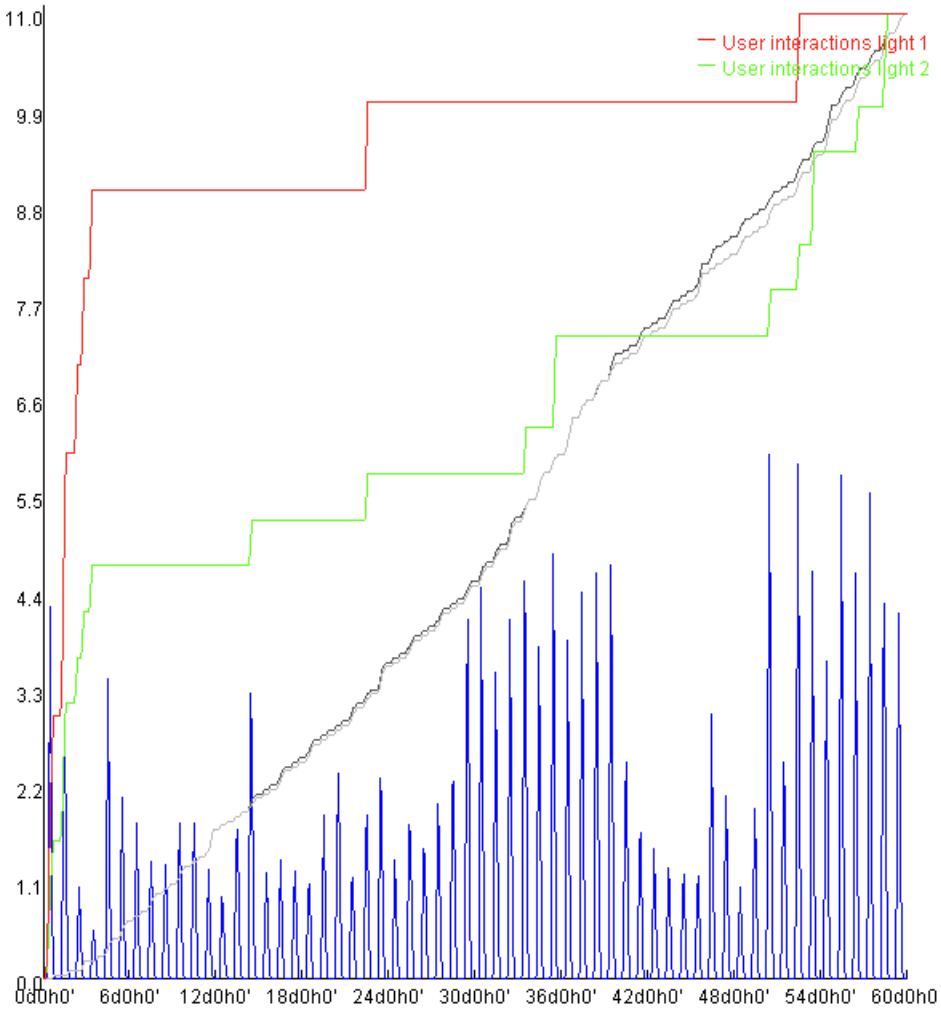


Figure 19.10: The backprop performed very well. Very clear actions have been conducted also. Even the second huge daylight spike has successfully been mastered by one of the light predictors, where normally other algorithms would classically fail to state a correct prediction. Ordinate: User interactions

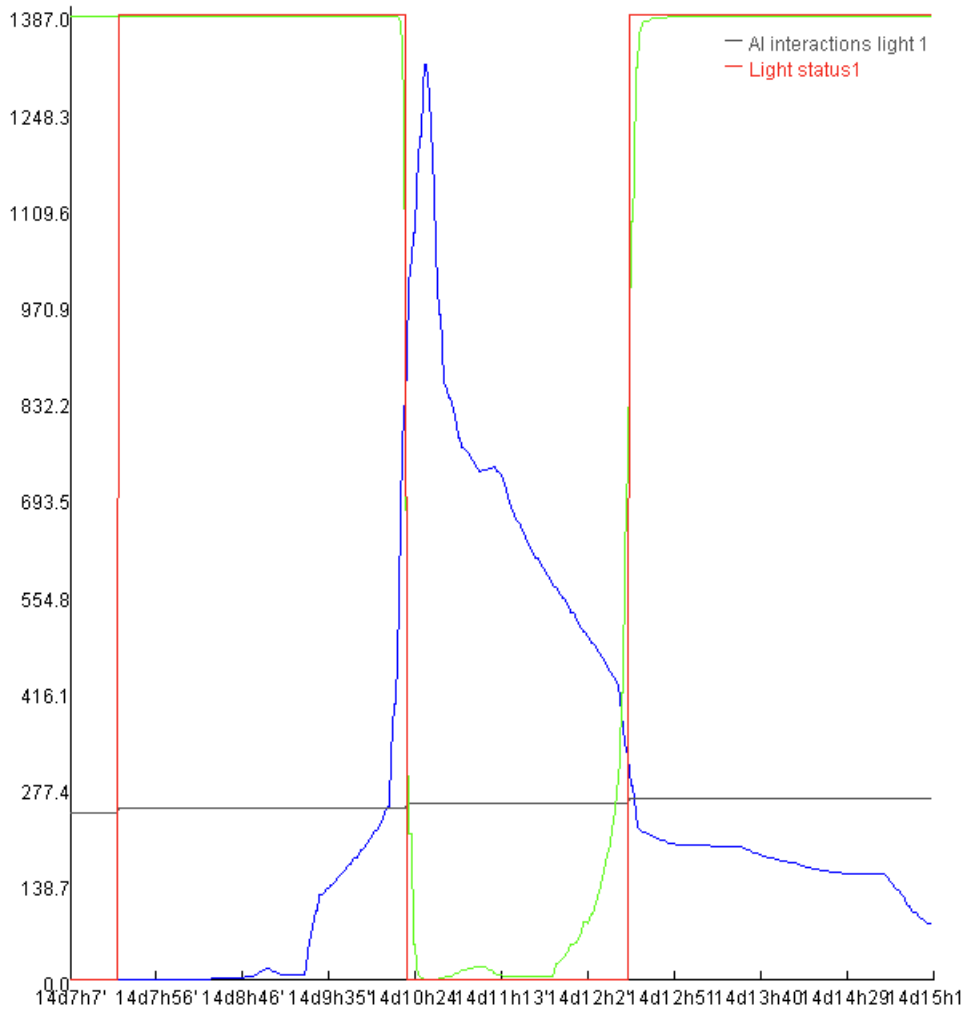


Figure 19.11: We now analyze the overcome barrier (2^{nd} huge daylight spike) a little closer. As usual the green curve illustrates the prediction and blue curve denotes the interior daylight. Although the zoom is a little far we still recognize that the fast rising peak, that within a few minutes had occurred, has correctly been interpreted by the backprop. Well it could have hit a little earlier but the result is impressively accurate for the very beginning and not to mention by such a steep slope. Ordinate: Interior daylight

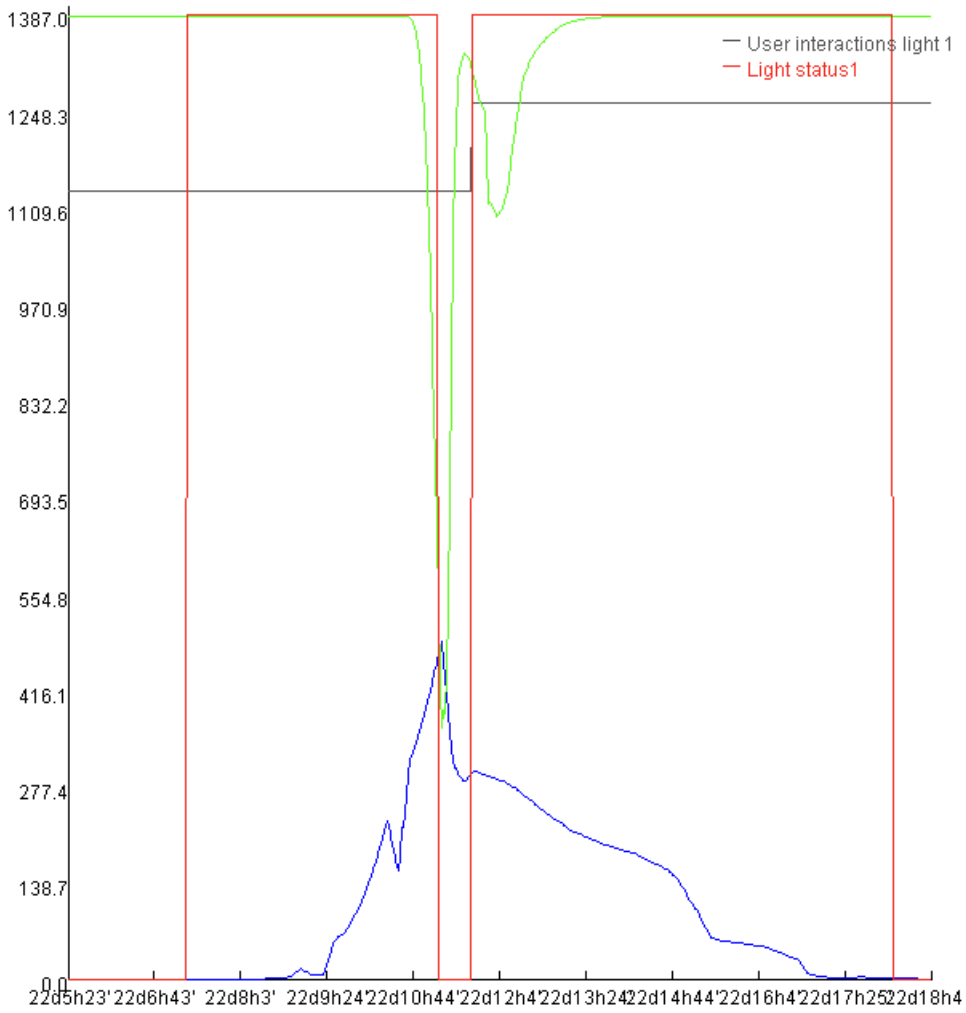


Figure 19.12: On top of the previous results, we even notice that one of the two user interactions happens to be unforced (according to section: 14.6) since basically any control of the environment is held up a certain amount of time (currently 1 hour) in order to avoid possible opposing system interactions. Hence, the second prediction would have been correctly made by the backprop but it wasn't allowed to conduct the action due to the delay. Ordinate: Interior daylight

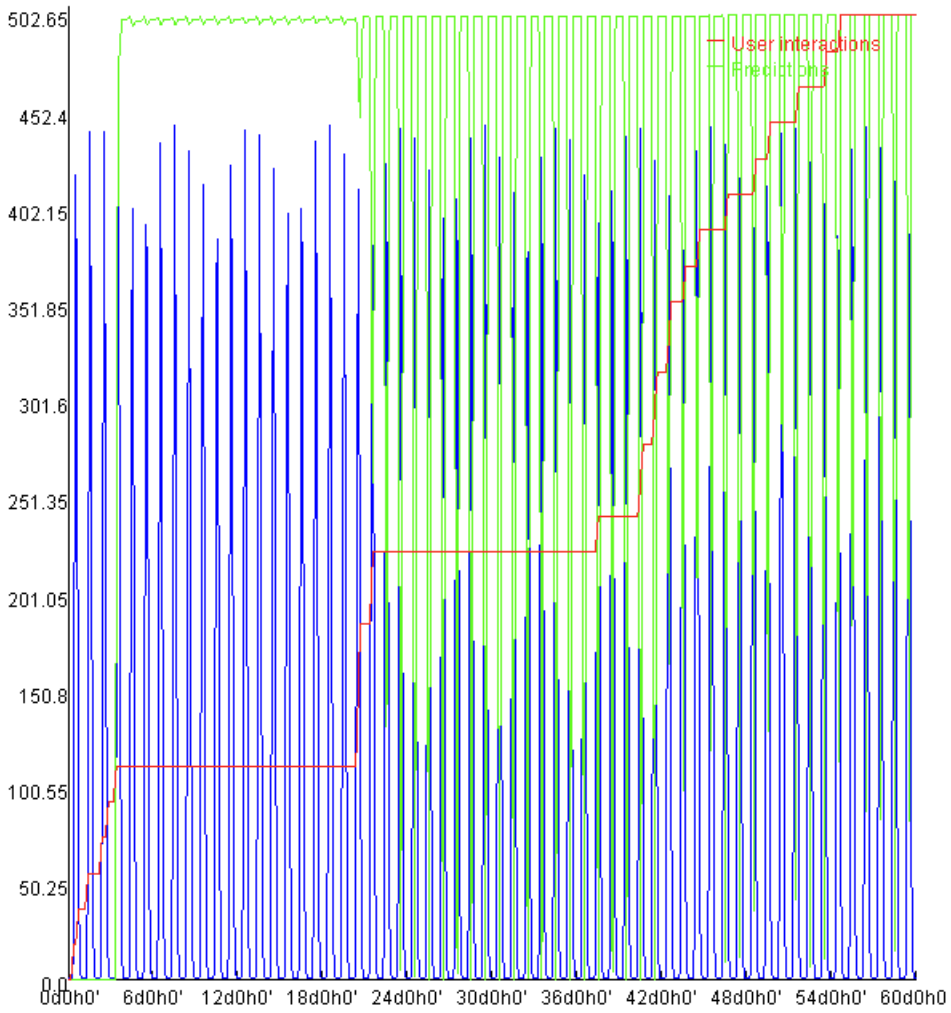


Figure 19.13: Getting to known changing customs. Ordinate: Interior daylight

Things for the most of the STM-LTM networks didn't turn out to be very successful. At least judged by their performance in the test-bed. Actually not that what we've been expecting.
The second solution (See figure: 19.7) we already started to test in real environments since at that time, we haven't really thought on implementing all three solutions. Additionally since real life tests are very important results we had to push it a little.
Furthermore the second solution sounded reasonable for us and did a quite a good job in one of our first test-sets. However a couple of days later we realized that the test-set was too easy. The reason why we haven't found out about that fact was because it was one of the first algorithms and hence we didn't have any other reference results to compare to.

The first approach, according to fig: 19.6 was actually performing not that bad in the introduced test-set (See section: 17.1.2).

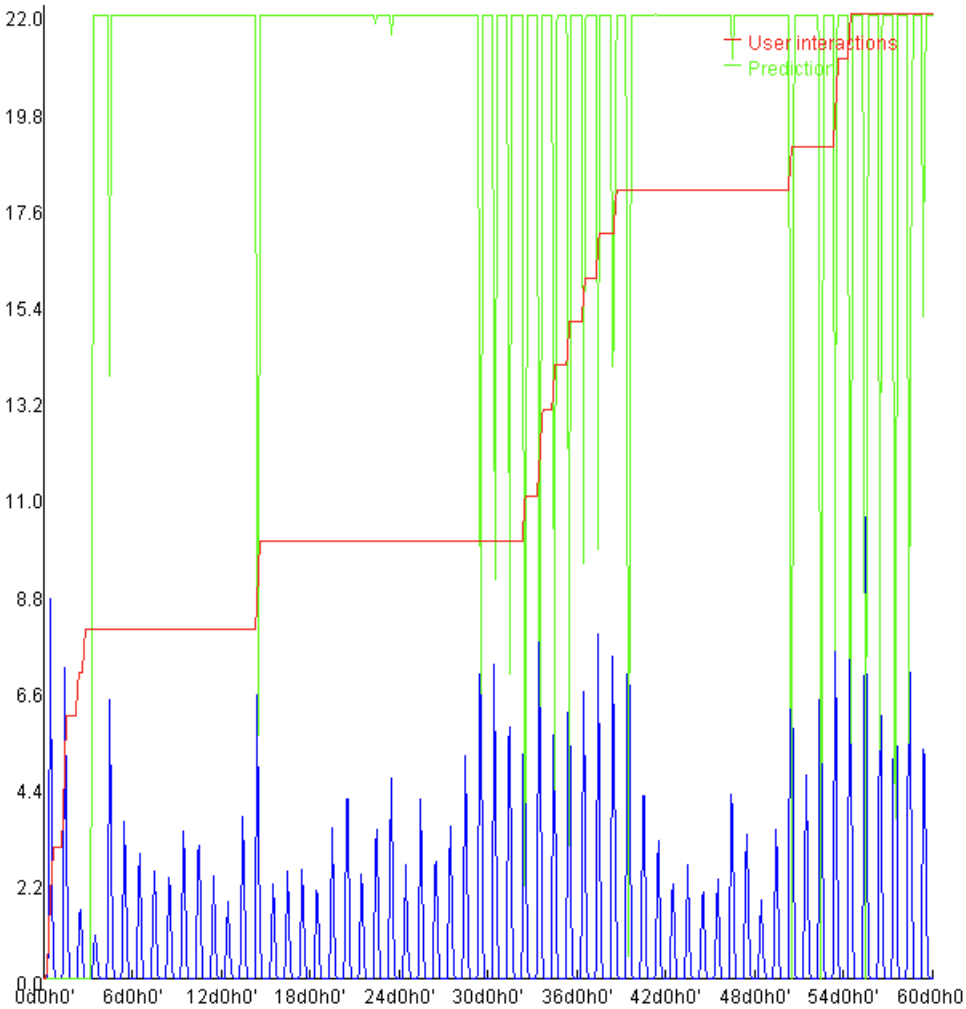


Figure 19.14: Very smooth predictions have been taken due to enhanced accuracy. STM size: 100, LTM size: 800, Ordinate: User interactions

One of the advantage we hope to gain by using an STM-LTM network structure is

- Achieving high prediction accuracy
- Enhance transition predictions by changing customs

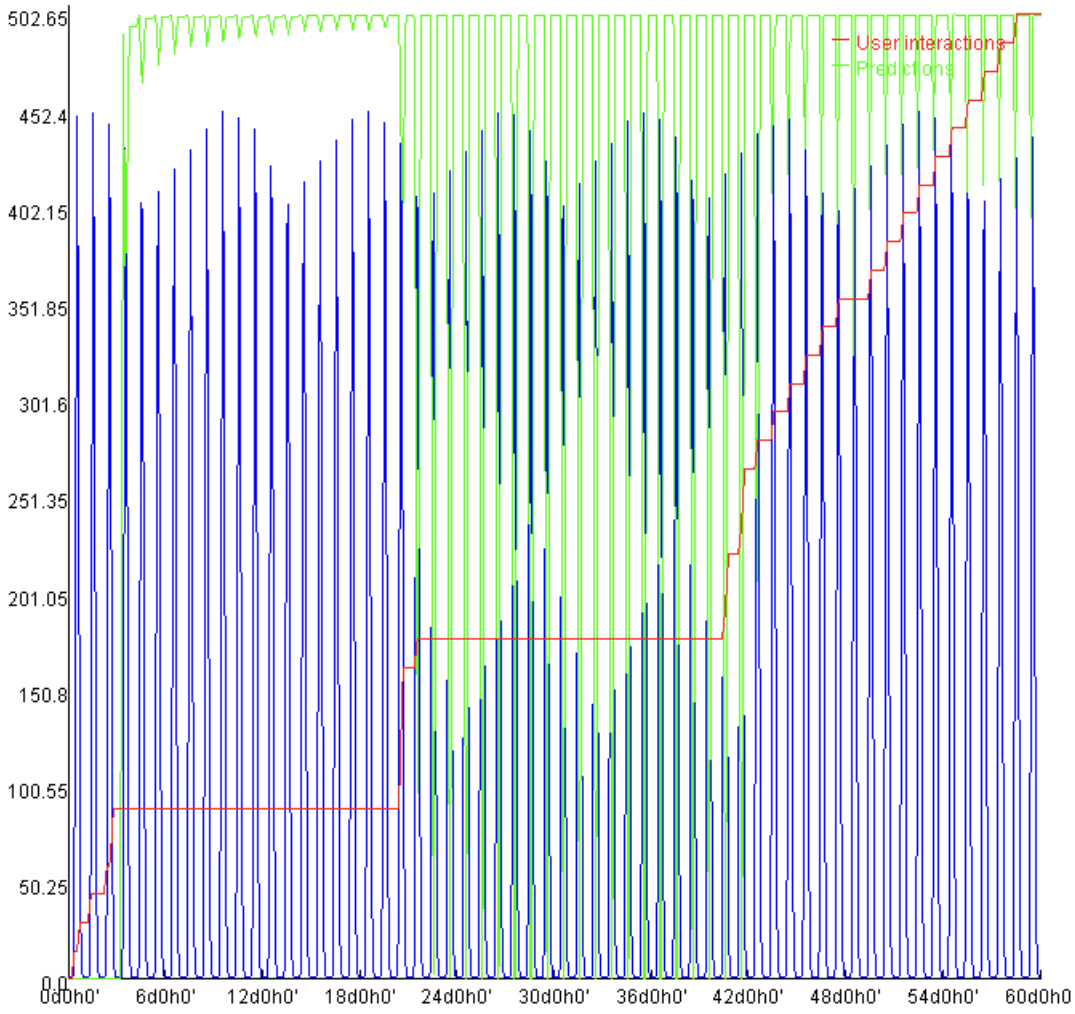


Figure 19.15: At first sight, the predictions look quite good. However the increasing user inputs are getting kinda contemplative at the end. When zooming in though (See figure: 19.16), we see why the results of a test-bed should be trusted with care and not only be measured by the number of user inputs. STM size: 100, LTM size: 800, Ordinate: Interior daylight

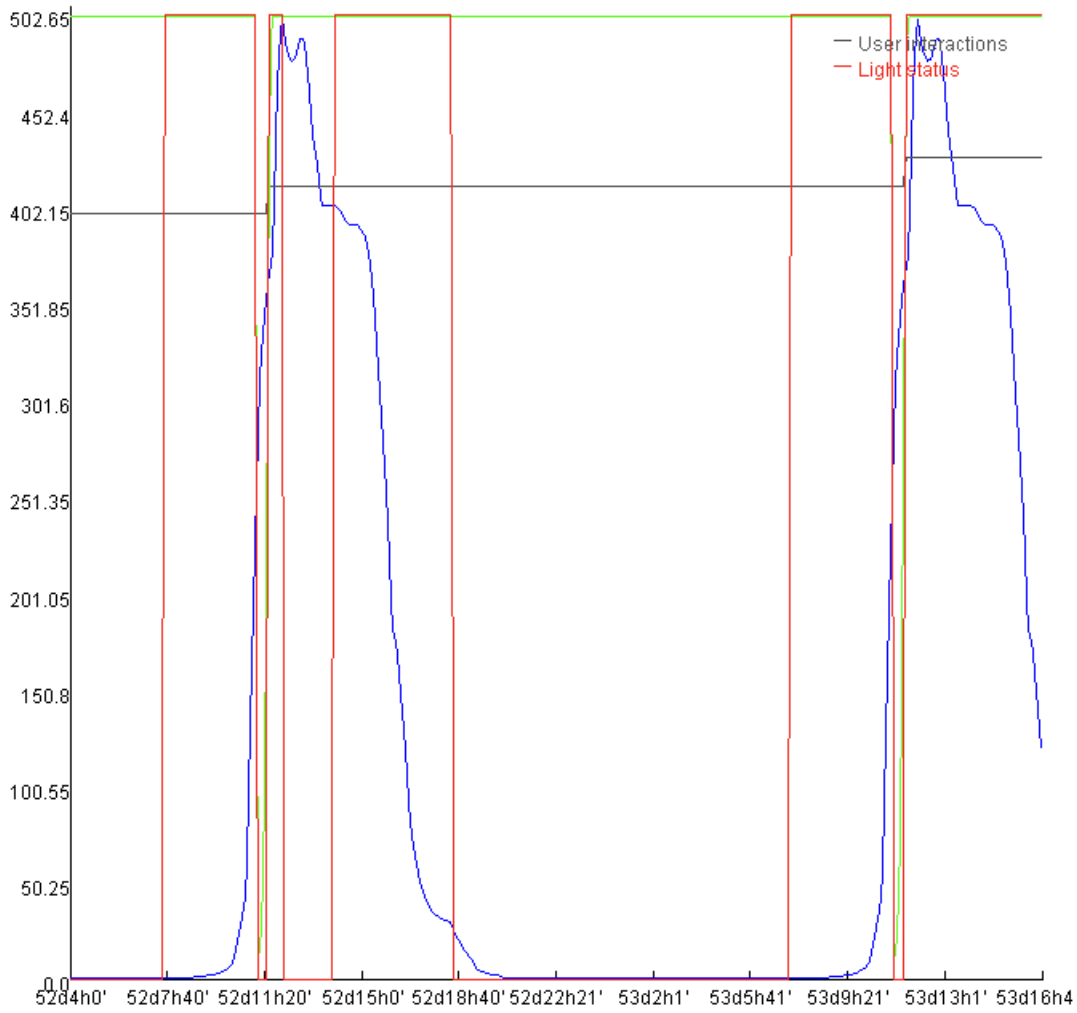


Figure 19.16: We notice that practically all user inputs were unforced. The reason is because each day remains equal. The same daylight, temperature and even humidity. Hence the network will get overfitted since no noise has been provided. Neither from the synthetic sensory inputs nor from **Helen**. Hence, in this situation, the network has learnt to switch on and off the lights always under the same condition. Hereby the security delay doesn't really even matter, although the backprop would have switched the light right back always. The major drawback of this simulator (or at least this test) can herewith proven to be inadequate for a neural network since all it will do is to start to memorize distinct slopes since every day remained the same. We'll be discussing this issue in the conclusion chapter a bit more.
As usual, when illustrating zoomed graphs, the red curve corresponds to the light, the blue to the interior daylight, the green to the prediction and the gray to the accumulated user input count. Parameters: STM size: 100, LTM size: 800, Ordinate: Interior daylight

According to those hypothesis we conclude that it is worth to test all three STM-LTM algorithms in a real environment since we don't known how occupants will be interacting with their environment.

Chapter 20

Self Organizing Maps (SOM)

Another solution that we've been looking at and testing, were Kohonen Networks. But to tell the truth right beforehand didn't come off well.
Nevertheless some of the concepts provided by self organizing maps are quite interesting and frankly gave us the initial attention to dig into clustering a bit more since it owns certain features that might improve other algorithms remarkably (See chapter:21 and 22).

However self organizing maps have many drawbacks in our context and hence we would like to discuss them in a short summary since the results gained from this chapter might serve as a good basis for the next chapters.
We first give a first introduction in SOMs since many different versions of them coexist. Additionally we outline certain problems and why pure SOMs do not quite provide a suitable solution to our problem.

20.1 SOM

20.1.1 Overview

One way to look at our environment is a sort of topographic map where different sensory configuration maybe stored. Just like human brains we might be able to classify our environmental data without actually need to provide prior knowledge on what we might really going to cluster and hence learn with it.
In other words: The constructed clusters might give us some insights in the structure and the relation of environmental data.
A Kohonen map ([SOM]) is a neural network comprised of an input layer and an output layer. Unlike the backpropagation, which is a supervised learning algorithm, kohonen maps or also known as feature maps, perform unsupervised learning.
Basically a SOM can be visualized as a high-dimensional data sets on a 2-dimensional regular grid of neurons (units) which would actually be fully-mashed among each other but since this is hard to illustrate we omitted to show the connections (See figure: 20.1). Each unit u is hereby represented by a n-dimensional weight vector $\vec{w}$ where n is equal to the number of attributes in the training data.

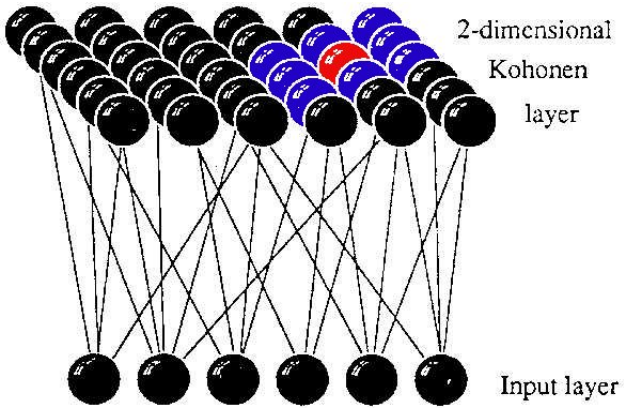


Figure 20.1: Kohonen Network structure

We're going to illustrate how such networks are being trained since its quite a simple algorithm and then rather discuss how we realized the SOM.
Generally the algorithm works like this: Each time an input vector $\vec{x}$ is presented to the network, its euclidean distance to each unit in the output layer is computed. The output with the smallest distance to the input vector is then declared as the winner. The winning unit and a set of units in the neighborhood weights are then adjusted by moving the weights toward the input vector. The basic idea is that not only the maximal excitatory winner unit w_k should learn but rather also the units within the neighborhood. This procedure is also known as *blockparty* and addresses now that some kind of radius must be considered that determines how far we need to consider our neighborhood units. Hereby a so called *neighborhood function* defines how strong the units within the map need to be altered. Several different *neighborhood function* exist which we refer to as $\varphi(dist(u_{winner}, u_j), r)$. Hereby u_{winner} and u_j correspond to two units in the output layer and r denotes the block-radius:

Common *Neighborhood functions*:

$$\textbf{\textit{Gaussian distribution function:}}\ \varphi(dist(u_{winner}, u_j), r) = e^{\frac{(\|winner - j\|)^2}{2r^2}} \tag{20.1}$$

$$\textbf{\textit{Cone function:}}\ \varphi(dist(u_{winner}, u_j), r) = \begin{cases} 1 - \frac{(\|winner - j\|)}{r} & if\ \|winner - j\| < r \\ 0 & else \end{cases} \tag{20.2}$$

Initially the neighborhood radius and the learning rate are quite large and therefore many units are moved around the input space to match the set of training vectors. Gradually though, the neighborhood shrinks and the learning rate will be decreased. Therefore its name, self organizing map since it self organizes over time. We now present our slightly altered version of the SOM algorithm in its short form:

20.1.2 Problems

The major drawbacks of SOMs as well as other clustering algorithms such as the k-means (See section: 21.1.1), is that they either require to specify the number of clusters in advance, or leave the decision of the number of clusters to the user (hierarchical clustering). To address such shortcomings of fixing a priori the number of clusters, other advanced algorithms must be considered (See chapter: 21 or 22). Although initial attempts have been made to counteract this issue by so called Adaptive Self-Organizing Maps ([YWN05]) which adaptively decides the best architecture for the self-organizing map. This is accomplished by some sort of a growing grid that basically gathers statistical information about its clusters and at each adaptation step, decides where in the map a new row or column needs to be inserted.

SOM learning

Initialize all weights (all vectors $\vec{w_{u_i}}$) to small random number (e.g. between -0.05 and 0.05) as well as the initial learning rate (α_{init}) and radius (r_{init})

repeat

Randomly select a new input vector $\vec{x_{in}}$

Calculate a new current learn rate α and new radius r

$$\alpha = \alpha_{init}\left(\frac{\alpha_{final}}{\alpha_{init}}\right)^{\frac{epoch}{epoch_{max}}}$$

$$r = r_{init}\left(\frac{r_{min}}{r_{init}}\right)^{\frac{epoch}{epoch_{max}}}$$

Determine the winner unit u_{winner} for $\vec{x_{in}}$

foreach *output neuron u_i in the map* **do**

$fact \leftarrow \varphi(dist(u_{winner}, u_i), r)$

$\vec{w_{u_i}} \leftarrow \vec{w_{u_i}} + \alpha \cdot fact \cdot (\vec{x_{in}} - \vec{w_{u_i}})$

end

$epoch \leftarrow epoch + 1$

until *Stop condition reached*

Algorithm 6: SOM

$\vec{x_{in}}$:	input vector
α:	epoch dependent learnrate $\alpha = \alpha_{init}\left(\frac{\alpha_{final}}{\alpha_{init}}\right)^{\frac{epoch}{epoch_{max}}}$
α_{init}:	initial learnrate
α_{final}:	final learnrate
r:	radius in which the learning should take effect
r_{min}:	minimal radius
r_{init}:	initial radius (we used the square of the neuron count ($\sqrt{\lvert U\rvert}$))
$epoch$:	current epoch
$epoch_{max}$:	maximal epoch count
u_i:	specific unit (also called neuron) $u_i \in U$ with a net position ($u_i = (x_{u_i}, y_{u_i})$) and a vector for its room position ($\vec{w_{u_i}}$)
U:	A set containing all units $U = \{u_1, u_2, \ldots, u_n\}$
u_{winner}:	winner unit ($u_{winner} = \arg\min_{u_i \in U} \langle \vec{x_{in}}, \vec{w_{u_i}} \rangle$)
$\vec{w_{u_i}}$:	vector which represents the room position for u_i
$dist(u_i, u_j)$:	distance of the net positions ($dist : (u_i, u_j) \mapsto \sqrt{\left(x_{u_i} - x_{u_j}\right)^2 + \left(y_{u_i} - y_{u_j}\right)^2}$)
φ:	the so called neighborhood function which calculates the training factor part which depends on $dist(u_i, u_{winner})$ and r (see text for details)

Other common problems that need to be addressed are:

- How to evaluate the resulting clusters.
- Neighborhood size influences smoothness of the learning.
- In order to achieve a good accuracy the number of learning steps needs to be high enough.
- Number of neurons need somehow to be guessed because a large number of output neurons might burn

distinct holes into the map since we don't have a clue about the variety of the data. On the other hand too few neurons affect the data exploration since the contours will eventually become blurred and affect the generalizability for unseen examples.

- A large input vector $\vec{x_i}$ and a large number of neurons is a question of performance.

20.2 Realization

One major benefit when clustering with a SOM is that knowledge is centralized where many crucial environmental data have been collected. Even if we should once or twice receive exceptional data it should score at some other place on the map and we might be capable of preserving such a learnt knowledge, perhaps a little longer then regular neural networks.

As mentioned in the latter section the problems seem quite hard to be solved by a conventional SOM. Nevertheless we wanted to convince ourselves about its capabilities by setting its different parameters such as the number of neurons, training epochs and even the dimension of the input vector.

As usual we've implemented and tested our own version of the SOM in a small little test application (See figure: 20.2).

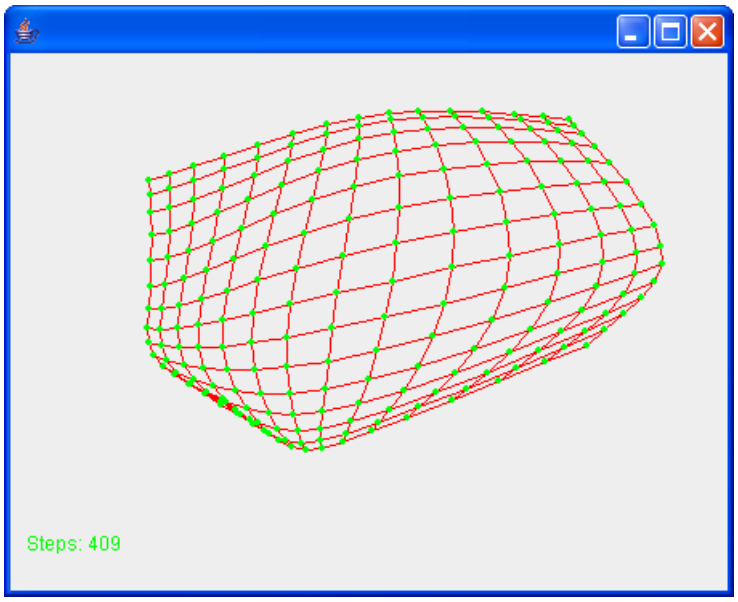


Figure 20.2: The classical 2-dim SOM net

One problem that has been mentioned in section: 20.1.2 still remains. Namely the question on: *How to evaluate the resulting clusters*

Since we wanted to test the SOM in ABI we made it fairly easy since we have prior knowledge on each of the output that we've encountered upon collecting the regular sensory inputs. Hence we simply need to store each corresponding output of vector $\vec{x_i}$ to the winner neuron. Each neuron keeps track of its data by applying a ring buffer like history that holds different datasets.
At each prediction step, we hence call up the classify method that simply determines the current winner neuron for the current environmental configuration $\vec{x_i}$.
Naturally, one way of obtaining a prediction would simply be to just ask for the average output. A slightly improved version of that can be achieved by taking the prediction from adjacent neighbors into account as well (See below).

```
1  public double classify(double[] measurement)
2  {
3      Neuron winner = getWinnerNeuron(measurement);
4
5      int col = winner.getColumn();
6      int row = winner.getRow();
7      int c = 0;
8      double accumOutput = 0;
9      for (int i = col - 1; i <= col + 1; i++)
10     {
11         for (int j = row - 1; j <= row + 1; j++)
12         {
13             if (i < 0 || j < 0 || i >= outputNeurons.length
14                     || j >= outputNeurons.length)
15                 continue;
16             if (i == col && j == row)
17                 continue;
18             accumOutput += outputNeurons[i][j].getAVGOutput();
19             c++;
20         }
21     }
22     double output = (accumOutput / c + 2 * winner.getAVGOutput()) / 3;
23     return output;
24 }
```

measurement : corresponds to the current sensory input vector $\vec{x_i}$.

getWinnerNeuron() : Returns the nearest neuron toward the input vector $\vec{x_i}$ (By calculates the euclidean distance).

getAVGOutput() : Returns the average output of the winner neuron. Hereby, the return corresponds to a probability (light on).

Of course depending on the history size of a neuron, we could think of hooking on a regular neural network to each neuron to maximize the hypothesis. However we didnt't follow up on this matter since the major problem we still haven't been able to solve. - Choosing the right number of neurons.

20.3 Evaluation and Discussion

As indicated in the beginning of this chapter, SOMs didn't turn out to be very successful (See figure: 20.3 and figure: 20.4).

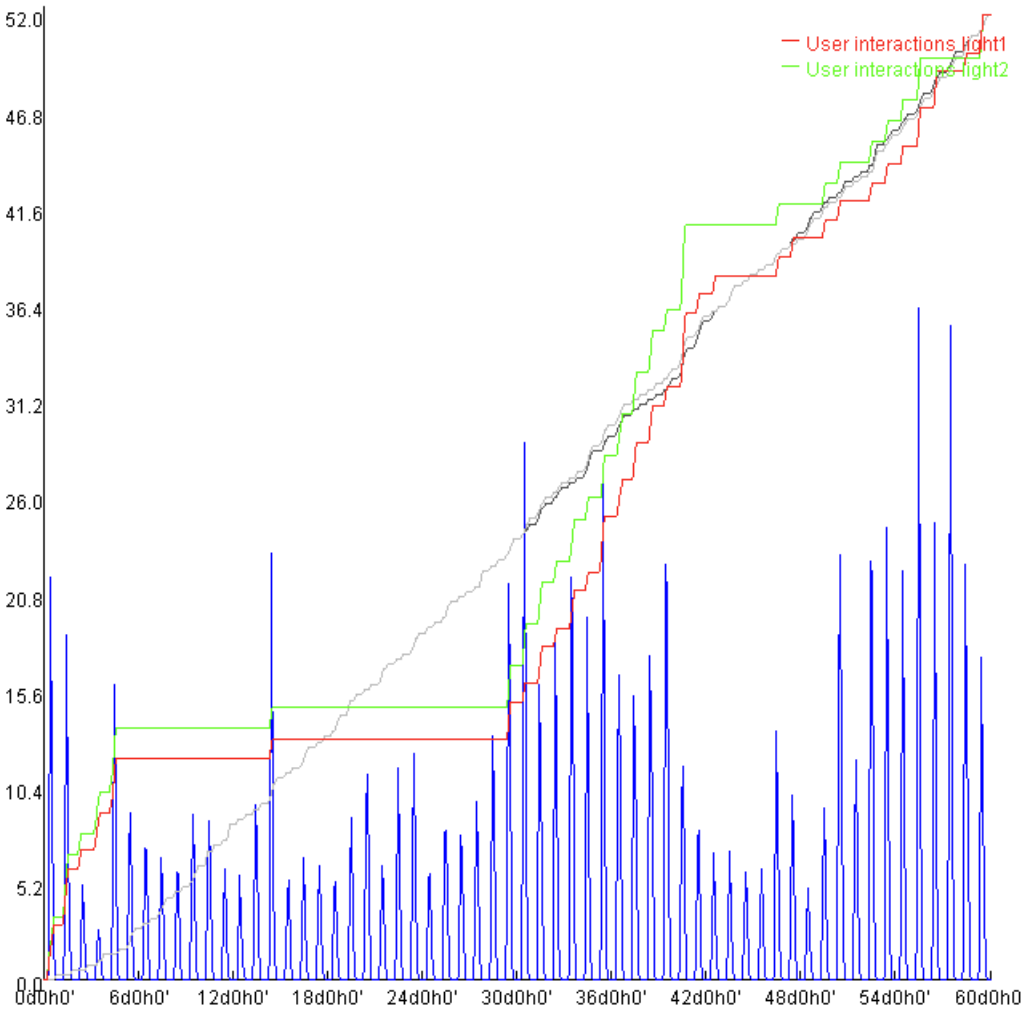


Figure 20.3: Poor results by just applying a regular SOM with a 15 by 15 network, Ordinate: User interactions

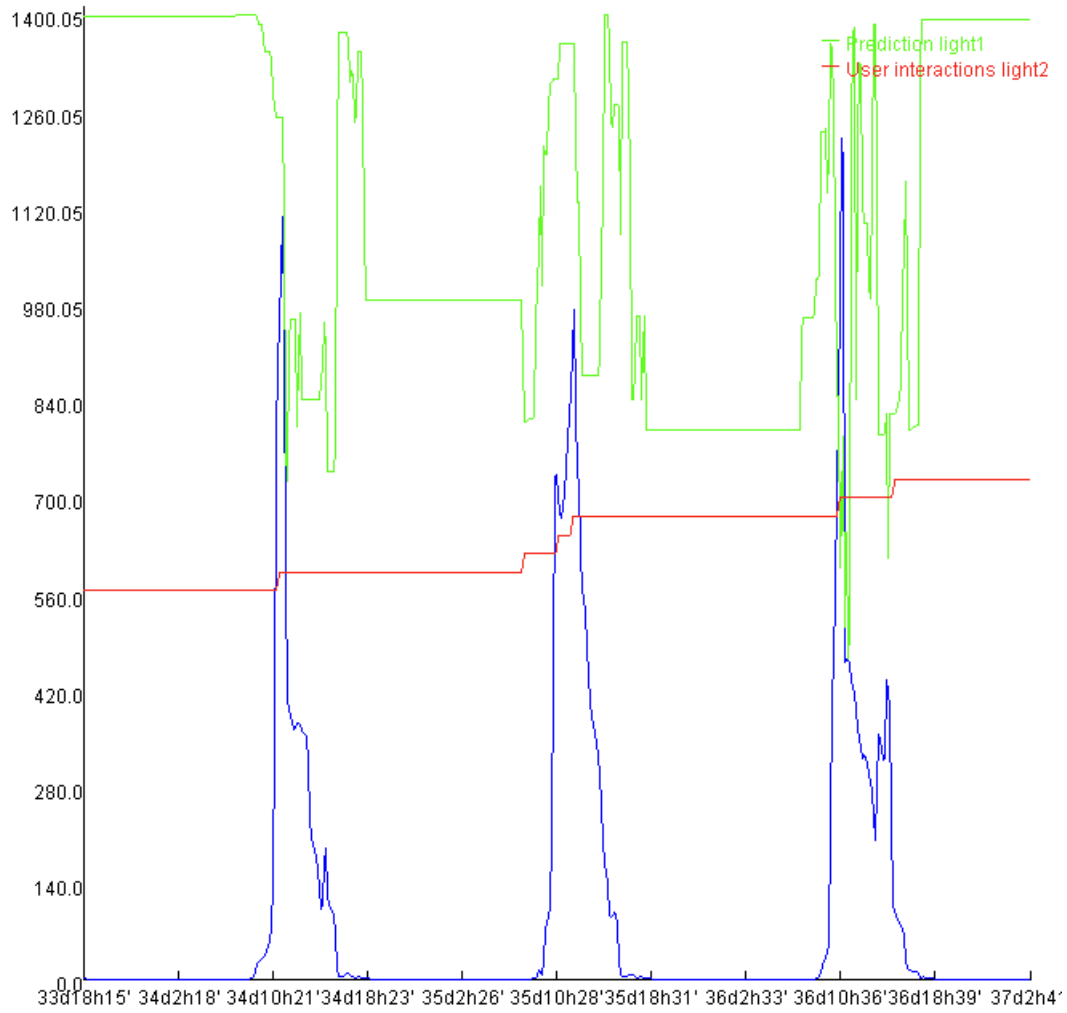


Figure 20.4: Unclear predictions are have been taken, Ordinate: Interior daylight

In general we can't really rely on the SOM at all since the magic number of neurons is unknown. Of course the prediction could have been slightly enhanced by applying a backpropagation network over the neurons. But this doesn't actually really matter since the entire concept would be specific to either the simulator or an environment later on. Any prediction algorithm should try to get to known its environment without having to deal with specific configurations (self adapting). The concept of clustering though is a very interesting fact since such an approach might provide us with additional information which would elsewise just drown by other strong input data that are infrequent.

Furthermore the algorithm is very computing intensive which impacts any deep testing in a negative way also. As mentioned in the latter section (See section: 20.1.2), the major shortcoming of the SOM algorithm has been addressed by one major paper: [YWN05] where a dynamic growing SOM has been proposed. However other cluster algorithms such as the k-means and its derivatives were at that point more appealing for us and hence, we didn't follow up on this matter any longer.

Due to the unstable results we forgo on doing any tests which illustrate the reaction to user behavior changes (See section: 17.1.2).

Chapter 21

GMeans

The latter chapter (See: 20), we mentioned that guessing the right number of neurons which actually would correspond to the number of clusters is hard. By just trial and error is totally irresponsible since we don't know the environment in that extent. Hence we need a procedure that kinda estimates the number of clusters.

Before we start to introduce a quite new cluster algorithm called *g-means* we will make a digression to clustering in general. After that we'll be introducing one of the most popular cluster algorithms called *k-means* that will partly be used by the *g-means* as well.

21.1 Overview

Clustering is an unsupervised learning problem, which tries to group a set of points into clusters such that points in the same cluster are more similar to each other than points in different clusters, under a particular similarity metric (Jain & Dubes, 1988). Such learning algorithms just observe a set of points without actually observing any corresponding class or category labels. The problem when it comes down to clustering is that the number of clusters K is quite often an unknown parameter. Hence one common problem that classical cluster algorithms such as the *k-means* bring along is that it must somehow have prior knowledge on the number of clusters. Unfortunately most problems in nature actually are not aware of its count ge on the number of clusters. Unforntunately most problems in nature actually are not aware of its count beforehand and may need to guess or by good trial and error find a suitable number of clusters that perfectly fits the data.
Basically two different types of cluster models exist.

Partitional clustering: A partitional clustering algorithm divides the data into K partitions (K given as input to the algorithm) by grouping the associated feature vectors into K distinct clusters. As mentioned one such common algorithm is the *k-means* that basically assigns each point to the cluster whose centroid is nearest. Hereby the centroid corresponds to the average value of all the points in the cluster.
The *g-means* algorithm can be seen as an advanced derivative of the *k-means*. Hence our approach will concentrate in partitional clustering (See figure: 21.1).

Hierarchical clustering: Data is not partitioned into a set number of classes, but classification consists of a series of partitions. Results can be presented as a diagram known as a dendrogram. Hierarchical clustering can be either agglomerative or divisive.
Agglomerative: The first partition is hereby contains n single member clusters and the last partition is one cluster that contains all n individual clusters (See figure:21.2).
Divisive: The first partition is only one cluster that contains all n individuals clusters. The last partition hence contains n single member clusters.

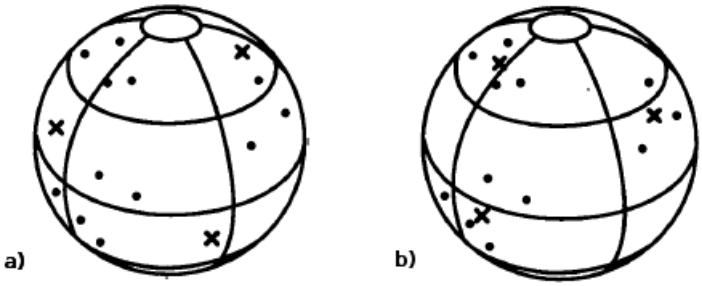


Figure 21.1: (a) organization before clustering (b) after clustering ([Lyt02])

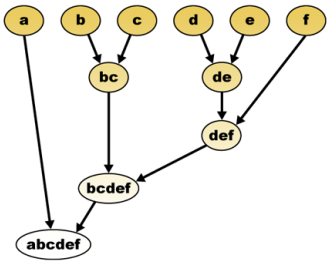


Figure 21.2: Agglomerative hierarchical clustering ([WIK])

21.1.1 A review of the K-Means algorithm

Before digging into the g-means algorithm that has been presented in a paper by Greg Hamerly and Charles Elkan back in 2003 we're first going to illustrate the k-means algorithm ([HE04]).
The k-means algorithm is one of the most popular iterative clustering methods. It's intended for situations in which all variables are of the quantitative type, and squared euclidean distance.
Its task is to classify or to group objects based on attributes or features into K number of groups. Hereby the grouping is done by minimizing the sum of squares of distances between data and the corresponding cluster centroid. Thus, the purpose of K-mean clustering is to classify the data.

What determines a "good," or representative, clustering? Consider a single cluster of points along with its centroid or mean. If the data points are tightly clustered around the centroid, the centroid will be representative of all the points in that cluster. The standard measure of the spread of a group of points about its mean is the variance, or the sum of the squares of the distance between each point and the mean. If the data points are close to the mean, the variance will be small. A generalization of the variance, in which the centroid is replaced by a reference point that may or may not be a centroid, is used in cluster analysis to indicate the overall quality of a partitioning; specifically, the error measure E is the sum of all the variances ([Fab]):
The objective that k-means tries to achieve is to minimize the function (total intra-cluster variance):

$$E = \sum_{k=1}^{k} \sum_{j=1}^{n_i} \|x_{ij} - \overline{c}_i\|^2 \tag{21.1}$$

Where x_{ij} is the j^{th} point in the i^{th} cluster, c_i is the reference point of the i^{th} cluster, and n_i is the number of points in that cluster. Hence, the error measure E indicates the overall spread of data points about their reference points. To achieve a representative clustering, E should be as small as possible.

In summary the algorithm works like this:

Result: Clustered data with k clusters
input: A set of input vectors $X = \{\vec{x_0}, \vec{x_1}, \ldots, \vec{x_m}\}$
1. Specify k, the number of clusters to be generated.
2. Init all cluster C at random, with k centers (usually $C \leftarrow \{\vec{x_0}, \vec{x_1}, \ldots, \vec{x_k}\}$*)*
repeat
    3. Remove all elements from the clusters
    4. Assign each instance to its closest cluster center using euclidean distance.
    foreach $x_i \in X$ **do**
        assign x_i *to cluster* c_{min} *where dist(*c_{min}, x_i*)* $= \arg\min_{c_j \in C}$ *dist(*c_j, x_i*)*
    end
    5. Calculate the centroid (mean) for each cluster, use it as new cluster center. Reassign all instances to the closest cluster center.
    foreach $c_i \in C$ **do**
        set center of c_i *to the mean of all elements assigned to the center* c_i
    end
until *Cluster centers don't change any more.*

Just like every other algorithm we've implemented the k-means algorithm in a 2-dimensional representation (with 2 attributes).
Figures: 21.3, 21.4, 21.5 and 21.6 illustrate how the k-means basically progresses.

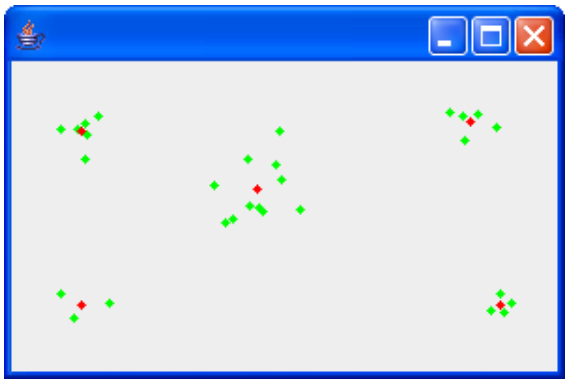


Figure 21.3: Stage: 1

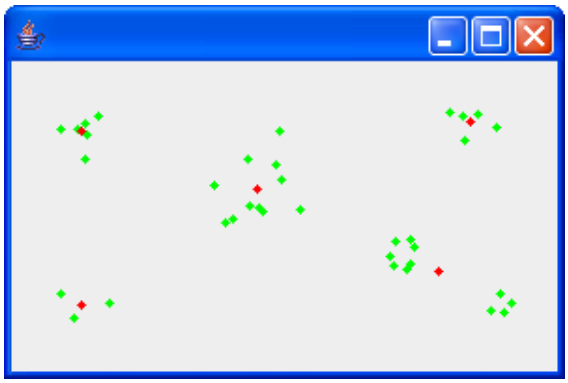


Figure 21.4: Stage: 2

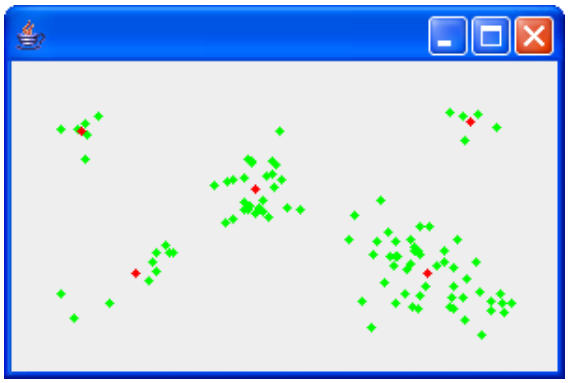


Figure 21.5: Stage: 3

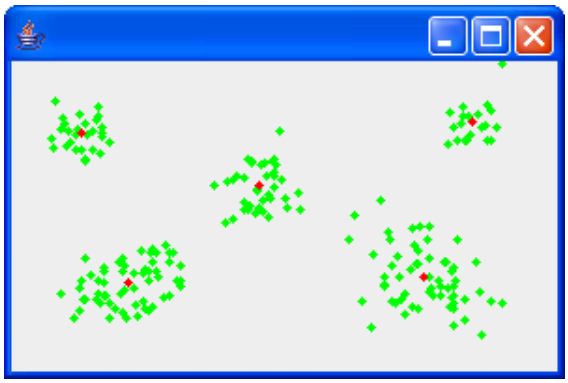


Figure 21.6: Stage: 4

However as mentioned before, most clustering algorithms such as the *k-means* require prior knowledge on the number of clusters k. Evidentially it isn't always clear what is the best value for k. For instance figure: 21.7 and 21.8 shows two situations where the number of k was improperly chosen. Since we don't have prior knowledge on the correct number of clusters, we must somehow guess the right number of clusters. The g-means algorithm, as it was proposed a couple years ago, presents a solution to this problem that we also found very interesting and worth to try.
Broadly speaking the g-means algorithm discovers an appropriate k using a statistical test for deciding whether or not to split a k-means center into two new cluster centers.

There is actually another algorithm called x-means which addresses the same problem. Also, in hierarchical clustering other methods may be employed to determine the best number of clusters. Such as merging sub-trees within the dendrogram. But since we stick on using regular *Partitional clustering* we didn't consider this option.

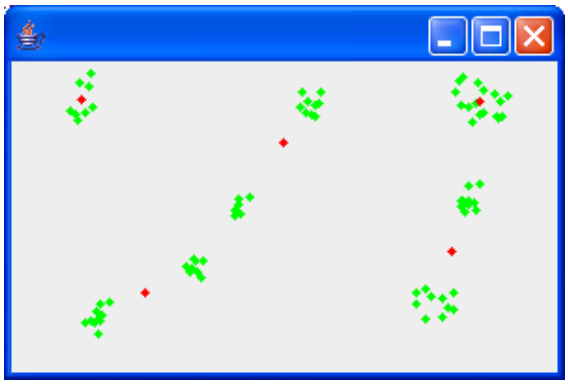


Figure 21.7: Improperly chosen number of k. Too few clusters

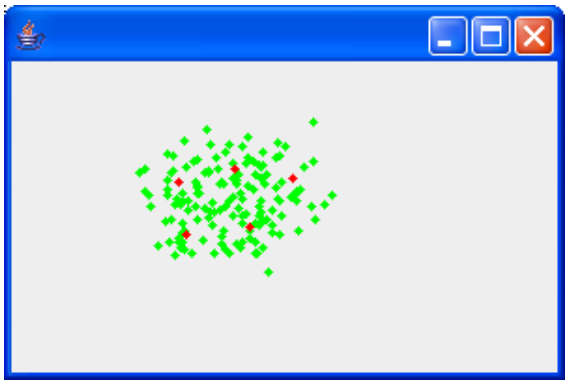


Figure 21.8: Improperly chosen number of k. Too many clusters

21.1.2 PCA

Back in chapter: 11 we mentioned the importance of data *pre-processing*. There we briefly discussed that input data complexity could be reduced by applying a dimensionality reduction. This is exactly what we'll be applying in the g-means algorithm. In order to simplify the test for a gaussian fit the g-means proposes to project the input data to one dimension where such a test is simple to apply. The major benefit when projecting to one-dimension is that the representation of the data only considers data along the direction that k-means has found to be important for separating the data.
Hence in order to accomplish the projection we must first figure out the direction of the vector along the attribute space. This vector corresponds to nothing less then the *main principal component* that will help us to explore the data points that have the largest variance.

Some of the readers may not be familiar with the definition or the calculation of the main principal component. Therefore we briefly summarize it since it contributes quite an important factor to the g-means algorithm understanding. The theory about PCA is has been obtained from: [Hol].

In statistics, principal components analysis (PCA), is a technique that can be used to simplify a dataset. More formally it is a linear transformation that chooses a new coordinate system for the data set such that the greatest variance by any projection of the data set comes to lie on the first axis (then called the first or main principal component), the second greatest variance on the second axis, and so on.
PCA can be used for reducing dimensionality in a dataset while retaining those characteristics of the dataset that contribute most to its variance by eliminating the later principal components (by a more or less heuristic decision). These characteristics may be the "most important", but this is not necessarily the case, depending on the application. PCA is also called the Karhunen-Loève transform. ([WIK])

One simple way to calculate the main principal component is through an empirical covariance matrix of the input vector $\vec{x}$. Where $\vec{x}$ corresponds to a (feature) vector in the form of: $x = \{x_1, x_2, \ldots, x_n\}^T$.
The covariance is the measure of how much two variables vary together. This means that the covariance becomes more positive for each pair of values which differ from their mean in the same direction, and becomes more negative with each pair of values which differ from their mean in opposite directions. In this way, the more often they differ in the same direction, the more positive the covariance, and the more often they differ in opposite directions, the more negative the covariance. There are many good resources about covariance calculations on the web ([MAT]).
The covariance is defined as:

$$cov(X, Y) = E\{(X - \mu)(Y - \nu)\} \tag{21.2}$$

Where $E(X) = \mu$ and $E(Y) = \nu$ and E is defined as the expected value

By finding the eigenvalues and eigenvectors of the *covariance matrix*, we find that the eigenvectors with the largest eigenvalues correspond to the dimensions that have the strongest correlation in the dataset.

The mean of that population is denoted by:

$$\mu_x = E\{x\} \tag{21.3}$$

Thus the following *covariance matrix* of the same data set can be extracted:

$$C_x = E\left\{(x - \mu_x)(x - \mu_x)^T\right\} \tag{21.4}$$

The components of C_x, denoted by C_{ij}, represent the covariances between the random variable components x_i and x_j. The component C_{ii} is the variance of the component x_i. The variance of a component indicates the spread of the component values around its mean value. If two components x_i and x_j of the data are uncorrelated, their covariance is zero ($C_{ij} = C_{ji} = 0$). The covariance matrix is, by definition, always symmetric. From a sample of vectors $x_1, \ldots, x_M$ we can calculate the sample mean and the sample covariance matrix as the estimates of the mean and the covariance matrix. From a symmetric matrix, we can calculate an orthogonal basis by finding its eigenvalues and eigenvectors. The eigenvectors e_i and the corresponding eigenvalues λ_i are the solutions of the equation:

$$C_x e_i = \lambda_i e_i \quad , where\ i = 1, \ldots, n \tag{21.5}$$

For simplicity we assume that the λ_i are distinct. These values can be found, for example, by finding the solutions of the characteristic equation.

$$|C_x - \lambda I| = 0 \tag{21.6}$$

where the I is the identity matrix having the same order than C_x and the $|\ldots|$ denotes the determinant of the matrix. If the data vector has n components, the characteristic equation becomes of order n. This is easy to solve only if n is small. Solving eigenvalues and corresponding eigenvectors is a non-trivial task, and many methods exist. One way to solve the eigenvalue problem is to use a neural solution to the problem. The data is fed as the input, and the network converges to the wanted solution. By ordering the eigenvectors in the order of descending eigenvalues (largest first), one can create an ordered orthogonal basis with the first eigenvector having the direction of largest variance of the data. In this way, we can find directions in which the data set has the most significant amounts of energy.
Instead of using all the eigenvectors of the covariance matrix, we only need to extract the main component (PCA1) (See figure: 21.9).

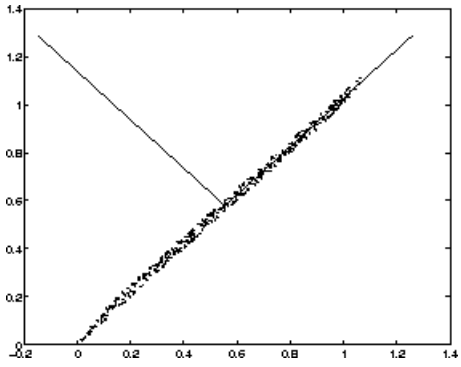


Figure 21.9: Main principal component (PCA1)

In summary: The purpose of the Principal Components Analysis is

- A method that reduces data dimensionality by performing a covariance analysis between factors. As such, it is suitable for data sets in multiple dimensions, such as a large experiment in gene expression.
- PCA is recommended as an exploratory tool to uncover unknown trends in the data
- PCA will explore correlations between samples or conditions

21.2 G-Means algorithm

We now would like to introduce the g-means algorithm in its detail since the applied principles are from a algorithmic point of view very interesting. Later we will then illustrate our version of the algorithm to be suitable for ABI. Especially what goals we're trying to achieve with the help of the g-means algorithm. Up to this point it's just important to know that we want to use clustering for further reasoning. Since the k-means algorithm basically requires to have prior knowledge on the correct number of clusters (See figures: 21.7 and 21.8). Hence we'll be using the g-means algorithm that counteracts this issue and is therefore first introduced.
The consideration and the development of the g-means algorithm has chiefly been inspired by three different sources. The most important sources were the papers: ([HE04], [DHZS02]) and partly also [PM00]).

Usually the G-means algorithm works by starting with two or more randomly chosen k-means centroids and then continually starts to increase the number of cluster centroids. Hereby at each iteration of the algorithm we will split those centroids whose "member data" appear not to come from a Gaussian distribution. Between each round of splitting, the regular k-means is run on the entire dataset and within the centroids to refine the current solution.
G-means repeatedly makes decisions based on a statistical test for the data assigned to each center. If the data currently assigned to a k-means center appear to be Gaussian, then we want to represent that data with only one center. Correspondingly if the same data do not appear to be Gaussian, then it will use multiple centers to model the data properly (See figures further below). The algorithm will run k-means multiple times (up to k times when finding k centers), so the time complexity is at most $O(k)$ times that of k-means.
The k-means algorithm as well as other algorithms that make use of the euclidean distance measure assume that the data points in each cluster are spherically distributed around the center.
Algorithms such as the g-means incorporates a different measure. Namely the assumption that the data points are scattered in a gaussian distribution kind of way. The Gaussian distribution test that the paper proposes is based on a test that tries to detect whether the data assigned to a center are sampled from a Gaussian.
In order to test the assumption the authors of the paper ([HE04]) applied a simple but effective test: *The Anderson-Darling test* (See below). The *Anderson-Darling test* is commonly used to test if a sample of data come from a population with a specific distribution. In this case a Gaussian distribution. According to the *Anderson-Darling test* we can state the following hypothesis ([MAT]):

- H_0: The data around the center are sampled from a Gaussian.
- H_1: The data around the center are not sampled from a Gaussian.

If we accept the null hypothesis H_0, then we believe that the one center is sufficient to model its data, and we should **not** split the cluster into two sub-clusters. If we reject H_0 and accept H_1, then we want to split the cluster.
The one-dimensional *Anderson-Darling test* has been shown empirically to be the most powerful normality test that is based on the empirical cumulative distribution function (ECDF).
The Anderson-Darling test is defined as:

$$A^2 = -N - S \tag{21.7}$$

Where S:

$$S = \sum_{i=1}^{N} \frac{2i-1}{N} \left[\ln(F(Y_i) + \ln(1 - F(Y_{N+1-i}))\right] \tag{21.8}$$

Where N is the sample size, Y_i are the *ordered* data and F is the *ECDF function* that is expressed in terms of the density function from a normal distribution and hence defined as:

$$F(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} \exp -\frac{(u-\mu)^2}{2\sigma^2}\, du \tag{21.9}$$

Since the the Anderson-Darling test makes use of a specific distribution, a tolerance like value called *critical value* must be calculated that allows a more sensitive test to be conducted. The advantage of such a test is that it also considers the number of samples to be tested by incorporating N in the calculation of the critical value (See equation: 21.8). This will prevent the g-means algorithm from making bad decisions about clusters with few data points. Hence each distribution must calculate and provide its own critical values.
Unfortunately it's kinda hard to find tables of critical values somewhere to fit our purpose. Hence according to the paper ([HE04]) we use the same confidence level α in our algorithm which corresponds to an α of 0.0001. The test is a one-sided test, and the hypothesis, that the distribution is of a specific form is rejected, when the test statistic, A, is *greater* than the critical value. Specifically speaking the critical value (tolerance) for an α of 0.0001 corresponds to 1.8692. As a simple example if the g-means scores a A^2 statistic that goes beyond the critical value we reject the hypothesis h_0 and consequently accept the new two centroids since they evidentially are not sampled from a gaussian.

Now we've discussed how to basically verify if the the data points around a distinct center are sampled from a Gaussian or not. We barely talked about how to split a centroid for which we would like to run the k-means on. Splitting in this context means that we need to cluster the data from one centroid to two centroids that will later be tested whether the data points around the original centroid are sampled from a Gaussian or not.
Hereby we define a new measure $\vec{m}$ that defines the space between each of the new centroids to the original centroid. Hence the new two centroids will be initialized with $c \pm m$, where c is a center and $\vec{m}$ is chosen.
The paper suggested two different ways to determine which direction we should split to and thus find out $\vec{m}$ that will help us to calculate the two new centroids (vectorgraphically). The first proposal was to choose m as a random d-dimensional vector such that $\|\vec{m}\|$ is small compared to the distortion of the data. However a more convenient way to calculate m is to compute the *main principal component* $PCA1$ (See section: 21.1.2) that we shall also name $\vec{s}$ of the data (having eigenvalue λ). Hereby m is calculated using:

$$m = \left(\sqrt{\frac{2\lambda}{\pi}}\right)\vec{s} \tag{21.10}$$

This deterministic method is then suppose to place the two new centers in their expected locations under H_0.

Having once discussed the mathematics of the g-means we can now go over to its entire algorithm.

Result: clustered data with gaussian distributed elements in the clusters
input: A set of input vectors $X = \{\vec{x_0}, \vec{x_1}, \ldots, \vec{x_m}\}$
1. init C with a initial set of centers (usually $C \leftarrow \{\text{mean}(\vec{x})\}$)
repeat
  2. $C \leftarrow kmeans(C, X)$
  foreach $\vec{c_j} \in C$ **do**
    3. let $X_j = \{\vec{x_i} | \text{class}(\vec{x_i}) = j\}$ be the set of data points assigned to center $\vec{c_j}$
    4. split $\vec{c_j}$ into two child centers $\vec{c_{j0}}$ and $\vec{c_{j1}}$ (see text for good ways to do this)
    5. run k-means on this two children with X_j
    6. $\vec{v} \leftarrow \vec{c_{j0}} - \vec{c_{j1}}$ is the vector connecting both centers
    7. project all elements from X_j to $\vec{v}$. $x'_i = \langle \vec{x_i}, \vec{v} \rangle / \|\vec{v}\|^2$. X'_j holds the 1-dimensional representation of this projected data.
    8. transform X'_j so that its mean will be 0 with a variance of 1
    9. calculate all $z_i = F(x'_i)$ (see text for details)
    if $A^2_*(Z) \leq \alpha$ **then**
      10. keep $\vec{c_j}$
    else
      11. replace $\vec{c_j}$ with $\vec{c_{j0}}$ and $\vec{c_{j1}}$
    end
  end
until *no more centers are added*

The operation of the g-means algorithm can visualized in the following way (See figures: 21.10, 21.11, 21.12, 21.13 and 21.14):

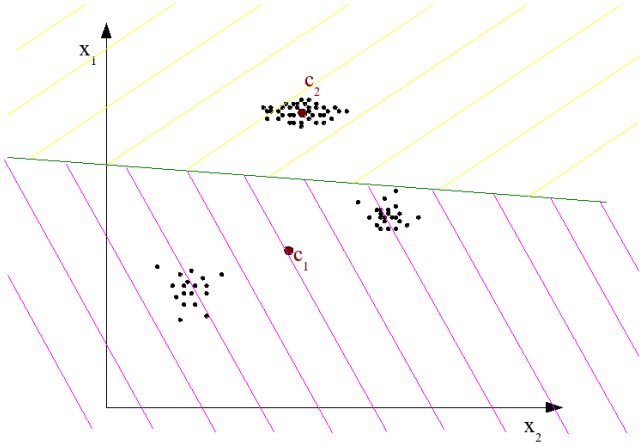


Figure 21.10: Stage: 1, We see that three major accumulating scopes are starting to materialize

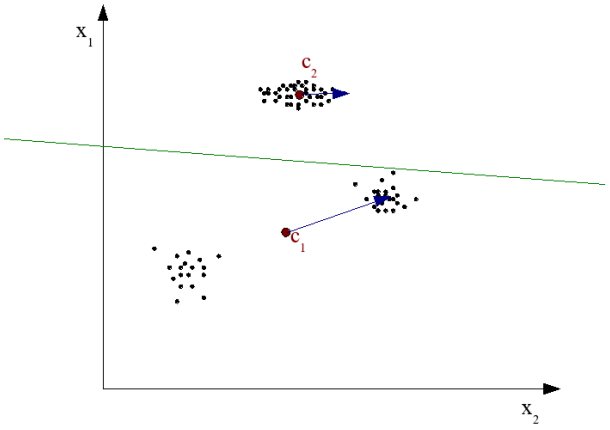


Figure 21.11: Stage: 2, For each centroid the *main principal component* is calculated to scope possible trends.

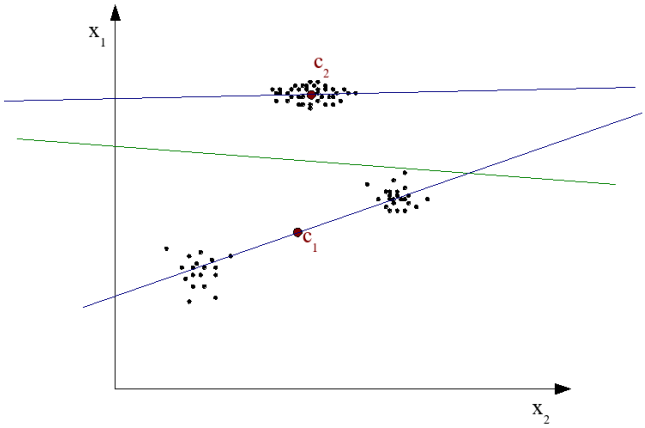


Figure 21.12: Stage: 3, The trend of the direction is now clear

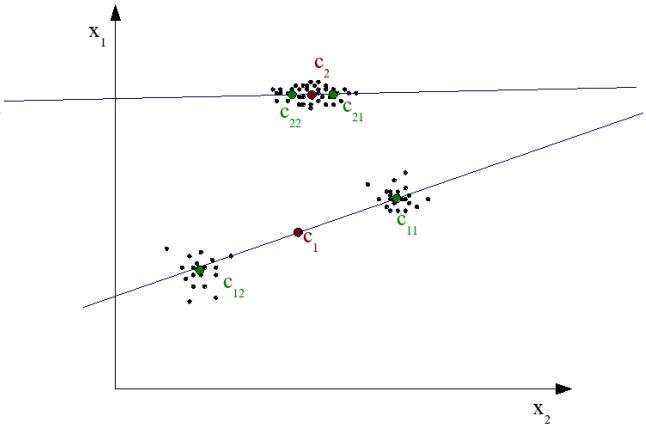


Figure 21.13: Stage: 4, The split has been initiated. The hypothesis H_0 and H_1 are to be investigated

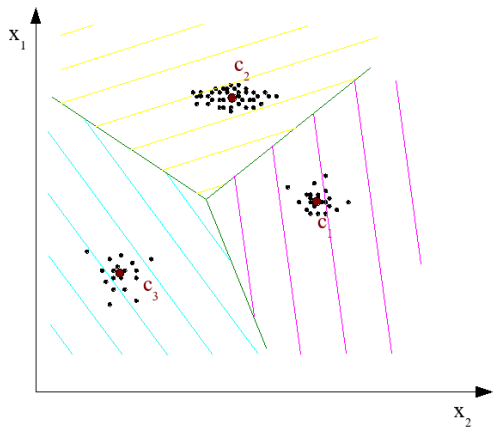


Figure 21.14: Stage: 5, Evidentially the hypothesis for C_2 succeeds whereas the hypothesis for C_1 is rejected and thus the two new centroids (C_{11} and C_{12})are kept.

21.3 Realization

Before we dive into further steps we'll illustrate our implementation of the g-means algorithm. Just like the k-means we can give a visual representation (2 dimensional) of the g-means (See figures: 21.15, 21.16, 21.17, 21.18, 21.19, 21.20:

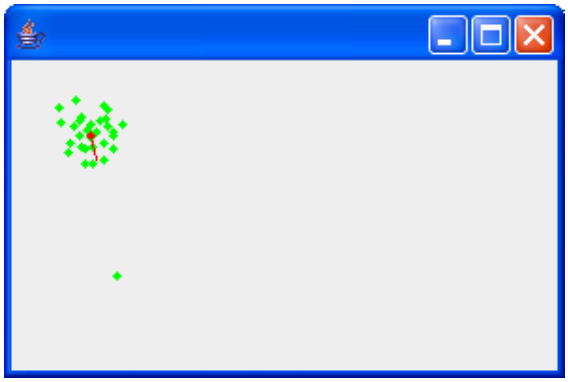


Figure 21.15: Stage 1, First centroid created, the covariance starts to increase

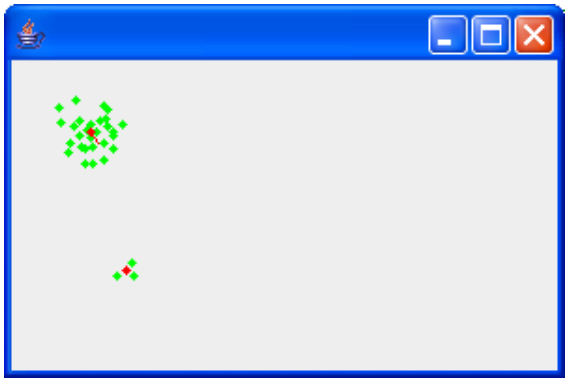


Figure 21.16: Stage 2, 2nd centroid created

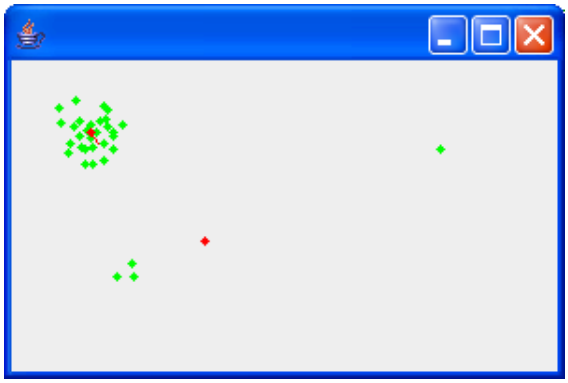


Figure 21.17: Stage 3, 2nd centroid starts to shift, the covariance starts to increase

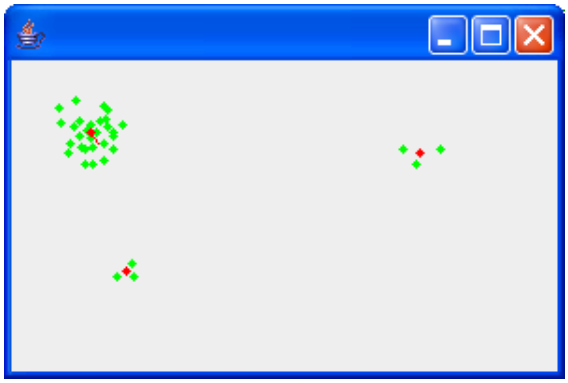


Figure 21.18: Stage 4, 3rd centroid created

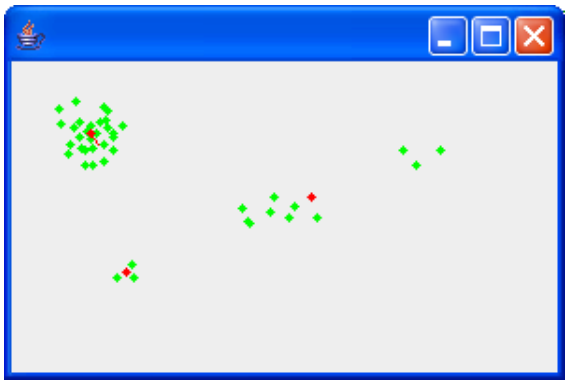


Figure 21.19: Stage 5, 3rd centroid starts to shift, the covariance starts to increase

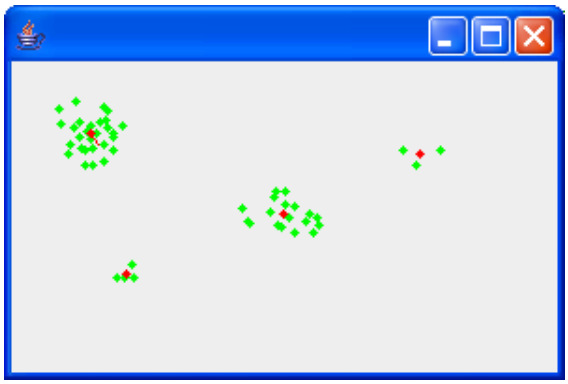


Figure 21.20: Stage 6, 4th centroid created

Now we're ready to cluster our environmental data. What next?

It has been addressed that the original intention of clustering is to improve the hypothesis. We achieve this by taking each of the data points provided by each of the gained clusters and simply perform a regular regression technique on them. Further two steps are necessary to realize a long term knowledge keeping. These are:

1. Each cluster is limited in size and thus we advance a) LTM-STM and b) speedup and c) accuracy
2. At each clustering step we perform a re-clustering. Hereby the re-clustering will be launched with an initial set of clusters (Depending on the number of clusters we had before). Thus we enable the clusters to be merged with other clusters and also prevent a form of over-clustering.

Our regression technique takes place by using a regular artificial neural network (See chapter: 19). Since it isn't always apparent where and when to perform the regression, we implemented two solutions. One solution performed a regression on each contents of a cluster (local) whereas the other technique did an overall regression on all the gained clusters (global).
Local regression was less successful since it rather tended to overfit. However a better suited size of the clusters would reduce overfitting.

Broadly speaking we use clustering to preserve collected data into categories. Herewith we provide a solution against losing exceptional knowledge since we're carrying some of the clusters that we also call *exceptional clusters* around us, all the time.
For a better illustration we could bring each of the clusters into linguistic form such as $FogCluster$ or $BrightCluster$. Therefore it is clear that limiting a cluster is necessary and obvious since if we would elsewise have tons of data representing bright days or hot days ($BrightCluster$). Such data would then rather tend to start to dominate all other clusters and this is something we don't want.
Nevertheless by being capable to store for instance behaviors from fogy days ($FogCluster$) which is rather sparse then frequent we can always re-predict back to such re-curring behaviors at every instance even across decades (assuming that it has never been fogy again). This because we're now capable to basically store *the right kind set of data*. This is exactly what is achieved by applying such a procedure since basically the huge amount of ambient noise which can now also be considered as the the unnecessary amount of data since each such "day" can now be represented which an equal set of data e.g 50 data points for a sunny day would be by far be sufficient then be represented by 200 data points. Especially such a representation would else rather tends to pull other "days" up since the amount of data is out of balance.
In summary we can say that clustering is an advanced form of data pre-processing that finally might enable us to approximate a better environment function since transitions should be faster re-produced upon trend changes. In oder words the goal that we hope to achieve with the aid of clustering in general is to enhance the prediction on distinct spots where conventional algorithms would fail to provide a fast reliable but necessary prediction since most algorithms are quite often not agile enough to react to such changes within a reasonable time.

The overall algorithm can therefore be shown to work like this: To round up this section the train method

Result: trained ANN filtered with help of the g-means algorithm
input: A set of continuous input-pairs $\Psi\{\vec{x}, y\}, y \in [0; 1]$
repeat
  fetch next input $in_t \leftarrow \{\vec{x}_t, y_t\}$
  if *somebody present* **then**
    put $in_t \rightarrow traininqueue$
    $counter \leftarrow counter + 1$
    if $counter \geq trainingsize$ **then**
      makeclusters()
      trainANN() (the ANN with the g-means clusters is being trained)
      $counter \leftarrow 0$
    end
  end
  $t \leftarrow t + 1$
until *Stop condition reached*

is now given that illustrates how the neural network that approximates the current environment function from k-means clusters from the g-means. Section: 19.2.2 already discussed some aspects concerning artificial neural networks. For instance: keeping the weights from previous training and such (Line 4-7). Initially though a new network instance is created with a number of input neurons which size corresponds to the current length of the input vector $\vec{x}$. The network is further composed of 6 hidden neurons and one output neuron.
Line 9 shows how the data (input and output) are obtained from the currently existing clusters which is then used to train the network (Line 23). Also note that the training epoch is lowered (Line 31) when the first training pass has been completed (initial).

```
1  private void trainANN()
2  {
3
4      if (this.net == null)
5      {
6          this.net = new BackPropNet(inputs.length, new int[] { 6 }, 1);
7      }
8
9      LearnData[] data = kmeans.getAllLearnData();
10
11     double[][] inputs = new double[data.length][];
12     double[] outputs = new double[data.length];
13     for (int i = 0; i < data.length; i++)
14     {
15         inputs[i] = data[i].getInputs();
16         outputs[i] = data[i].getOutput();
17     }
18
19     double error;
20
21     for (int i = 0; i < trainEpochs; i++)
22     {
23         net.train(inputs, outputs);
24
25         error = net.getTotError();
26         if (error <= 0.01 * outputs.length)
27         {
28             break;
```

```
29        }
30    }
31    trainEpochs = MAX_LATER_TRAIN_EPOCHS;
32 }
```

21.4 Evaluation and Discussion

The theory of clustering sounds quite reasonable but the practice (currently almost only the simulator) showed us that providing a decent synthetic set in the simulator is very hard. On the other hand accomplishing a convincing test is very essential for clustering. A "bad set" will result in wrong clusters to be created and thus presumably ends up in a continuous mis-prediction. It might even get worse then a regular backpropagation network which currently rather tends to fall into local minimas in the test-bed anyway due to in-appropriate blind movements. Especially upon conducting the second test where all days are repeated (See section: 17.1.2).
However the g-means was performing quite well in the test set (See figure: 21.21).

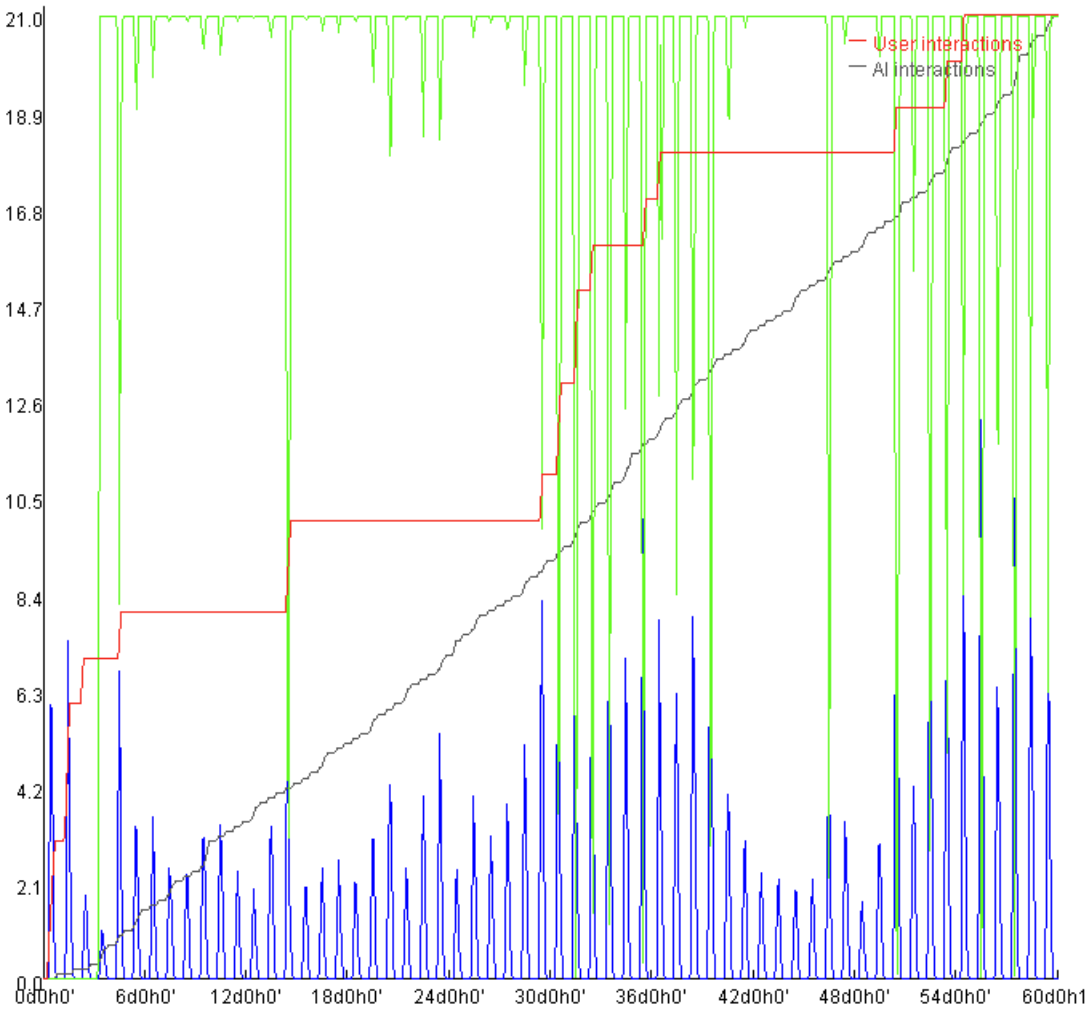


Figure 21.21: Very clear decisions have been taken by applying a global regression over the clusters, Ordinate: User interactions

On the whole we're confident that in real environments, clustering will be assisting other conventional methods to achieve more accurate predictions since long re-learn phases can massively be lowered herewith.

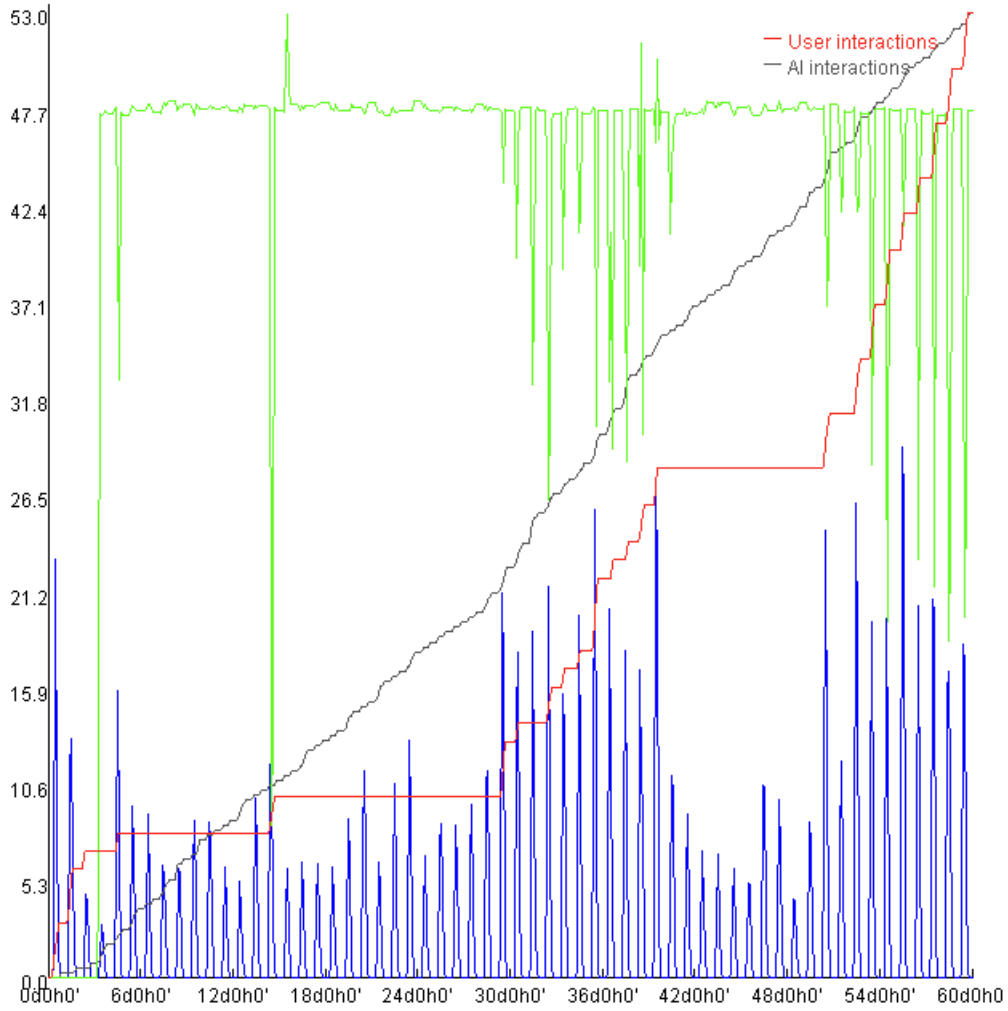


Figure 21.22: Performing a local regression on each cluster is in the current condition, as indicated in section: 21.3, less efficient then a global regression. This is due to overfitting. By increasing the history size in each cluster, we could counteract this issue a little. However if the resolution is too high we face the problem of not being able to react to user behavior changes. We'll be trying both solutions since an optimal chosen cluster size could achieve pretty good results and will even have the advantage of taking changing user behaviors a bit stronger into account. Additionally, a more inexact clustering may even be more suitable, in order to lessen the amount of clusters. Ordinate: User interactions

Chapter 22

GNG

Recent tests and the entire theory have shown that clustering could bring some major improvements to our prediction system. Hence we continued to follow up on this matter a little longer. The ANN g-means algorithm presented in the last chapter (See section: 21.3) is very good at predicting recuring patterns that had occurred in the past. Nevertheless we were interested if we could find a similar method that might even score better results as the g-means has been accomplishing so far.
Hence another clustering technique that we've investigated is *Neural Gas* (NG).

22.1 Overview

The architecture of a NG is very simple since it is composed of only a set of output neurons which are not interconnected with each other unlike the SOM (See chapter: 20). Frankly speaking we can hardly consider it as a network because all reference vectors $\vec{w_{nx}}$ that it owns are determined from the "input neurons" (See figure: 22.1)

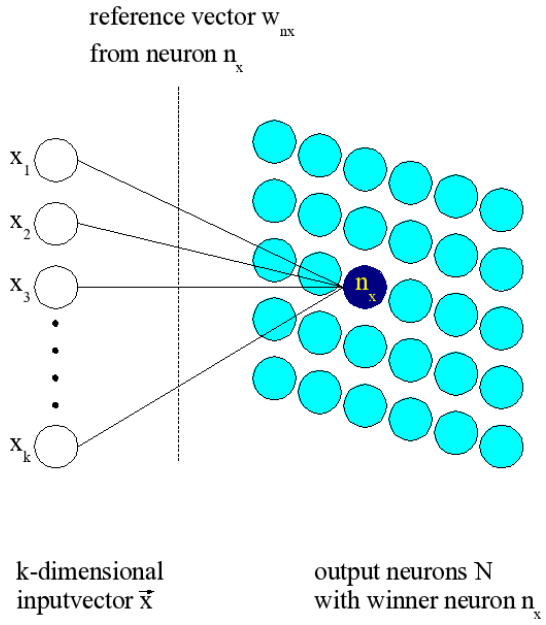


Figure 22.1: Neural Gas (NG) structure

The NG algorithm ([Fri95]) applies a soft competitive learning algorithm that alters its weights similar to Self Organizing Maps.
The NG algorithm works by sorting each neuron n_j by its distance given from their reference vectors w_{nj}. Hereby the sorting is done for each input vector $\vec{x}$.
Based on this sorted order the reference vector $\vec{w_{nj}}$ of each neuron is altered according to a dynamic calculated data. By repeating this process, the neurons will eventually organize themselves into a fixed number of clusters that of course complies to the number of neurons being used.

The complete neural gas algorithm is the following:

Result: Neural Gas (NG) clustering
input: A set of input vectors $X = \{\vec{x_0}, \vec{x_1}, \ldots, \vec{x_m}\}$
repeat
1. initialize all neurons n and hence the reference vector $\vec{w_n}$ with small random numbers e.g, between -0.05 and 0.05
2. determine and initialize the initial learn rate: α_0 and the minimal learn rate: α_{min}
whereas $\alpha_0 > \alpha_{min}$
as well as the initial neighborhood distance: λ_0 and its minimal neighborhood distance: λ_{min}
whereas $\lambda_0 > \lambda_{min}$

3. fetch next input vector $\vec{x_t}$

foreach *neuron* n_j **do**
$dist_j = \|\vec{x_t} - \vec{w_j}\|$
end

5. sort distances to a sequence S where $S = \{s_1, s_2, \ldots, s_n\}$
whereas: $s_1 \leq s_2 \leq \ldots \leq s_n$

foreach *neuron* n_j **do**
6. alter the weights for each neuron n_j by calculating:

$$w_{js} \leftarrow w_{js} + \alpha_t h(t,k) \cdot (\vec{x_t} - \vec{w_{js}})$$

Hereby $h(t, n_j)$ is calculated using:

$$h(t, n_j) = e^{\left(\frac{1-n_j}{\lambda_t}\right)}$$

$$\alpha_t = \alpha_0 \left(\frac{\alpha_{min}}{\alpha_0}\right)^{\frac{t}{t_{max}}}$$

$$\lambda_t = \lambda_0 \left(\frac{\lambda_{min}}{\lambda_0}\right)^{\frac{t}{t_{max}}}$$

end
$t \leftarrow t + 1$
until *Stop condition reached*

Quite often NG is illustrated by providing an infinite data source that is usually distributed following different shapes of data e.g. a ring (See figure: 22.2).

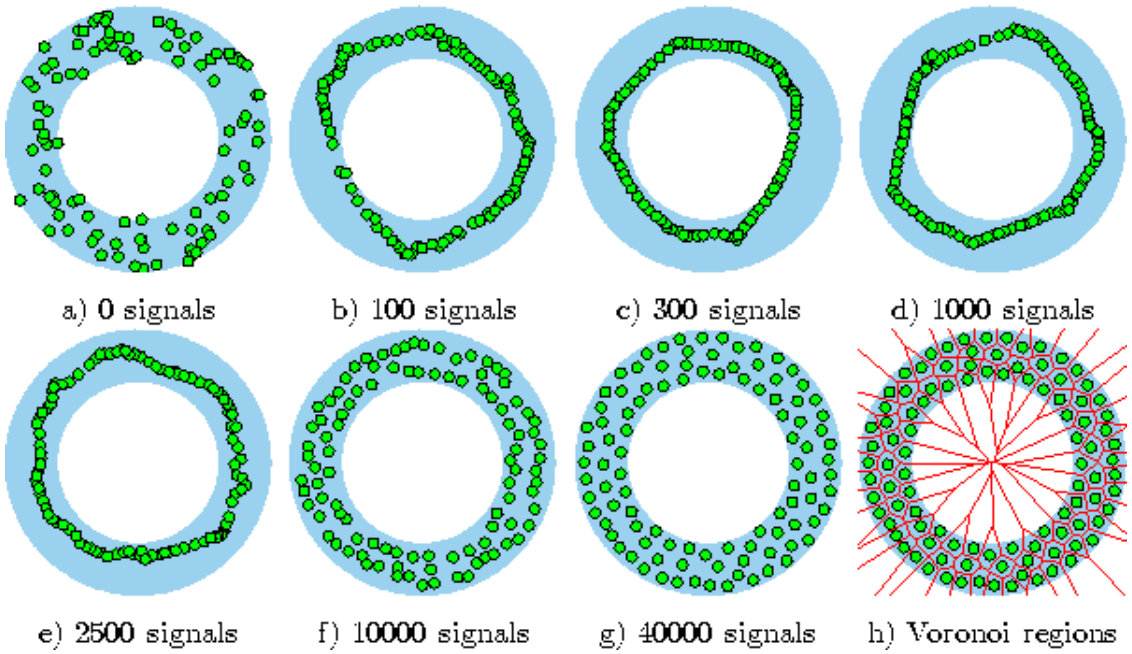


Figure 22.2: Neural Gas (NG) in action. Data source: Ring, ([Fri95])

22.2 GNG Algorithm

The NG however suffers, similar to the SOM, exactly from the same kind of issue. The unknown number of neurons. Luckily there is dynamic version of the NG algorithm that employs a set of growing gases (neurons) to be inserted However the new algorithm that is called GNG (Growing Neural Gas) distinguishes from the regular NG algorithm quite a bit.

The consideration and the development of the GNG algorithm has chiefly been inspired by two major sources: ([UL03], [GNG]).
Although our implementation differs from the original one as we'll see later.

Lots of applications which use any kind of artificial neural networks such as SOMs and regular artificial neural networks, face the problem to seek for the right number of neurons. Especially regular artificial neural networks since an optimal number of hidden neurons (when considering a three-layer ANN) can quite often only be determined by trial and error.
If the number of neurons is too high we're rather tend to overfit since most networks start to get over-trained and thus lose their generalizability.

To counteract the problem of finding the right set of clusters or neurons when applying the neural gas approach, is an advanced dynamic form that is, as it has been mentioned, the GNG (Growing Neural Gas) algorithm.
The procedure that is applied in the GNG is to first start with a small amount of neurons and as the knowledge base gets higher incorporate new neurons that will help to model and represent the input data.
Actually, to tell the truth the GNG doesn't actually exhibit any similarity to the NG since lots of it is accomplished differently.
One major difference we notice that neurons in the GNG are kind of loosely connected with each other. Loosely in the sense that each connections may be evicted upon being aged-out across a distinct period of time. Further we can identify that the weights are not altered based on a global neighbor function anymore but rather a local one that is only applicable for the "winner" neuron n_1 and the "second winner" neuron n_2. For each input pattern the neurons n_1 and n_2 are determined and will also be inter-connected with each other (when the connection doesn't exist).
Generally one can say that new neurons are actually gradually created and others evicted upon being no

longer referenced by other conntections. New neurons are created between the "worst" and the "second worst" neuron. The term "worst" hereby corresponds to the biggest accumulated error that is calculated at every round. Hence we can expect the newly inserted neuron to reduce this error in the near future.
This is reasonable since we need to add more neurons, in order to provide a better knowledge representation for those neurons which happen to be more often visited then others. To get back to the aging: The aging process is conducted on a roundly basis for those connections that are barely used and vise versa reset back to zero when being once recalled back to life.

Having once discussed the general principles of the GNG algorithm we can now illustrate the GNG in its entire form:

Result: trained growing neural gas
input: A set of input vectors $X = \{\vec{x_0}, \vec{x_1}, \ldots, \vec{x_m}\}$
1. initialize gas layer $N = \{n_1, n_2\}$
2. initialize reference vectors $\vec{w_{n_1}}, \vec{w_{n_2}}$ randomly
3. initialize set of connections $C = \{\}$
repeat
  4. $step \leftarrow step + 1$
  5. fetch input $\vec{x_{in}}$
  6. find nearest neuron s_1
  7. find 2^{nd} nearest neuron s_2
  if $c_{s_1 s_2} \notin C$ **then**
    8. create connection $c_{s_1 s_2}$, $C \leftarrow C \cup \{c_{s_1 s_2}\}$
  end
  9. set $age(c_{s_1 s_2}) = 0$
  10. correct local Error: $E_{s_1} \leftarrow E_{s_1} + \|\vec{x_{in}} - \vec{w_{s_1}}\|\hat{2}$; 11. actualize reference vector from s_1: $\vec{w_{s_1}} \leftarrow \vec{w_{s_1}} + \epsilon_{winner} \cdot (\vec{x_{in}} - \vec{w_{s_1}})$
  foreach $n_i \in neighbors(s_1)$ **do**
    12. $\vec{w_{n_i}} \leftarrow \vec{w_{n_i}} + \epsilon_{neighbor} \cdot (\vec{x_{in}} - \vec{w_{n_i}})$
    13. increment age of connection $age(c_{s_1 n_i}) \leftarrow age(c_{s_1 n_i}) + 1$
  end
  foreach $c_{n_i n_j} \in C$ **do**
    if $age(c_{n_i n_j}) > age_{max}$ **then**
      14. remove $c_{n_i n_j}$, $C \leftarrow C \setminus \{c_{n_i n_j}\}$
    end
  end
  foreach $n_i \in N$ **do**
    if $neighbors(n_i) = \{\}$ **then**
      15. remove n_i, $N \leftarrow N \setminus \{n_i\}$
    end
  end
  if $step \equiv 0 \pmod{\lambda}$ **then**
    16. n_p is the neuron with the highest accumulated error
    17. let n_q be the neighbor neuron from n_p with the highest accumulated error
    18. create neuron n_r with $\vec{w_{n_r}} = 1/2 \cdot (\vec{w_{n_p}} + \vec{w_{n_q}})$, $N \leftarrow N \cup \{n_r\}$
    19. create connections $c_{n_r n_p}$ and $c_{n_r n_q}$ and remove $c_{n_p n_q}$, $C \leftarrow (C \cup \{c_{n_r n_p}, c_{n_r n_q}\}) \setminus \{c_{n_p n_q}\}$
    20. actualize errors $E_{n_p} \leftarrow (1 - \alpha) \cdot E_{n_p}$ and $E_{n_q} \leftarrow (1 - \alpha) \cdot E_{n_q}$
    21. set errorvalue $E_{n_r} \leftarrow 1/2 \cdot (E_{n_p} + E_{n_q})$
    foreach $n_i \in N$ **do**
      22. $E_{n_i} \leftarrow (1 - \beta) \cdot E_{n_i}$
    end
  end
until *end criterion reached*

The symbols used in the algorithm.

X:	set of input vectors
$\vec{x_{in}}$:	specific input vector $\vec{x_{in}} \in X$
N:	set of all gas neurons
n_i:	specific neuron $n_i \in N$
$\vec{w_{n_i}}$:	vector which represents the position of neuron n_i
s_1:	winner neuron ($s_1 = \arg\min_{n_i \in N} \|\vec{x_{in}} - \vec{w_{in}}\|$)
s_2:	neighbor of s_1 with the nearest distance to $\vec{x_{in}}$ $s_2 = \arg\min_{n_i \in N \setminus \{s_1\}} \|\vec{x_{in}} - \vec{w_{n_i}}\|$
$c_{n_i n_j}$:	connection between neuron n_i and n_j (obviously $c_{n_i n_j} = c_{n_j n_i}$)
C:	set of all connections
E_{n_i}:	accumulated error of neuron n_i
ϵ_{winner}:	weight adjustment value for winner neuron s_1
$\epsilon_{neighbor}$:	weight adjustment value for all neighbors of neuron s_1
α:	a small factor which predicts how much the error will be decreased when two neurons are splitted
β:	factor which regularly decreases the error
$step$:	count of steps (fetched inputs)
λ:	a constant (each how many steps the net will grow)
$neighbors(n_i)$:	set of all neurons n_j with a connection $c_{n_i n_j} \in C$, or more mathematical $neighbors : n_i \mapsto \{n_j \| c_{n_i n_j} \in C\}$
$age(c_{n_i n_j})$:	current age of the connection $c_{n_i n_j}$
age_{max}:	maximal age before a connection will be removed

Just like the NG, the GNG algorithm can be illustrated quite similar (See figure: 22.3).

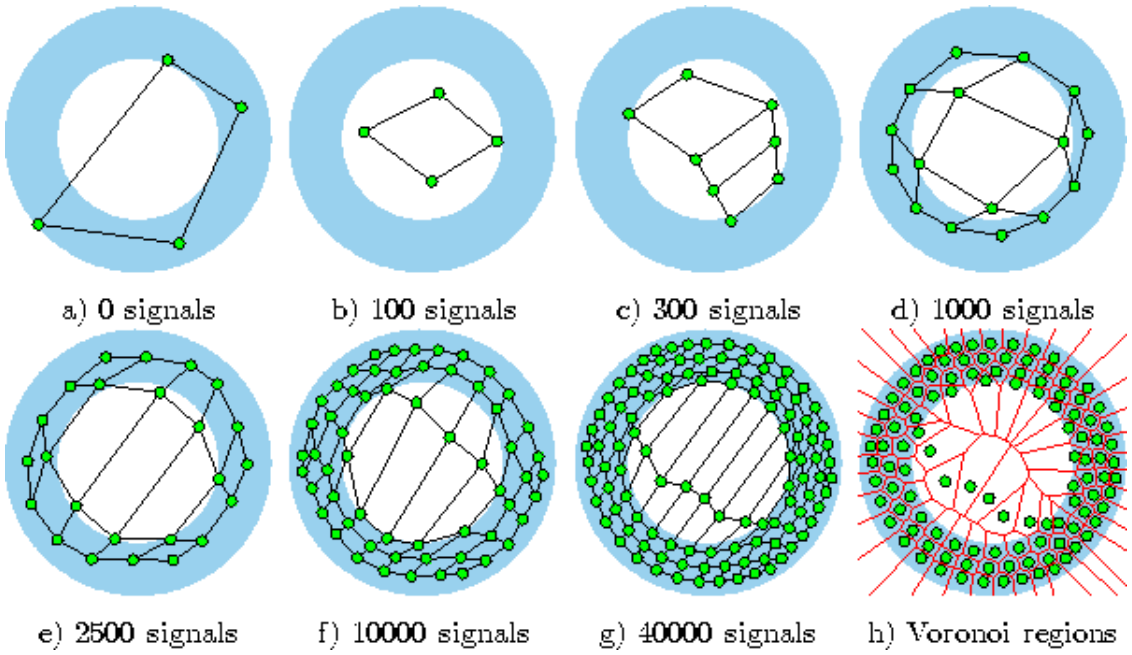


Figure 22.3: Growing Neural Gas (GNG) in action. Data source: Ring ([Fri95])

22.3 Realization

We found that the GNG algorithm is a very powerful and successful cluster algorithm. The GNG algorithm is quite good at keeping long as well as short term data. Especially we found that neurons are created where it is necessary and rather tends to evict neurons which are left alone. This is because the GNG relies on the procedure of connection aging.
However as every other algorithm, it also brings some problems along that we'll briefly need to discuss since the original algorithm suffers from one major issue that we need to circumvent in some way. The problem that the original GNG is facing is the adaption to a non-stationary distribution which would hence also correspond to our non-stationary environment since its basically the event source.

The problem that we encounter is that neurons may get stuck in former regions of high probability density and will eventually die upon changing sources.

Let's have a look at following two situations:

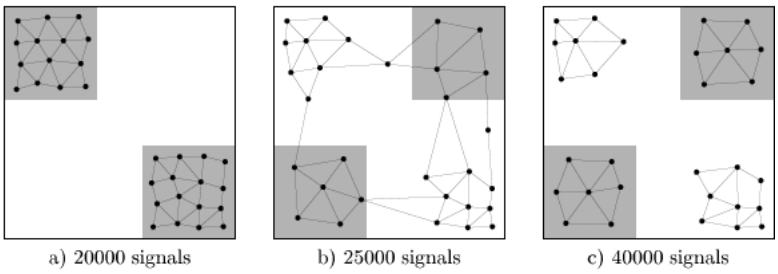


Figure 22.4: A Growing Neural Gas (GNG) network tries and fails to track a non-stationary signal distribution. Used parameters: Max neurons: unknown, λ: 500, age_{max}: 120, ϵ_b=0.05, ϵ_n=0.0006, β = 0.0005 α=unknown, ([Fri97])

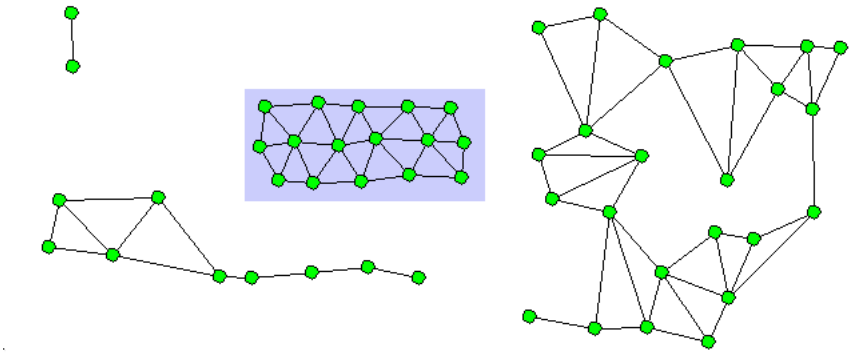


Figure 22.5: Another GNG issue. Used parameters: Max neurons: 50, λ: 600, age_{max}: 88, ϵ_b=0.05, ϵ_n=0.0006, β = 0.0005 α=0.5, ([LF])

Well frankly speaking we may take this a little further though since one might say that such a feature doesn't necessarily allude to be bad for an algorithm to have.
This is true from a long term memory point of view since we actually intend to preserve already learnt knowledge. Also since we've been actually high praising to provide algorithm which should does all that (See chapter: 21).

However since the original GNG **continuously** produces neurons that basically try to minimize the overall quantization error E we apparently need a way to limit the number of neurons to be generated. Upon doing this we notice that the determination of this number is quite vital for this algorithm to do his correct job.

Hence in reference to figure: 22.4 and 22.5 we cannot simply limit the number of neurons to some fixed size since upon changing customs we would herewith also limit the knowledge representation as well. Speaking by example if the number of neurons is to few we might not be able to acquire new knowledge anymore.

Let us briefly address the problems from a rather algorithmic point of view. Basically "Competitive Hebbian Learning" is used to create new connections by always inserting a connection between the neuron n_1 and n_2 nearest and the second-nearest to the current input vector $\vec{x}$. When such a connection already exists, its age is set to zero. Consequently, the age of all connections which are adjacent to n_1 is increased. If the age of a connection surpasses a certain maximum age_{max}, the connection is removed as well as neurons without connections. Hence there is no direct influence to other neurons which are not adjacent. Therefore, their connections do not age anymore which will eventually lead to such a distortion (See figure: 22.4 and 22.5). If we would just leave it hereby we could just forgo the use of clustering at all since outliers will eventually get separated and will steal other neurons portions.

The increasing number of neurons is therefore a problem which we need to solve. Just removing the units with a low accumulated error which signifies that the neuron has barely been used in the past, isn't a practical solutions since the source of a density function may be rather point-like and thus would be inappropriate to delete.
This major problem has also been identified by Bernd Fritzke ([Fri97]). In this paper he addresses the problem that non-stationary data is quite difficult to handle, especially for incremental networks topologies such as the GNG.
Bernd Fritzke thoughts proposed a removal criterion, that introduces a so called utility measure that basically directly computes how much the error for a given input would increase, if the winner would be deleted all the sudden. If this error falls above a certain threshold, the deletion of this neuron wouldn't be a good choice.

Next to this solution, we also applied a different procedure to evict connections and thus also its connected neurons. Instead of performing the aging process by only adjacent neurons, we conducted a continuous aging over all neurons in the topology. Although we must pay the resulting consequences that neurons, which rather hold infrequent environmental data will eventually be evicted. Evidentially the right balance is quite hard to find and will be always be an issue. However, a good long-term solution we should be solving with other algorithms, such as the g-means, which rather tends to keep old and also infrequent data, but with the little drawback of being a little lazier then the GNG.

Similar to the ANN g-means we also applied a neural network over the clusters:

Result: trained ANN filtered with help of a growing neural gas
input: A set of continuous input-pairs $\Psi\{\vec{x}, y\}, y \in [0; 1]$
repeat
  fetch next input $in_t \leftarrow \{\vec{x}_t, y_t\}$
  if *somebody present* **then**
    put $in_t \rightarrow traininqueue$
    $counter \leftarrow counter + 1$
    if $counter \geq trainingsize$ **then**
      add all input-pairs of the trainingsqueue into the growing neural gas
      train ANN with all extant input-pairs of the neural gas
      $counter \leftarrow 0$
    end
  end
  $t \leftarrow t + 1$
until *Stop condition reached*

Like any other algorithm we implemented and tested our own version of the GNG in a small little test application. First we give the dead unit problem and then our solution. Note that the stages have been numbered in increasing order as well as their corresponding number of signals.

First we illustrate the dead neuron problem and then our solution as explained, to it.

A Growing Neural Gas (GNG) network tries and ***fails*** to track a non-stationary signal distribution. Used parameters: Max neurons: 30, λ: 500, age_{max}: 250, ϵ_b=0.05, ϵ_n=0.0006, β = 0.00001 α=0.5)

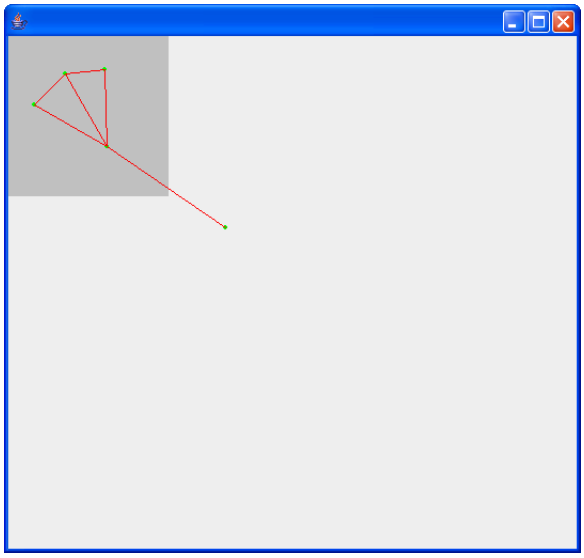


Figure 22.6: Stage: 1

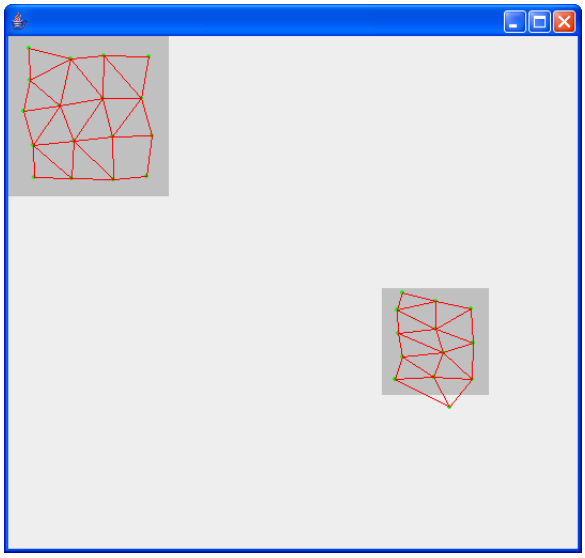


Figure 22.7: Stage: 2

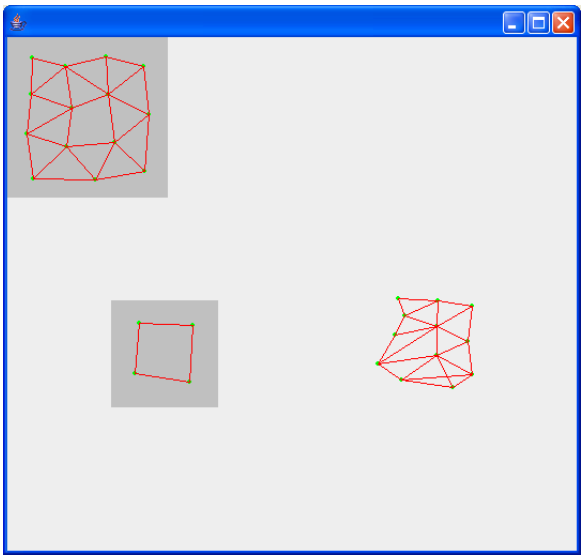


Figure 22.8: Stage: 3

A Growing Neural Gas (GNG) network tries and ***succeeds*** to track a non-stationary signal distribution. Used parameters: Max neurons: 30, λ: 200, age_{max}: 250, ϵ_b=0.05, ϵ_n=0.0006, β = 0.00001 α=0.5)

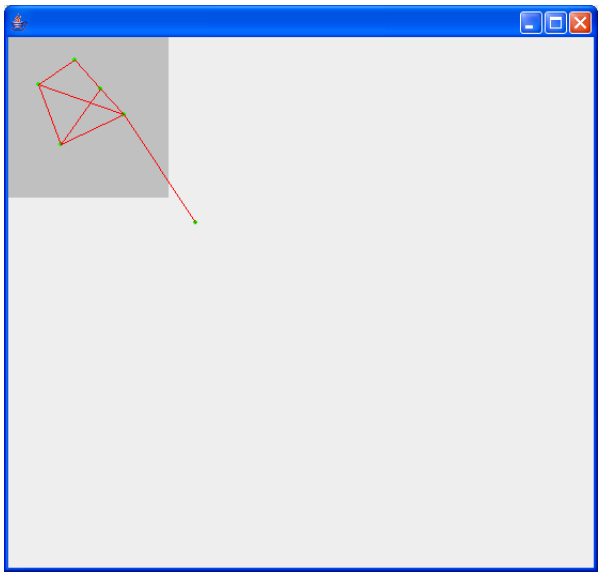


Figure 22.9: Stage 1, GNG

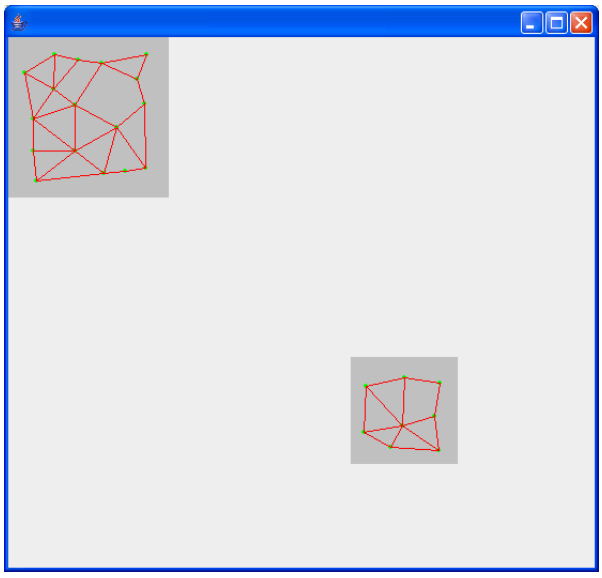


Figure 22.10: Stage 2, GNG

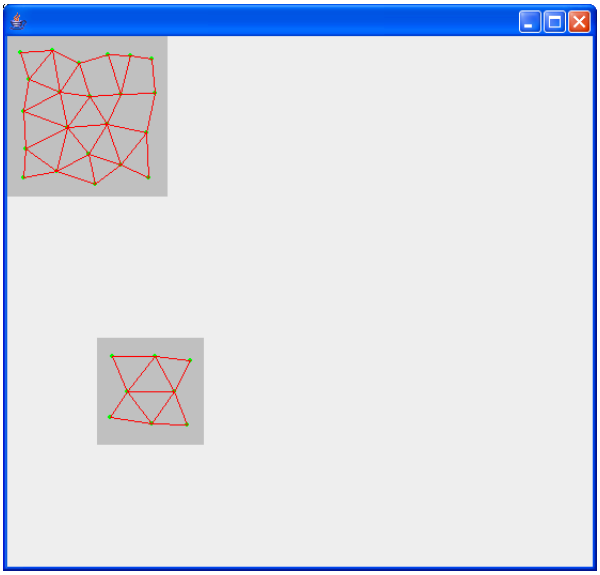


Figure 22.11: Stage 3, GNG

22.4 Evaluation and Discussion

To tell the truth continuously altering the connections wasn't a good solution at least judged by evaluating both solutions in the test-bed. However just like the STM-LTM solution we've already started to test the "less efficient" version of the GNG within real environments. The neuron evicting solution proposed by Bernd Fritzke ([Fri97]) that according to section: 22.3, introduced an utility measure, was producing better results in the test-bed (See figure: 22.12).

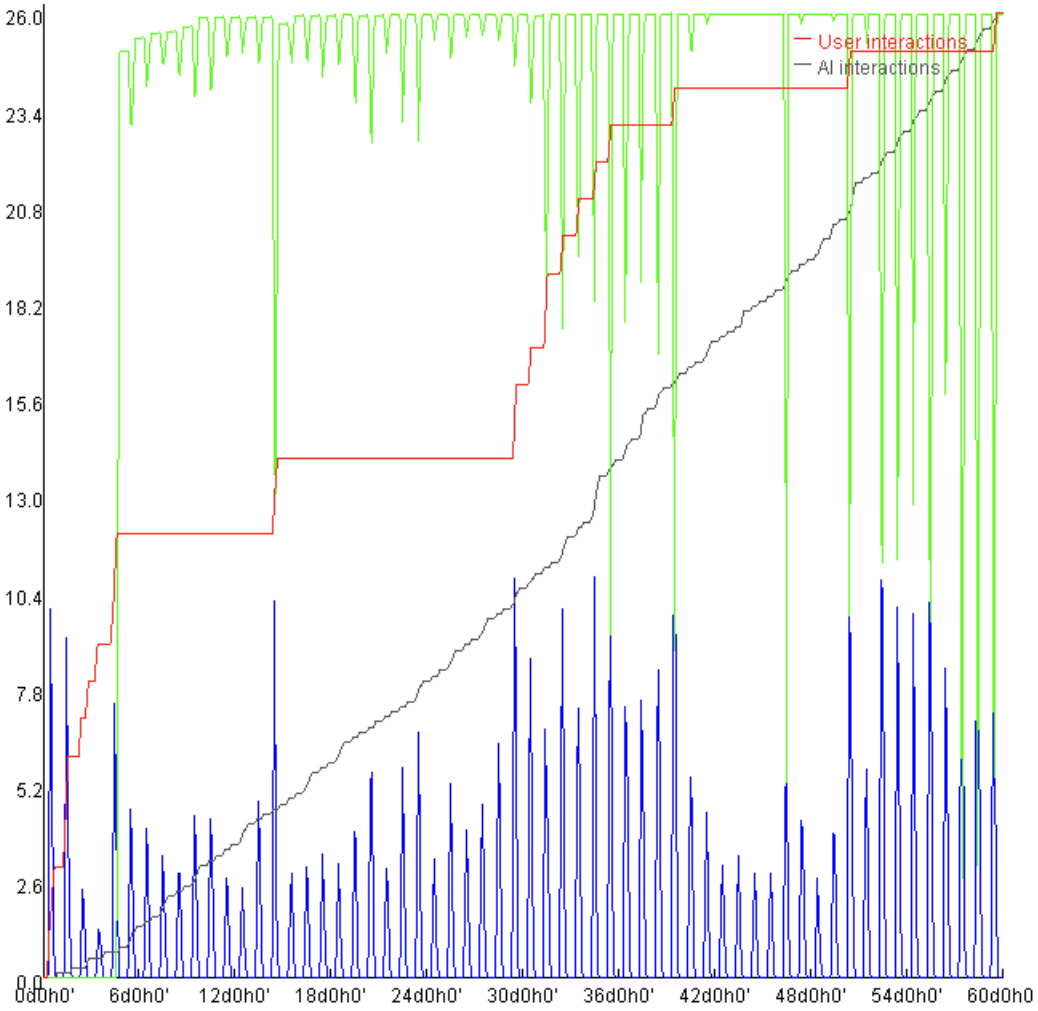


Figure 22.12: Neuron eviction by applying an utility measure. Clear predictions are taken. Half of the inputs even unforced. However it is hard to say how the algorithm will behave in a real environment since humidity and temperature are basically neglected in the entire decision process. Ordinate: User interactions

Although we can't really prove that clustering enhances the prediction with the usage of this simulator that much (See section: 17.1.2), however we can still clearly outline one of the most powerful strengths of the ANN-GNG (Artificial Neural Network Growing Neural Gas) algorithm. The incredible ability to adapt to new user desires. In particular by applying our own removal criterion, where neurons are continuously aged within the entire network in each epoch, achieved impressive results. We can observe the results by conducting the second test that considers changing users (See figure: 22.13).

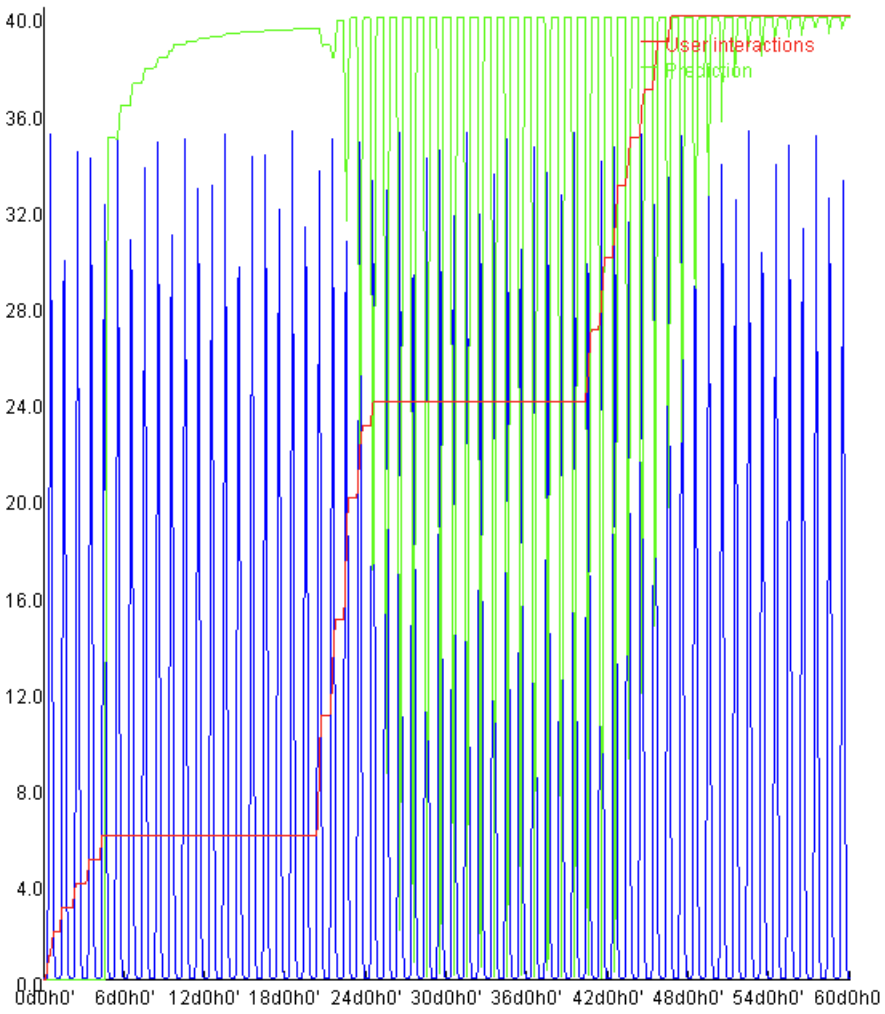


Figure 22.13: Especially we notice that neurons are created in spots, where the occupant fees comfortable. Changing user preferences involves, (next to different electrical lighting behavior) also a different working hours to be adhered. The GNG strengths hereby underlies to seek out for such changes within an environment and subsequently starts to evict misbehaving environmental data (neurons). Hereby we gain a very interesting result that maybe incorporated with other algorithms. Ordinate: User interactions

In general though when applying a pure GNG solution, we might rather consider the other neuron eviction strategy, which although is a bit lazier but more has the advantage of being more accurate (See figure: 22.14). However such results must first be analyzed and enhanced quite a bit before starting to state any premature decision which of the two would perform better in real environments. Generally all algorithms should be improved and optimized by using the IB Framework (See chapter: 12) which advances such variations to be tested and measured. In particular the cluster algorithms need to be investigated by far more then just within the perimeter of this thesis.

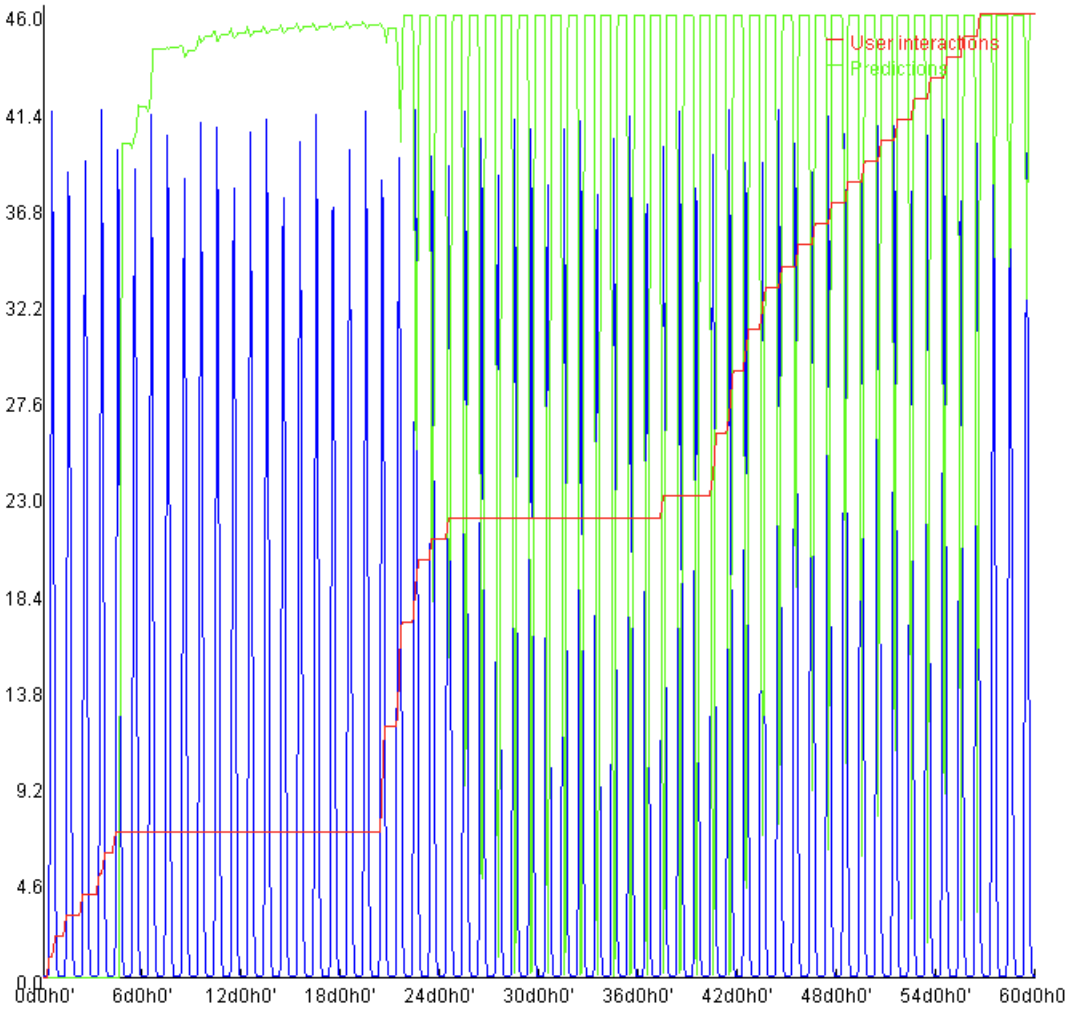


Figure 22.14: Generally speaking still good results since we notice that the bad results in the last part have been caused due to similar problems already recognized and explained in figure: 19.16 (STM-LTM ANN). Ordinate: User interactions

Part VII

Results and Discussions

Chapter 23

Introduction

In this part we will illustrate the gained results as far as it was possible for us to either record or evaluate due to the limited time we had to conduct this thesis.

Next to the overall results which mostly have been performed by a real-time simulation, we've been monitoring 17 days in 3 different real environments.
It should be noted that individual algorithms have already been tested in the latter part and thus not shown anymore. However according to the IB Framework we've been selecting a couple algorithms among the presented ones, to gain an anytime overall prediction by competing them against each other.

We recall that our major task was to explore new different learning approaches for controlling building effectors and not to perfectly optimize them to perform at their best by adjusting all parameters to their major possible extent.
The material of this thesis is hence to illustrate possible ways to learn dynamic space behaviors by invoking different learning algorithms. Each of which hereby possesses its individual strengths and weaknesses that will ultimately control its environment by all means possible.

23.1 Test settings

Following test settings have been applied for real (See chapter: 25) and simulated environments (See chapter: 24) tests.

- All predictions have been held back for 3 days (*creep time*) in order to collect some initial knowledge before starting to perform any possible predicted actions (Except switching off the lights upon losing track of a room's presence).
- After each conducted action (user or IB) a security delay of 1 hour is being held (With the exception that the action has been caused due losing presence also).
- The presence is filtered using a window of 10 minutes (Filtered Presence).
- Framework frequency: 1 minute in the simulation and 10 minutes in real tests.

Additionally it may be worthwhile to mention that all real tests have been started on the 25^{th} of November, 12.15 noon. This has not been planed or anything like that but rather was the cause of a major network breakdown that has been affecting Internet access as well local area network access completely. On top of that, the LON router needed to be restarted due to an internal software crash and alone would have been a sufficient reason to entirely re-start all tests again.

When not stated otherwise, following colors will apply for the graphs in both tests.

Red: User interactions
Dark Grey: AI/IB interactions
Blue: *Real:* logarithmically scaled interior daylight, *Simulator:* normal interior daylight
Green: AI/IB predictions
Light Blue: Filtered Presence status
Pink: Light status (on,off)

23.2 IBF Parameters

It may be appropriate to give a short overview about the used parameters in the IBF algorithm. Each prediction algorithm parameter though, we will not highlight due to their inessentiality. However the IBF algorithm impacts the entire learning process in the whole and thus may be worth to explain.
The meaning and the impact of each parameter should be looked up in the corresponding chapter (See chapter: 12.2)

α_{Max}: Simulator: 0.01, Real: 0.1 $(\alpha_{Max} = \frac{1}{framework frequency \cdot 100 \cdot 60})$
t_{hl}: The time where $\frac{\alpha_{Max}}{2}$ is reached has been set to 20 minutes

Chapter 24

Results in a Simulated Environment

Next to the results presented in the learning part (See part: VI), we will now reveal how a set of distinct algorithms have been performing in the latter defined test-sets (See chapter: 17). We've been considering following algorithms

- Regular Backpropagation Artificial Neural Networks (See chapter: 19)
- LTM-STM Network decomposition with no user input control (See figure: 19.6)
- Statistical Learning with a dynamic determined α (See chapter: 18)
- Self Organizing Maps (both Gaussian distribution and cone function) (See chapter: 20.1)
- ANN-GNG (Artificial Neural Network Growing Neural Gas), global regression (See chapter: 22)
- ANN-GMeans (Artificial Neural Network GMeans), 2 versions (global and local regression) (See chapter: 21)

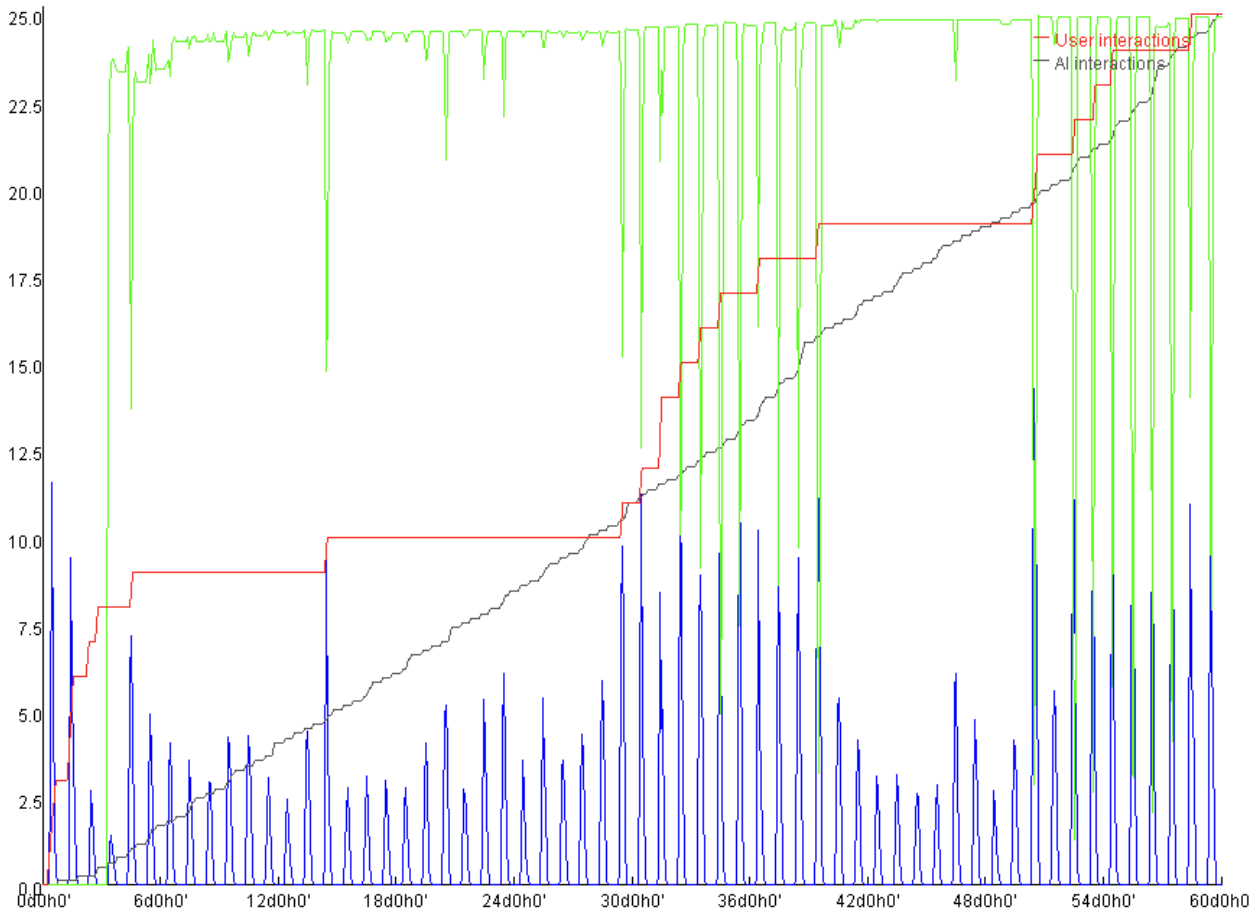


Figure 24.1: As expected the overall predictions turned out to be quite good. Of course upon competing all algorithms against each other, the overall decisions making gets a bit lazier but is in general still very successful. Ordinate: User interactions

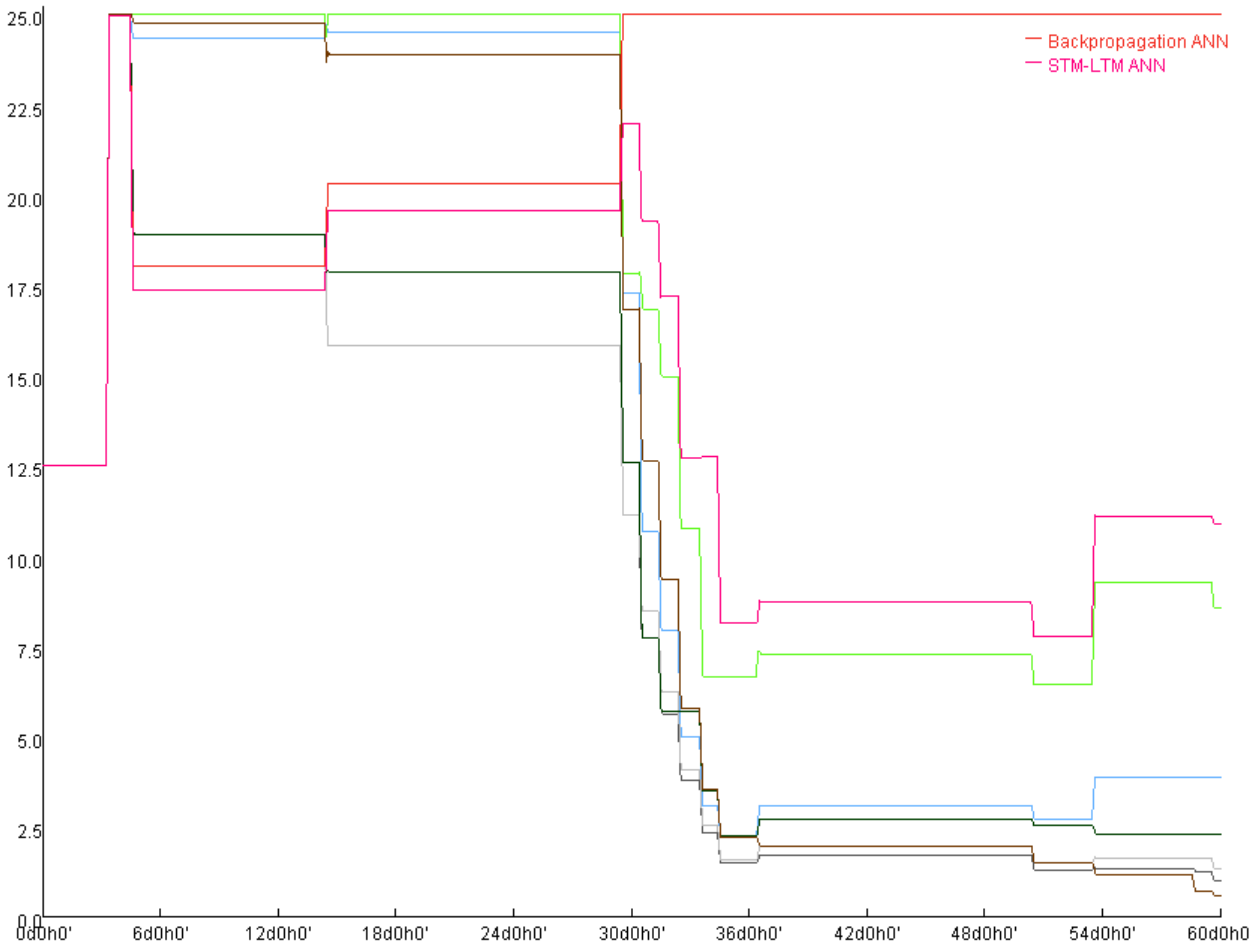


Figure 24.2: Since the IBF algorithm can keep track of each different algorithm and within every single prediction, we can clearly depict that the regular backpropagation network was the overall winner within the first test simulation. Hence, our previously conducted tests in the learning part (See: VI), have herewith been proven to succeed within the overall decision making also. Ordinate: User interactions

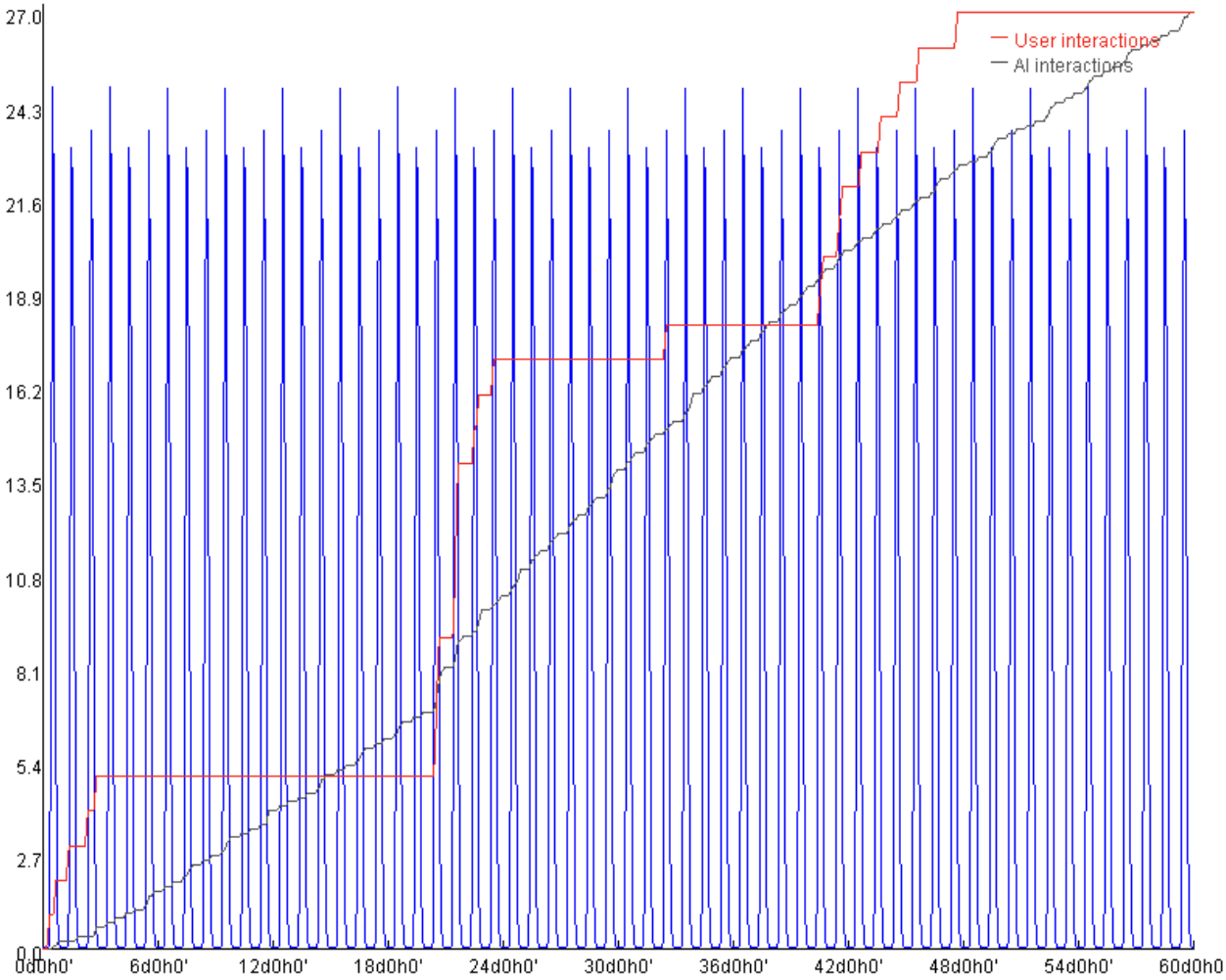


Figure 24.3: Astonishingly we can depict that the IBF algorithm can react to changing customs quite agile. However we see that some "opposing" actions have been taken by the decision controller, that thoughtfully needs to be investigated by zooming into those particular days (i.e. 20 and 21, see figure: 24.4). Ordinate: User interactions.

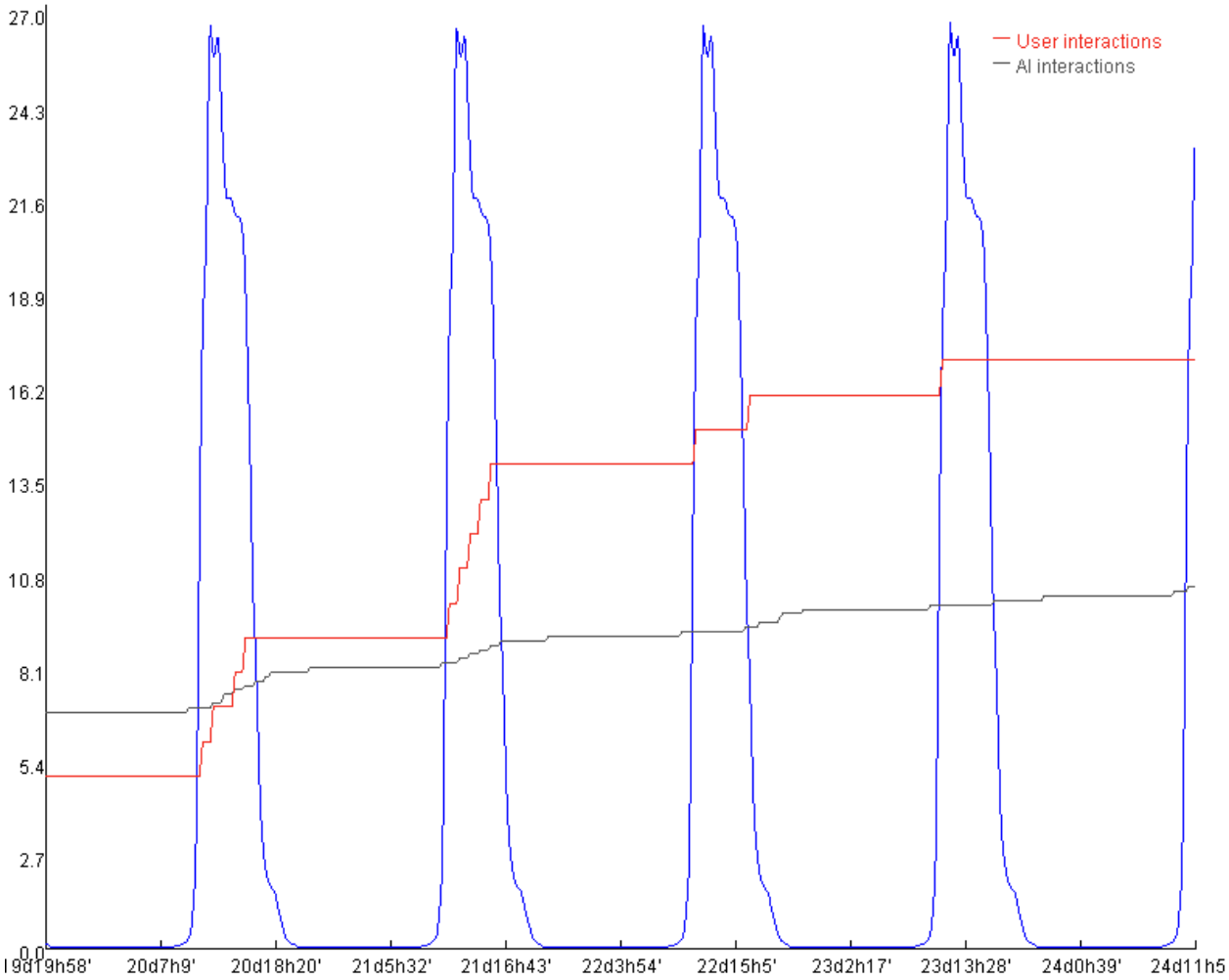


Figure 24.4: As annotated in the previous figure we can identify that in day 20 and 21 a couple of inputs have been conducted which might also occur in reality. This is due to the fact that new user preferences needed to be learnt. However the decision controller was a couple of times mis-predicting user desires and thereby supposable would also start to annoy its occupants. Ordinate: User interactions.

When concluding the first performed and evaluated tests we can still depict that some improvements on the entire decision making process are necessary. Improvements, which can positively impact user comfort. Back in the very beginning of this thesis we remarked one of the most important goals that any IB is suppose to achieve. -**Not disturb occupants**.
However in reference to figure: 24.4, we can extract that such results actually violate this statement in a major extent. We recall that disturbing occupants involve:

> *User wishes are upper priority. For instance: Blind motor noises, flashing or flickering lights, etc. should be minimized. User desires must be manually configurable and should not end up into opposing actions.*

In order to lower such incidents in the near future within the simulation and in particular in real environments as well, we propose a very intuitive and simple solution to this problem.
To illustrate this issue, we give a brief example that clearly outlines what measures need to be taken in order to comply to this specific stated goal.
Whenever occupants cause effectors to change their state, any particular associated device agent that controls and supervises a corresponding effector should basically wait to perform any actions until the predictions start to conjoin to occupant interests again. Thereby we would circumvent the problem that actions are

being performed which actually wouldn't have been appropriate to be conducted in the first place. Due to the fact that we assume that most of the algorithms that were wrong in their predictions would still not be able to recover from such a major change.
Additionally to avoid temporary mis-prediction caused by short lasting but correctly stated spikes, we can further specify how long a "re-prediction" needs to be, that will subsequently start to re-participate into the normal decision making process. Thus we can get rude of the introduced *security delay* that we, according to section: 14.6, also identified to be one of the weaknesses.
With this attempt we belief that such disturbing actions will be avoided at all since the shared opinion is demanded hereby and not any possible governing or autonomous prescription. We might need to reconsider this solution when more attention needs to be paid to energy savings. For instance we might need to incorporate some sort of teaching unit that basically forces occupants to think more ecological.

Accordingly, we applied the presented solution in the test-bed again, that we simulated using the standard tests prescribed in section: 17.1.2. It might be worth to mention that this alterations do not affect the implementation on any prediction algorithms directly since such modifications do only concern the IBF algorithm and therefore has been accomplished right there.

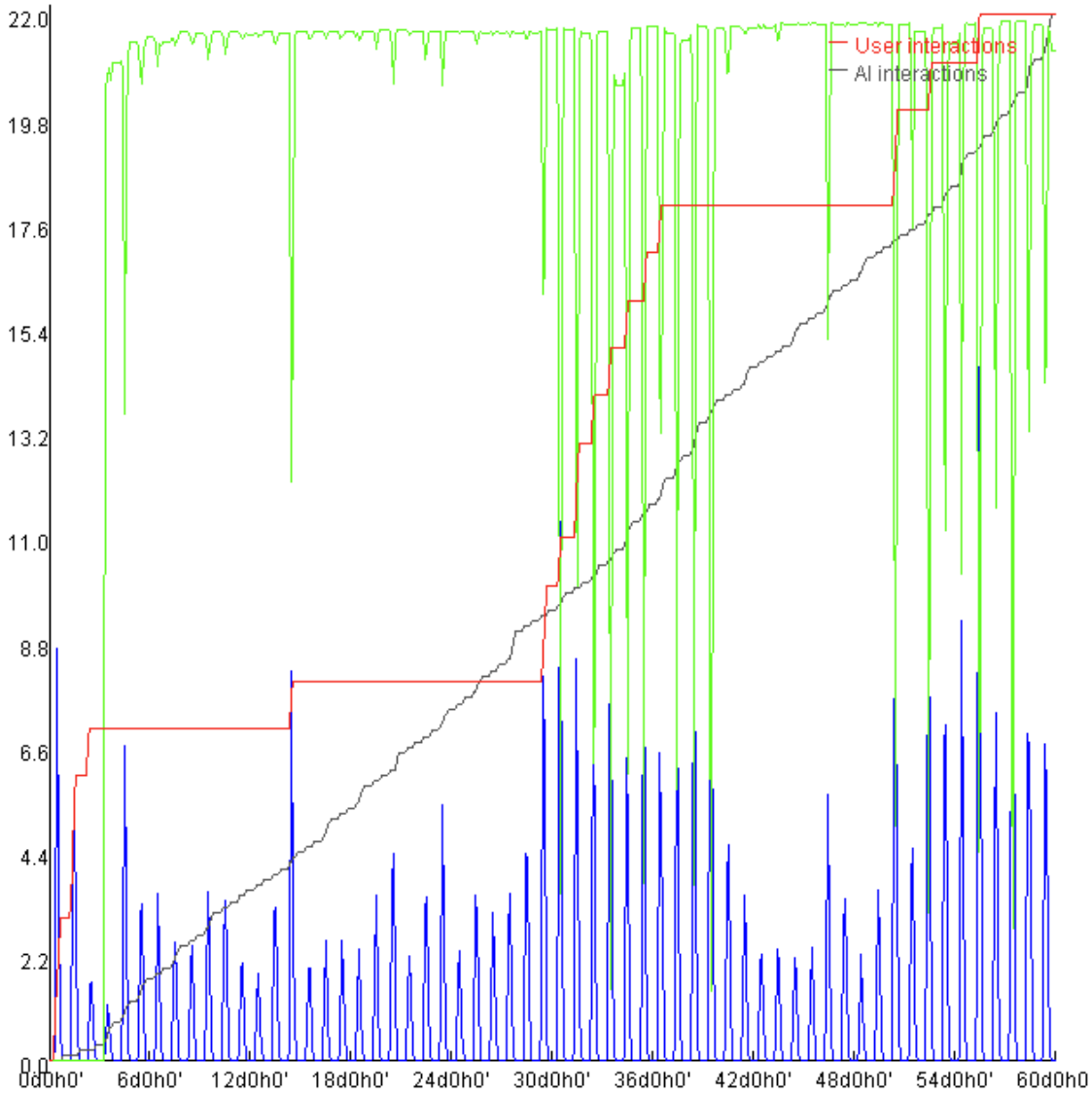


Figure 24.5: Although we cannot see any significant improvements at first glance, there will be a vital big difference that revealed us things that we never would have thought to get improved in the entire decision making for each single algorithm. We slightly also notice this by the lower amount of user interactions due to the removal of annoying opposing IB interventions around day 30 and 35. Ordinate: User interactions.

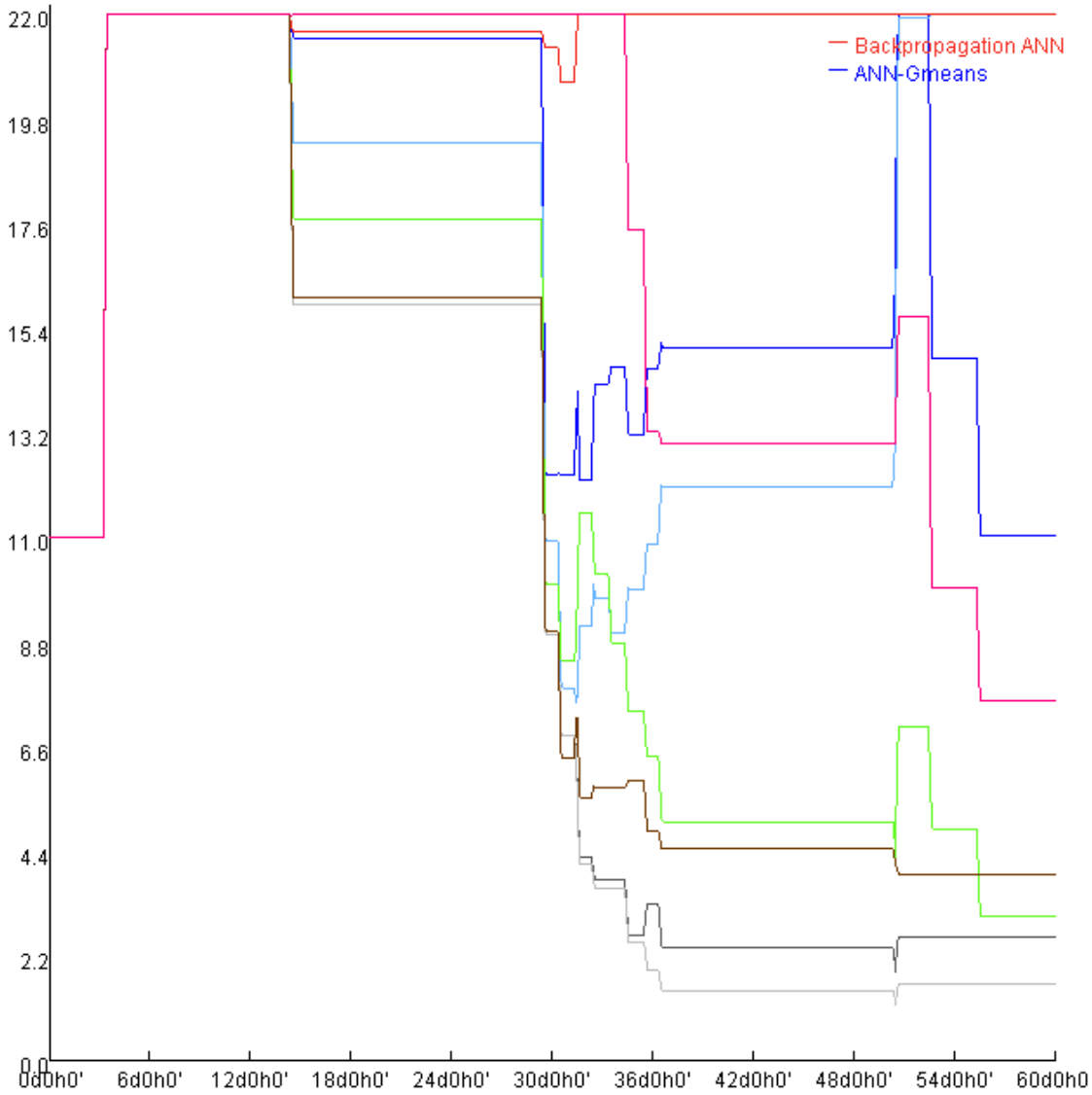


Figure 24.6: We further notice that some of the algorithms achieved better predictions then before. For instance the ANN-GMeans was pretty good in predicting user desires. Ordinate: User interactions.

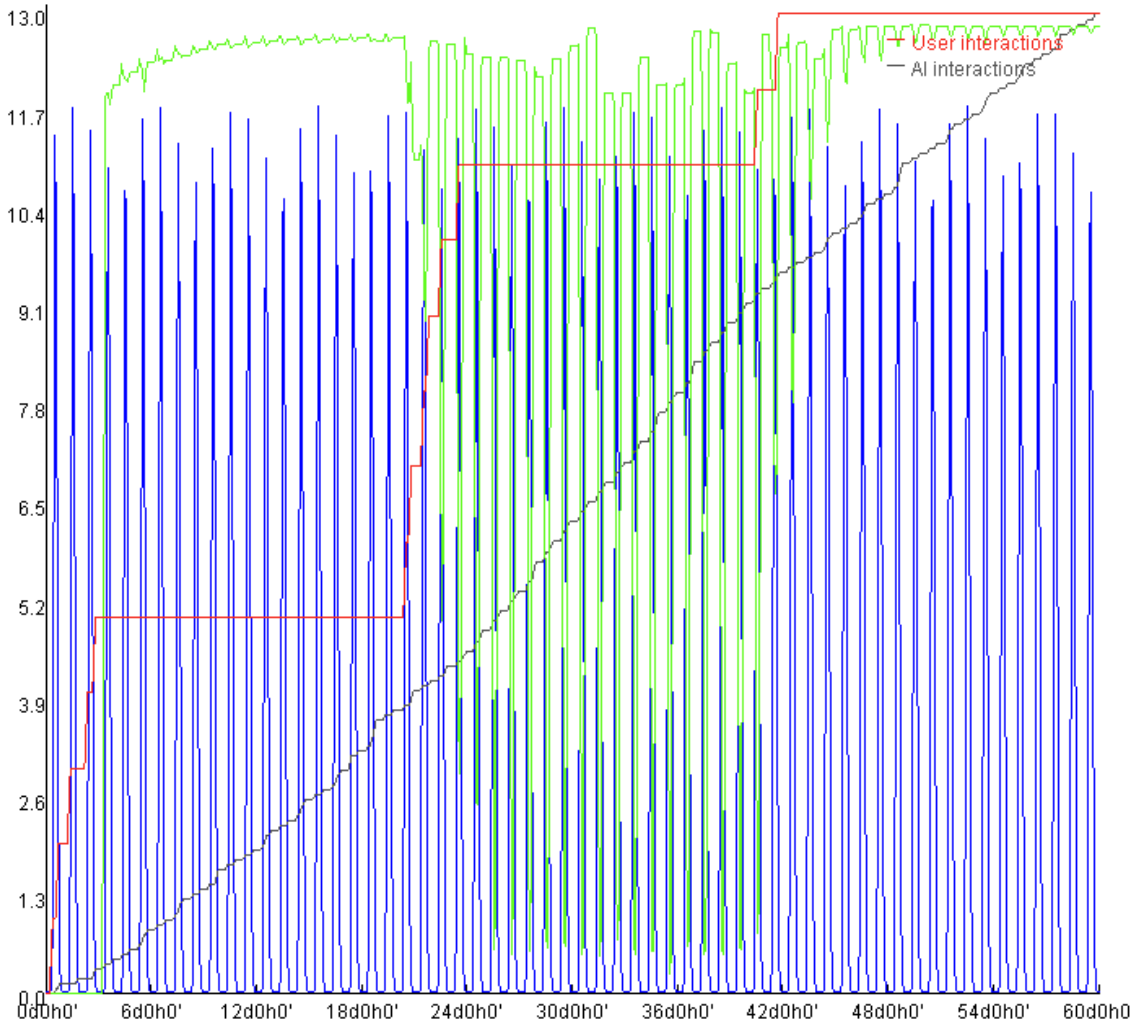


Figure 24.7: The last figure: 24.5, already showed us a good fraction of enhancements. However within this graph one can identify that lots of unnecessary user interactions could have been prevented upon applying the suggested approach above. Furthermore we can denote that the entire IBF algorithm is performing greatly. Greatly, since besides the IBF algorithm, most prediction algorithms where reacting quite agile to the sudden user behavior changes as well.

We would now like to demonstrate why this "minor" change did impact the entire decision making of the entire framework in a very positive way that much.
For this reason one might to have a look at latter scored results back in the latter presented figures (See figure: 24.1, 24.2, 24.3 and 24.4) again.

The major problem caused by most of the presented algorithms were that upon receiving user interactions, the decision making was (according to section: 14.6) held up for one hour in order to circumvent sudden opposing actions to be conducted by the decision controller.
However in most common cases the overall decision will not distinguishably have changed within this short period of time and thus probably result in an "opposing" action to be executed again. The consequence that needs to be paid upon reacting to such scenarios in such a way is that every prediction algorithm will then start to be trained with its own wrong environmental sensory as well as corresponding effector data again. Thereby the entire decision making will be slowed down as well as wrong predictions are being produced.
In real environments we've been observing similar happenings as we'll see later. Hence, we can conclude that the introduced calculated *user satisfaction level* within the simulator isn't that much of absurd and has

been modeled quite realistic. Realistic in the sense that the *user satisfaction level* will start to increase when the synthetic occupant starts to get uncomfortable with the created fictional environment. Subsequently upon each manual user interference, the *user satisfaction level* is reset since the satisfaction level has been reached.

Chapter 25

Results in real Environments

Due to the pressure of time, we had to pre-select a couple of algorithms that we started to test within three different real environments while we were still improving some of them. Not to mention that the latter introduced advancement has not been considered in our measurements since we've been discovering it a couple of days after having already started the real tests.

Nevertheless recording and observing real environments were very important and gave us insights in a before undiscovered problem. Before discussing the problem, we'will first illustrate the test results which have, as mentioned, been gathered from three different environments. Each of the rooms differ in their size, shape, number of occupants as well as in their own kind (i.e. laboratory or regular office).

Again, we would like to point out that the results shown here, only give first results and therefore in the current condition, does not correspond to a normal future operation at all since the latter discussed improvement, as well as general algorithm optimizations have not been applied. Hence we'll be restarting the learning by the beginning of January again, by considering all factors that were identified.

It may be worthwhile to mention that low user inputs, indeed gives a vital information on how good device agents have been performing, but doesn't actually really tell us anything about the user comfort.

Additionally we will forgo to illustrate any figures about the performance of each algorithm due to the limited time of recording.

25.1 Room 55.G.84

25.1.1 Corridor Light

According to the conducted environmental analysis (See section: 11.3), we stated this room as a *Simple Environment*. Hence we presume that our measurements should more or less confirm the latter gained analysis results.

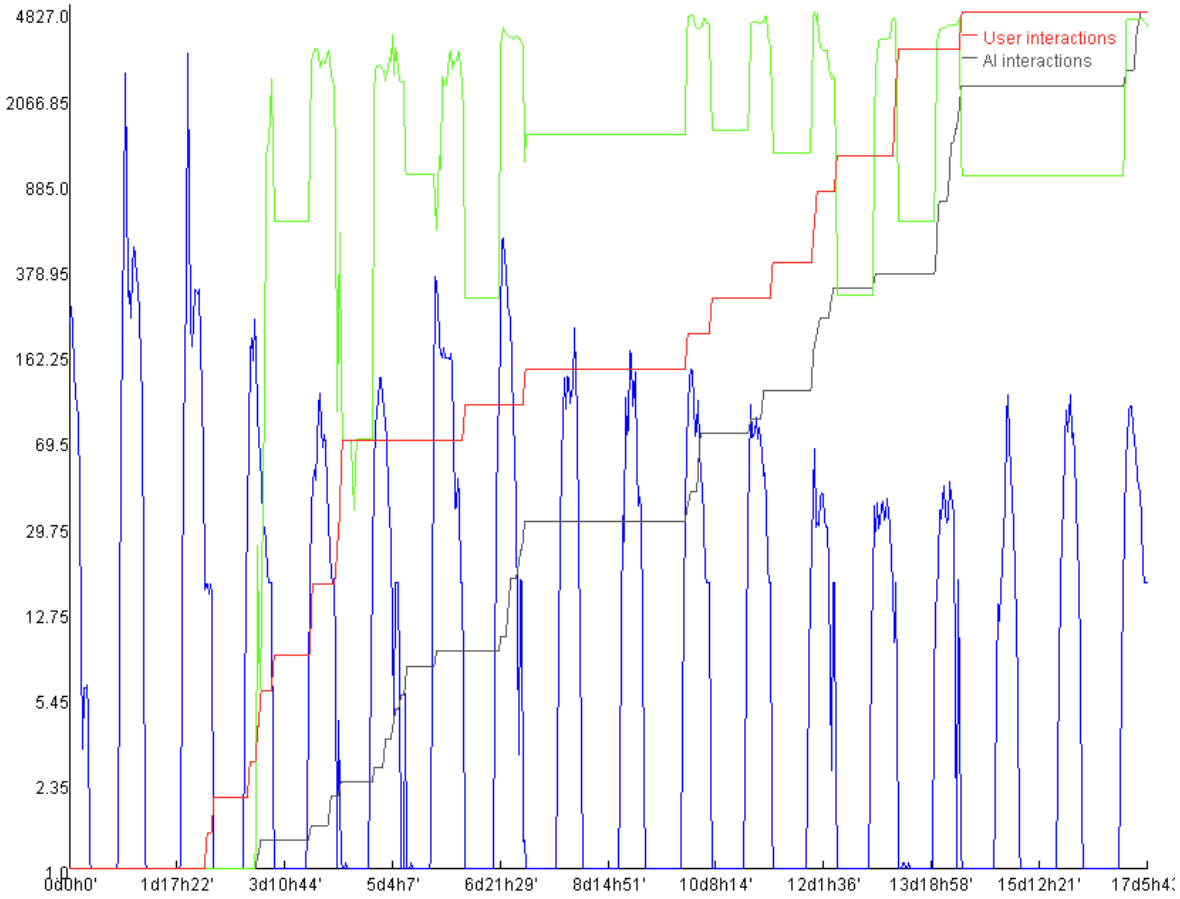


Figure 25.1: Generally our initial hypothesis turns out to be just as we expected. Namely, that the corridor light was predicted to be on all the time. However we notice that in the early starting phase, a couple of problems have occurred that we would like to illustrate and discuss a bit closer (See figure: 25.3).
Ordinate: Logarithmically scaled interior daylight value.

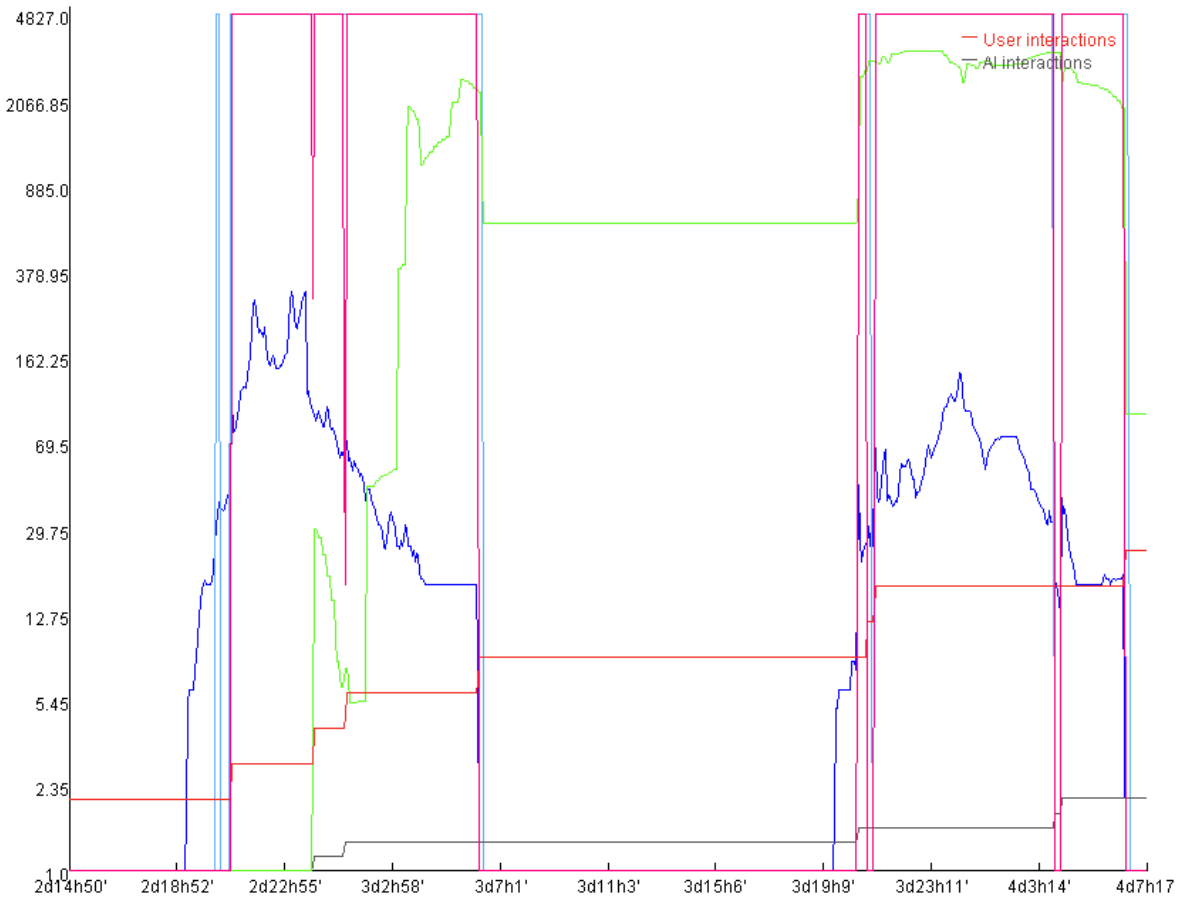


Figure 25.2: We notice that on day 3 (beginning of the predictions), a row of user inputs have consequentially been initiated by occupants due to that the corresponding light device controller subsequently started to switched off the lights again as well (faintly visible by tiny spikes).
However in the next day we can perceive that the device agent has already learnt from the previous mistakes.
Ordinate: Logarithmically scaled interior daylight value

We actually wondered why such mis-predictions could have happened in the first place since normally the AI was suppose to figure out the trends within those 3 days. Unfortunately due to the inappropriate IB starting date (Friday), we evidentially haven't been able to collect enough sufficient data that would reflect user needs (i.e. Light habits). Nevertheless the prediction shouldn't have turned out that bad anyway since a couple of useful inputs were still given up to this time.
The next figure (figure: 25.3) will uncover another problem that was totally new to us.

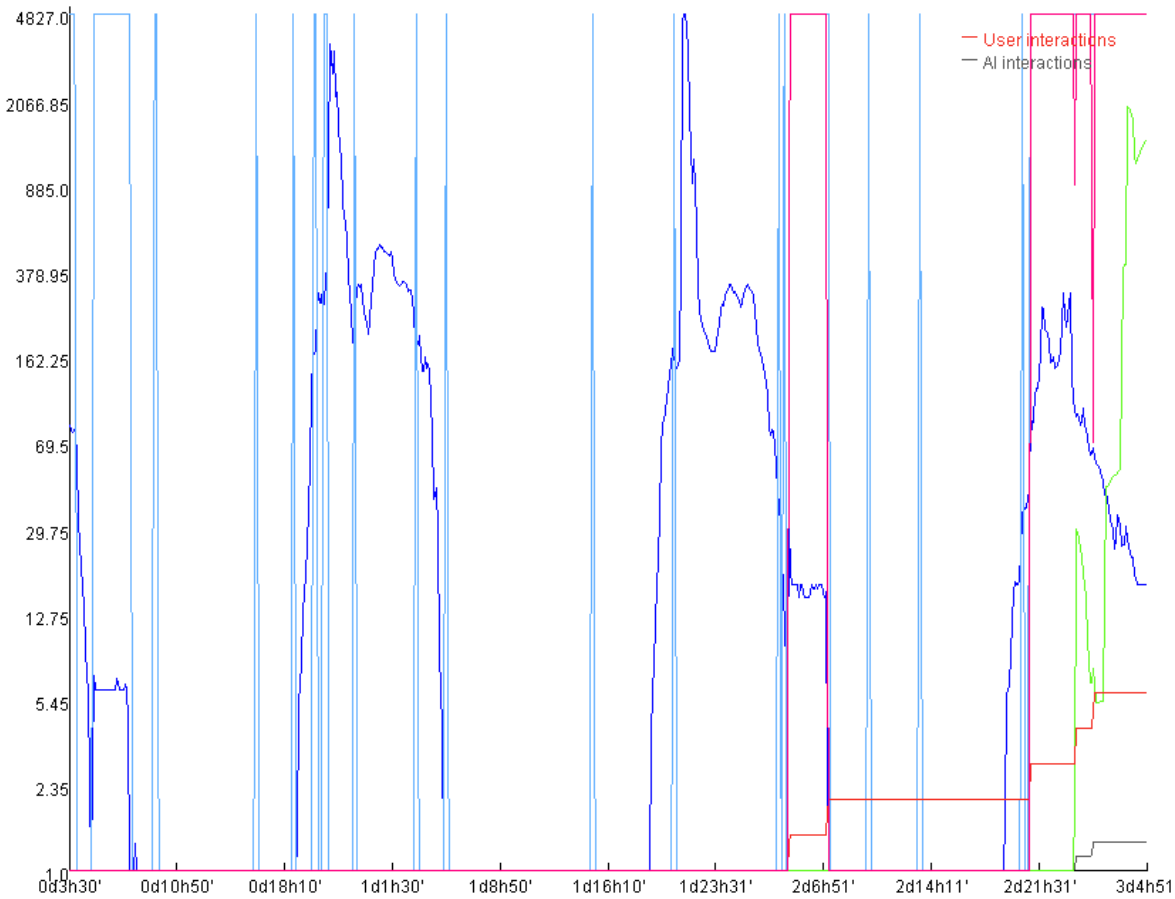


Figure 25.3: We see that almost throughout the entire weekend some kind of ghost was present that was teaching any AI algorithm to switch off the lights within those periods (depicted by the light blue curve, spikes of 10 minutes each).
More precise investigations have then shown that remote connections through ssh or vpn caused certain thin PC Presence client applications (See chapter: 8) to announce presence. However as depicted in the latter figure the AI was able to get accustomed to "new" user desires upon each interaction. We'll be discussing this issue in the future work part quite a bit more (See chapter: 27) and leave it hereby.
Ordinate: Logarithmically scaled interior daylight value

25.1.2 Window Light

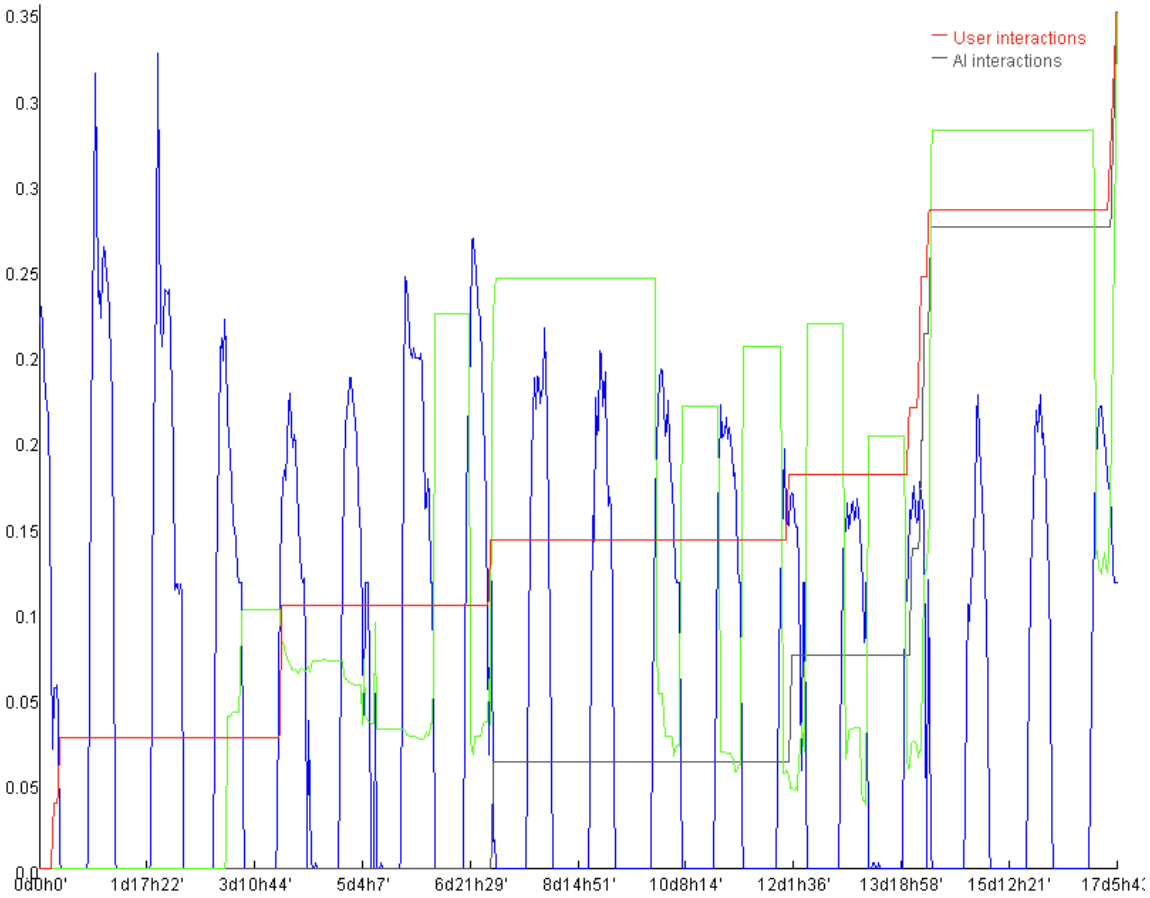


Figure 25.4: Unlike the corridor light (latter section) we expect the window light to be off most of the time. From a general point of view actually true also. However we also notice that the window light was suppose to be switched on all the sudden [1]. The acclimatization to new habits happens to work (although would be a bit faster by considering the upper solution (See chapter: 24)). However re-learn phases bring lots of unnecessary annoying user inputs along (day 16 and 17) which would also be solved by incorporating the new solution.
Ordinate: Prediction

[1] One of the three occupants (Tobi Delbruck) has left for a longer trip to the US.

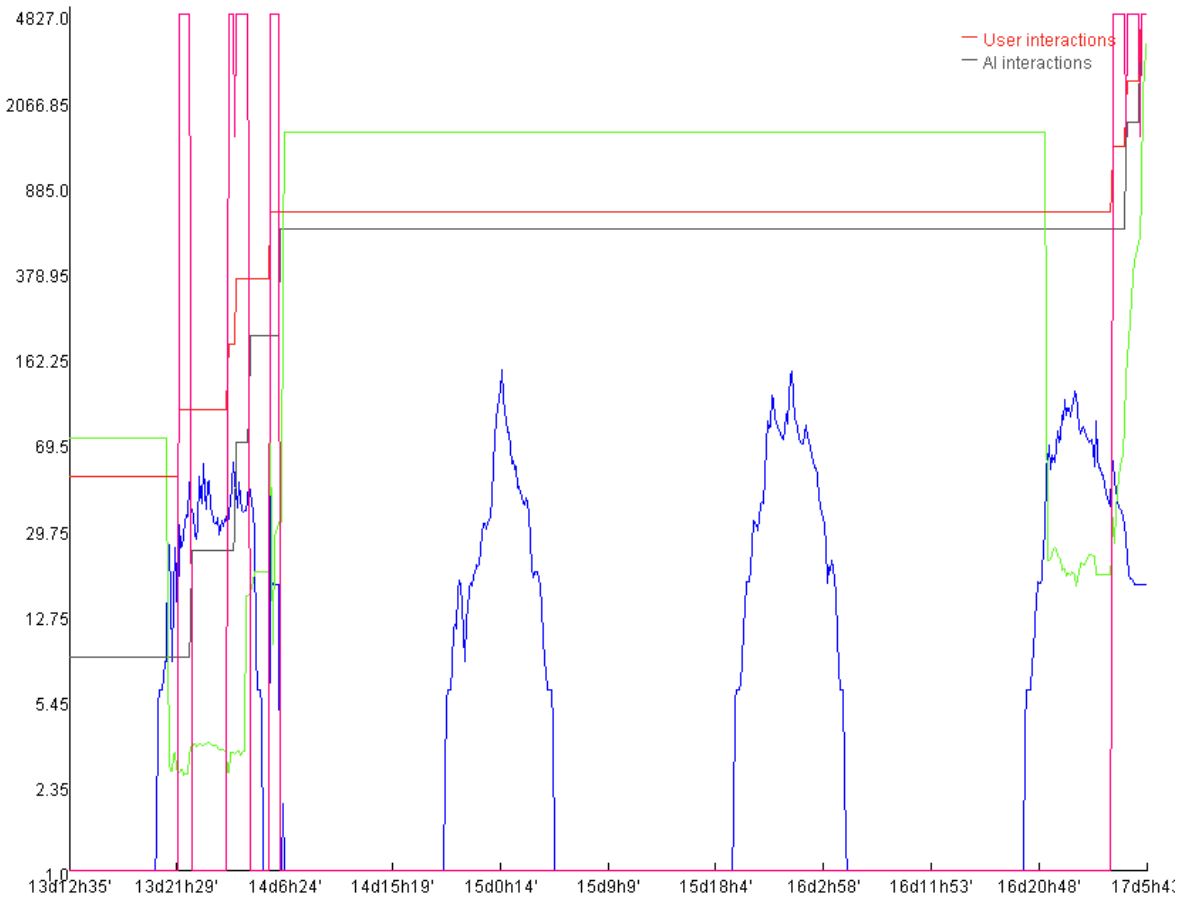


Figure 25.5: A zoomed view that illustrates the *opposing action problem* a bit better.
Ordinate: Logarithmically scaled interior daylight value

25.2 Room 55.G.26

We already stated that the number of user inputs isn't the only measure to look out for since as we'll see now, not all users are lazy. Hereby the term lazy needs to be extended by taking the location of the manual switches also into account since quite often we've been observing that certain race conditions have occurred where both (user and IB) were trying to switch on the light concurrently. However it happens to be that the user in this room is almost always faster (in both lights).

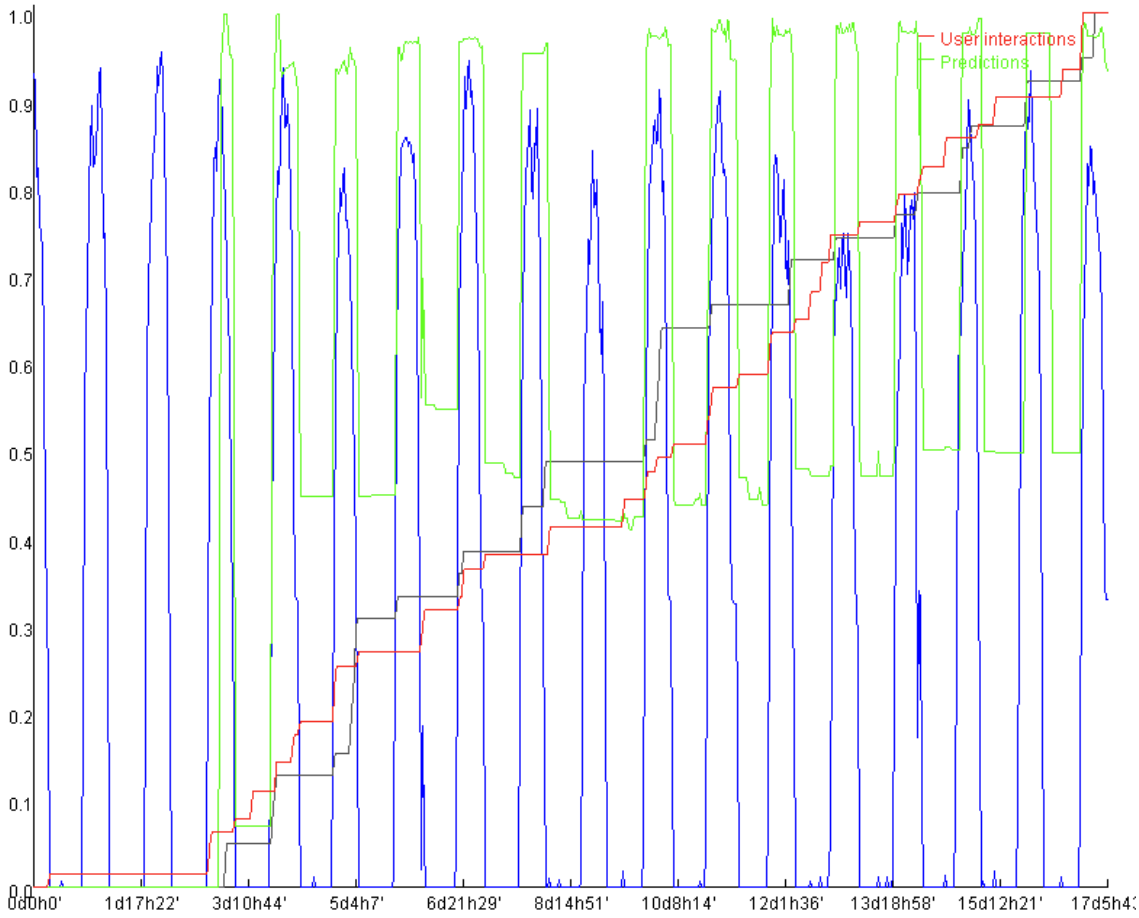


Figure 25.6: We only see here one of the two lights since they're almost identical due to the fact that they obviously have been used together all the time. The predictions where excellent except that the occupants were either faster or the device agent couldn't conduct certain changes due to waiting up security delays. Ordinate: Prediction

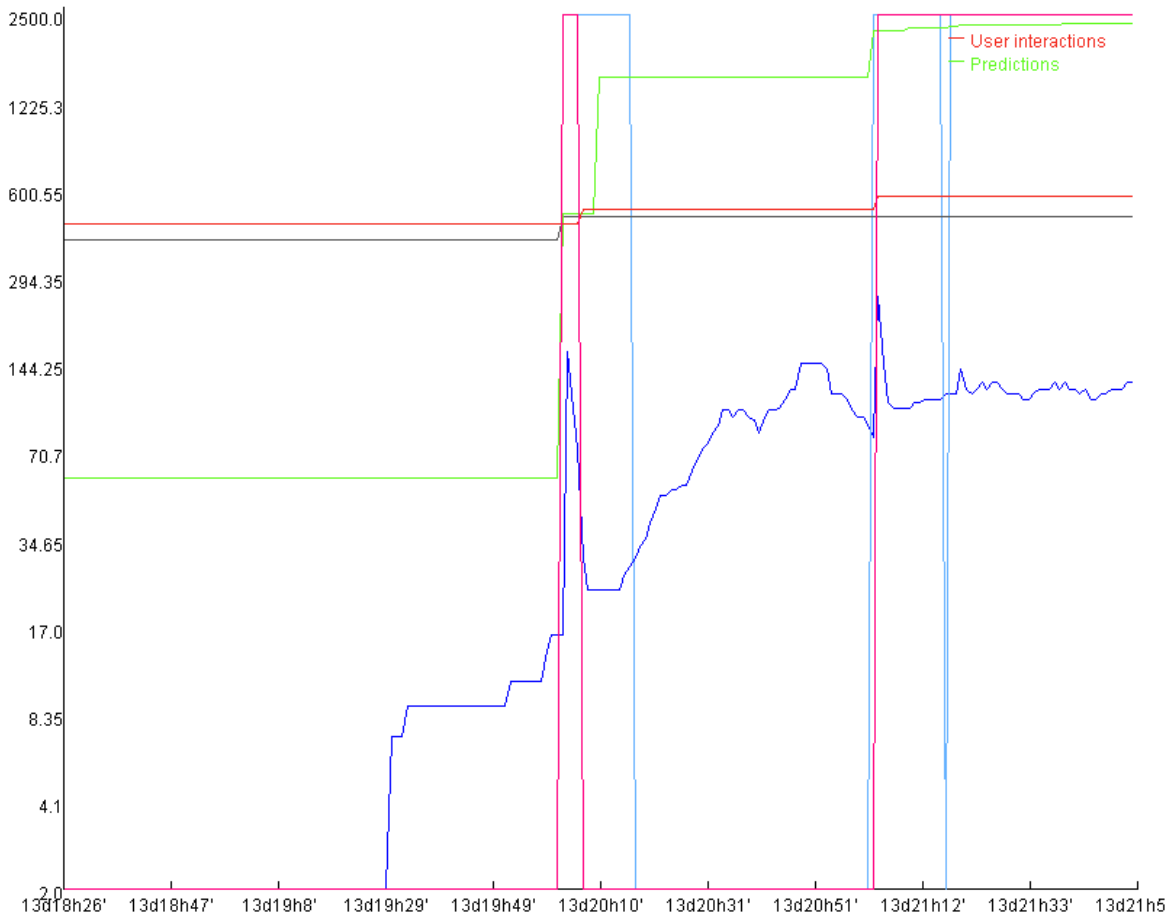


Figure 25.7: This is one of the situations where all occupants have left the office and manually turned off the lights which caused a *security delay* of 1 hour to be held. This is why the device agent couldn't perform the change which frankly would have happened. Again, by applying the solution explained in chapter: 24, we would avoid such incidents completely since no such delay would herewith be necessary anymore.
Ordinate: Logarithmically scaled interior daylight value

25.3 Room 55.G.74

Whereas room 55.G.26 (latter section: 25.1) corresponds to a regular office with two occupants in it, this room is our laboratory. Occasionally two other occupants were working within this environment at this time period and hence should have benefited from the autonomous IB as well. However it's reasonable that in contrast to the occupants in room 55.G.26, we were more or less aware that our room is being supervised and controlled by two separate device agents that were appropriately performing the actions we told them to learn. Hence it is not astonishing that the results were nearly perfect.

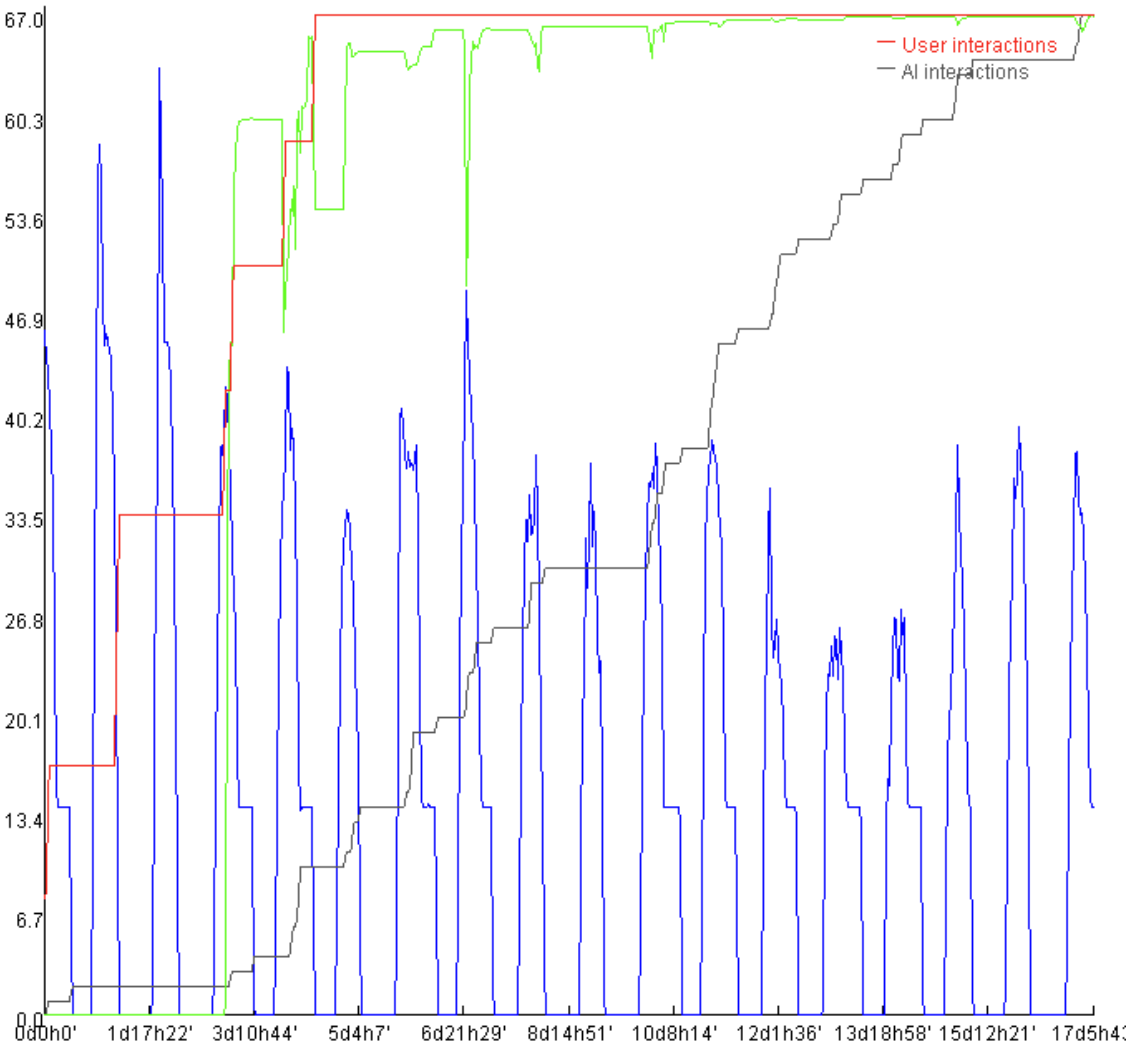


Figure 25.8: The window light was on all the time (our desk).Ordinate: AI interactions

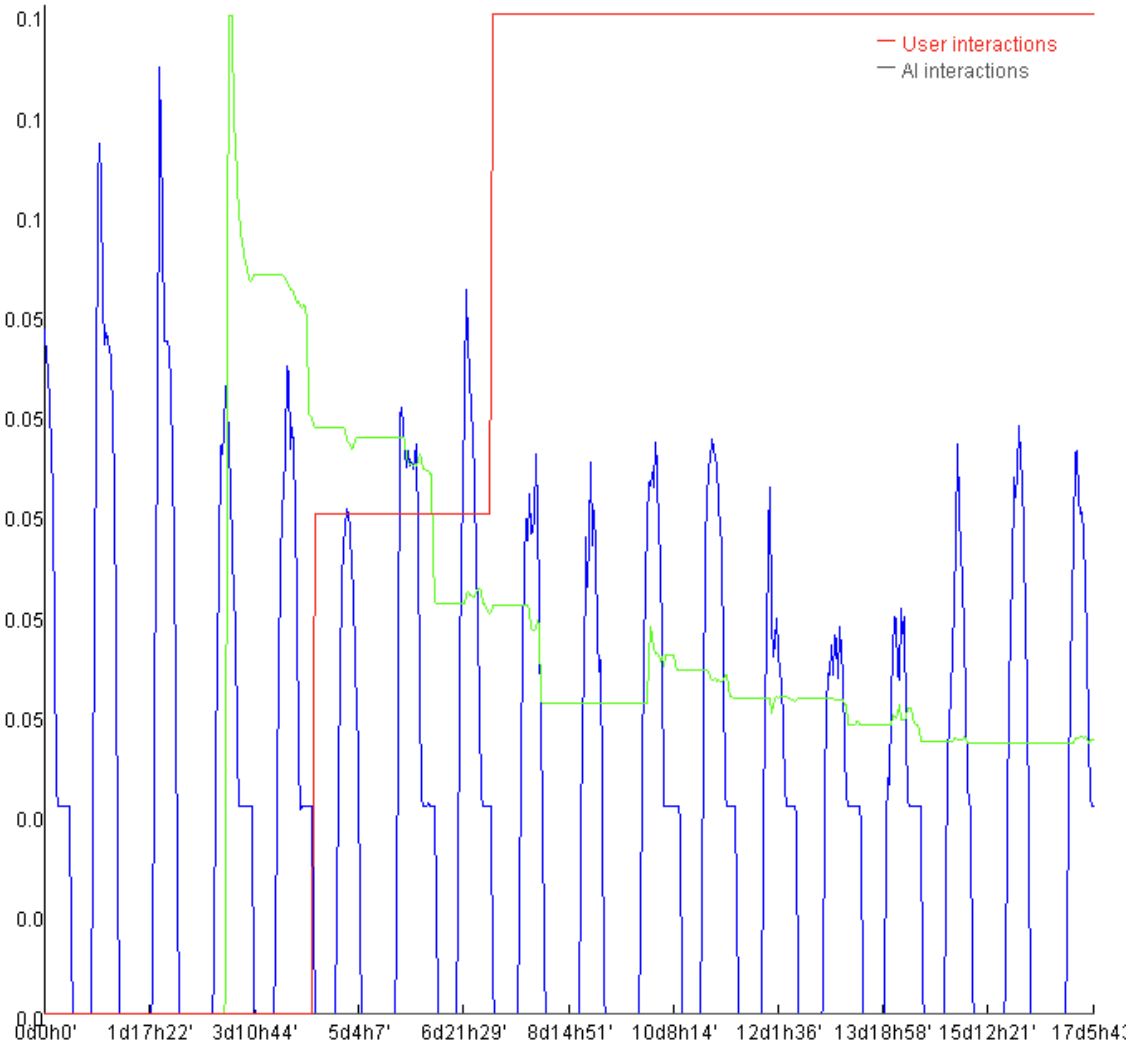


Figure 25.9: The corridor light was off all the time.Ordinate: Prediction

25.4 Other remarks

We further noticed a couple of side results which might also be of importance one day.

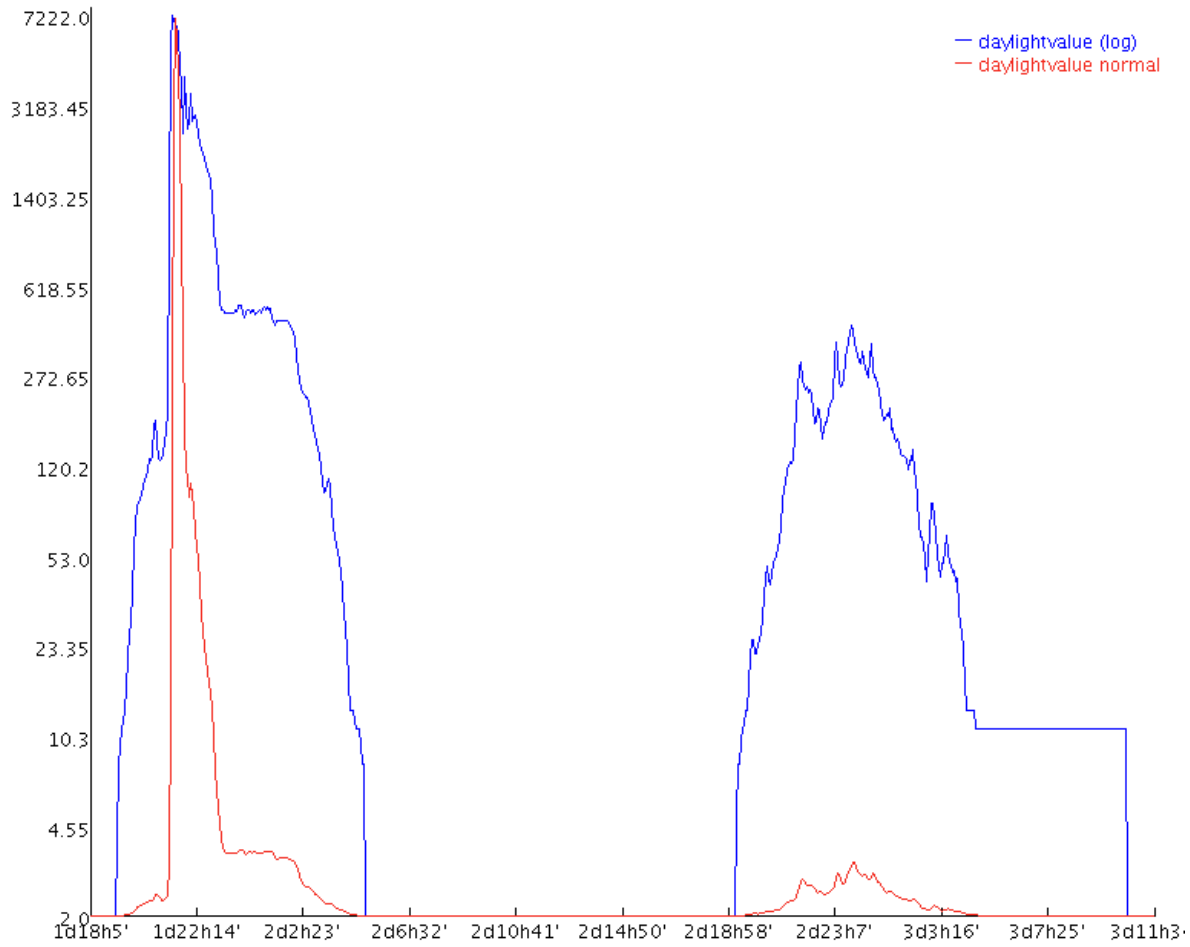


Figure 25.10: As discussed in section: 11.2, we can clearly depict that logarithmically scaling the interior daylight value seems to be adequate since we can still differentiate among a huge scale of values (up to 100'000) and allows us to clearly designate when a light needs to be switched on and when off. Additionally we can depict here that synthetic lighting impacts the environmental daylight a little.
Ordinate: Logarithmically scaled interior daylight value

Another important factor that we've been neglecting is the influence of the synthetic lighting. For instance in certain rooms (i.e. 55.G.84), we've been observing that this influence is quite noticeable especially at night when the exterior radiation spectrum stops to impact the perception of the internal daylight sensors. Consequently we can depict that within a couple of minutes the daylight strength starts to decay a little. Presumably due to the increasing internal heat of the lamp or so.

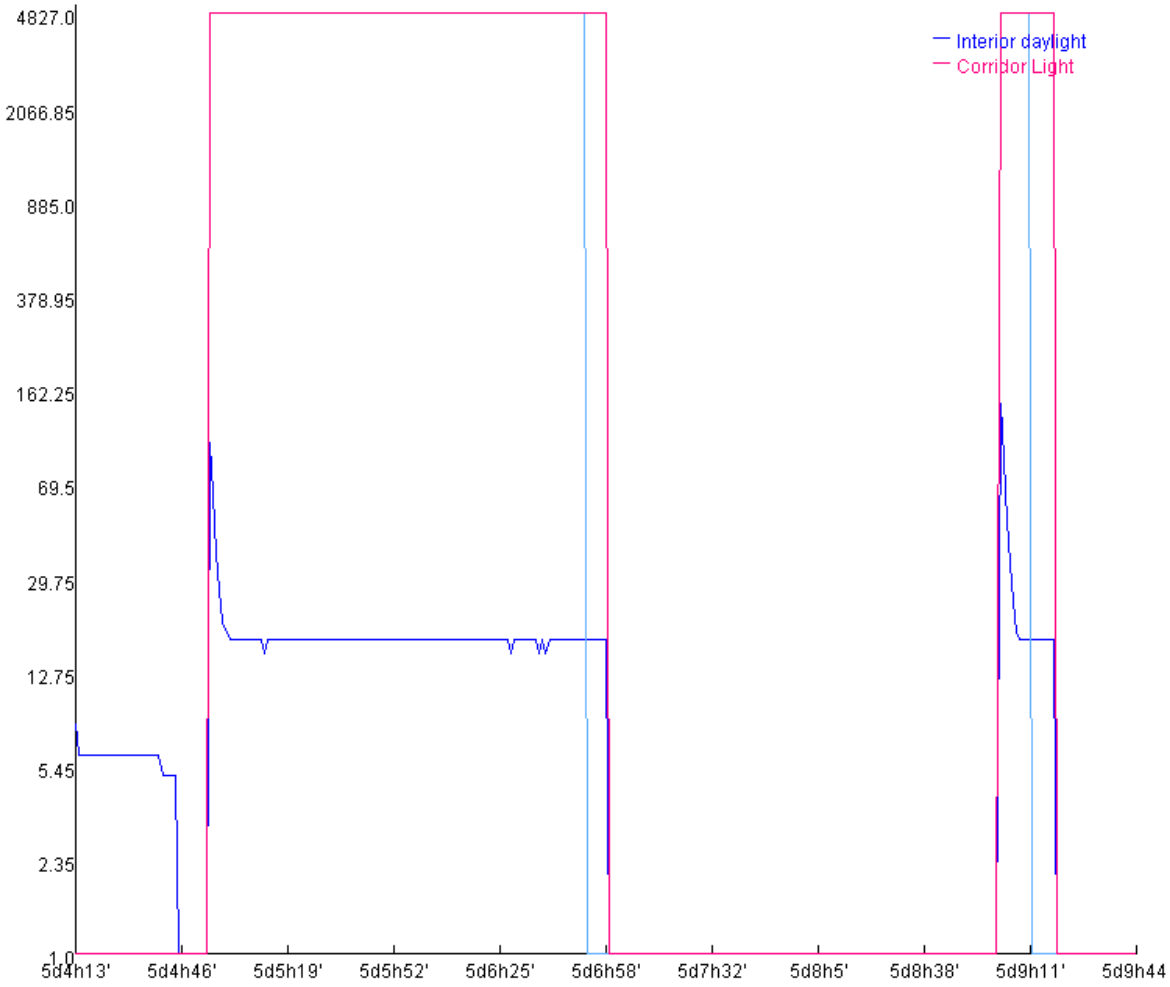


Figure 25.11: Synthetic lighting impacts the interior daylight.
Ordinate: Logarithmically scaled interior daylight value

Such time dependencies must especially be considered when starting to control the blinds or even the heating, cooling or the fan (See some of the topics in the future work part such as: 27.3).

Part VIII

Conclusion and Future Work

Chapter 26

Conclusion

26.1 ABI System based on OSGi

Within this thesis we realized a fully functioning new ABI System that has completely been realized using an OSGi Framework. Such a Framework defines a service platform specification that delivers an open, common architecture for service providers, developers, software vendors, gateway operators and equipment vendors to develop, deploy and manage services in a coordinated fashion. Thus, LON, as well as other technologies i.e. Wireless, can be integrated to one common piece and are herewith compatible with each other within this framework (hardware transparency).

Additionally, a variety of distributed user interaction agents have been accomplished which allow an administration and control over any environmental part within the INI.
In order to provide a possible reconstruction of sensory as well as effector data, a logging agent has been developed that persists past happenings which can later be used for data mining and generic reasoning when looking up certain incidents again.

26.2 Learning and Controlling

We developed an Intelligent Building Framework that has proven to succeed to handle different learning algorithms by judging them in their performance. Hereby, the performance is steered as a function of the user input. The IB Framework provides a common generic architecture that facilitates different prediction algorithms to be tested and competed against others. Additionally, to the entire framework, a real-time simulation (RTS) software component has been developed that can be plugged to the IBF by using a set of well defined interfaces. The RTS potentiates possible environments to be mimicked by adjusting a large variety of provided parameters. Such a simulation platform opens the capability to test algorithms on their different strengths and weaknesses by taking different climatic factors and other conditions into account, which elswise would simply not be possible. Nevertheless we've noticed a couple of things which would need to be improved within the simulator. For instance blinds have not optimally been steered by the fictional user as well as test configurations should be made more realistic by taking advantage of the sensory and effector data stored in the developed database.

According to the learning and the result part, we've seen that more excessive tests are necessary in order to judge whether the system was helpful in supporting its inhabitants.
Furthermore, it should be remarked that the current period is in our point of view one of the easiest seasons to learn since the exterior daylight level is commonly quite low and does not take any crucial jumps. We will be testing and observing season transitions that will better illustrate the capability of all device agents within the IB. The question hereby isn't about whether each of the device agents would adapt to new situations, since we already proved that they would. The most interesting question though is how fast will they get

accustomed to new situations.

Among other algorithms, we proposed a new variant, clustering, that we belief improves future predictions in their speed and accuracy by preforming additional ambient noise filtering.

We further illustrated the problem of keeping a fixed time delay that at first sight seems to be reasonable to be held upon receiving user interactions. However a couple of tests (real and simulation) have shown that such a proceeding partly violates against one of the most important IB principles, *not to disturb occupants*.

We also pointed out that certain input values need to be pre-processed in order to provide an optimal representative, pivotal information content i.e. One simple form has been demonstrated by scaling interior daylight values logarithmically.

Due to several presence problems, additional efforts must be taken in order to provide a stable and reliable presence detection, that finally allows us to re-consider the temporary solution of keeping a fixed presence delay. The goal that hereby must be achieved is, to reduce such delays (which in turn would minimize the energy consumption) without affecting user comfort.

Chapter 27

Future Work

This chapter addresses certain issues that have been encountered and identified while conducting this thesis and partly proposes a couple of suggestions which need to be realized at a later stage.

27.1 Presence Detection

We recall that we were having troubles with providing a reliable presence detection in any environment and had to incorporate additional PC Presence detectors (See chapter: 8) which helped us to improve the detection of presence within our test environments.

However according to figure: 25.3 some odd presence announcements have been made in certain environments. In example in the middle of the night at a weekend we evidenced positive presence although in fact their wasn't. Hence some presence sensor must have cheated somehow. We figured out that the PC Presence detector was announcing a misleading signal upon receiving a remote connection through ssh or such. Therefore remote connections may announce presence when being run under certain platforms (windows). Well this isn't that kind of tragic but quite a lot of occupants here at the Institute of Neuroinformatics, gradually work from their home office rather then at the institute.

As a result a dedicated building intelligence instance will start to learn stuff that it actually wasn't suppose to learn. Tasks such as switching certain lights on in the middle of the night, but also switching off the lights during the daytime (since the daylight has been categorized as sufficient enough).

There has always been a problem with providing a reliable presence detection. Anyhow the PC Presence detectors are one of the most important sources that can nonetheless announce a quite reliable presence signal but some alterations need to be developed in order to avoid such incidents in the near future. In example by consulting other system calls to retrieve mouse motions and keyboard hits.

Nevertheless we also noticed that certain occupants do not regularly work on their machine that much and thus a little unsteadiness will still remain. However we can at least provide a more sensitive way to detect presence when direct intervisiblity is provided.

As described in section: 2.1 we have been using a very simple form of filtered presence detection. When trying to be more ecological some more advanced techniques must be provided then just applying a certain delay upon sensing positive presence (See figure: 27.1). In the future we might think of a way to measure the busyness of a room in order to identify whether a room is frequently visited or not. Consider a kitchen for example. A kitchen such as a one here at the INI is in contrast to regular office less occupied or busy then a regular office. To take advantage of this fact we might think of a way to lessen such presence delays. Another interesting approach that might be worth to examine is to test whether such spiking presence announcements follow a certain distribution and perform a statistical test on the hypothesis (Similar to the

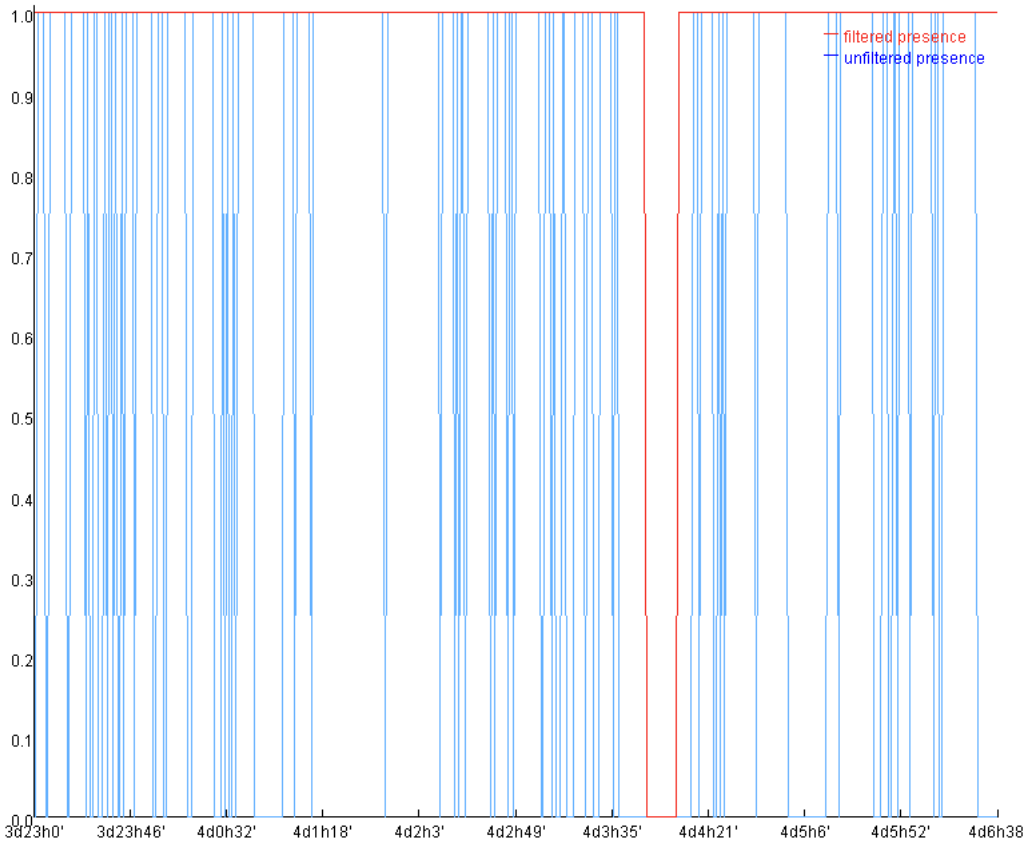


Figure 27.1: Filtered (10 minutes) vs. unfiltered presence status

GMeans, See chapter: 21). Broadly speaking it is quite difficult to deal with a reliable presence detection since aging of sensors can also influence the system performance. In some cases we may require a dynamic re-calibration to account for degradation. Hence the current solution is not really appropriate even though the used PIR presences sensors do have some self-calibrating mechanism incorporated. However we don't known how far and reliable this is and will be.

In order to enhance possible future presence detection we propose a couple of other solutions. One solution that we started looking into is motion detection based on computer vision through a camera. For this reason we developed a few test applications that we started to experience around with (See section: 27.1.1).

The other solution is rather based on the thought of seeking better suited presence detectors currently in retail (See section: 27.1.2).

27.1.1 Motion detection with a webcam

One simple way to improve the presence detection in an environment is to basically try to detect or even track body motions that can be recorded with high resolution cameras. A couple small test implementation that we've developed showed that even when motions are sparse in nature their is still a good chance to notice such motions (See figures: 27.2, 27.3, 27.5, 27.4).

However one may encounter several problems upon applying such an approach. Practically, each picture taken by such cameras is usually noisy and scratchy and therefore might lead in a false-positive conclusion of presence. Hence, some efforts must be taken to prepare the images, for instance by applying filtering or image segmentation.

Additional care must be taken when classifying such motions since some of them might have been caused by a blowing fan and thus would cause false tripping.

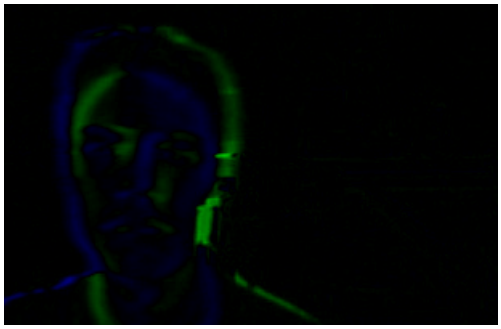


Figure 27.2: Normal human motions are clearly noticeable

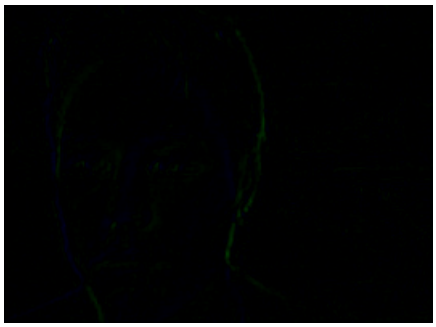


Figure 27.3: Sparse human motions are still noticeable

Figure 27.4: Normal human motions further away are clearly noticeable

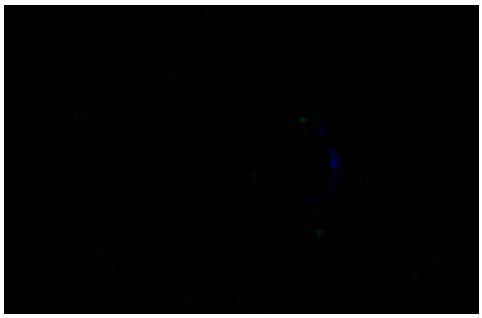
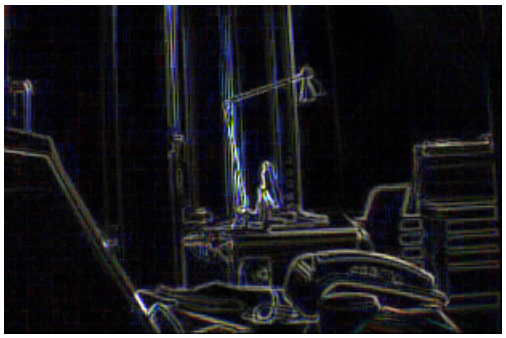
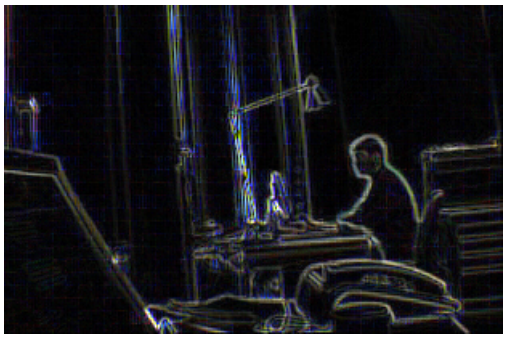


Figure 27.5: Sparse human motions further away are still noticeable

Although we might be able to sense such recurring motion patterns by applying a structure analysis that takes such constant velocities into account before taking a decision (See figure: 27.1.1, 27.1.1). Such recurring motions may for instance be possible to be detected by tracking features of a motion.





27.1.2 Occupancy sensors

Another solution which should also be considered is to incorporate better suited physical devices with more capability at least in a test environment.
Hence we thought that it is worth to itemize and discuss some of the motion sensors currently in retail nowadays. We're in particular interested in their advantages and disadvantages. Note that such a device must of course be able to be either hooked on to the LonWorks or an other physical bus that can also be integrated into the ABI System.

Basically there 3 different type of occupancy sensors are available. Here at the INI we deal with passive infrared sensors. Passive infrared sensors (PIR) are triggered by the movements of a heat-emitting body through their field of view. Commonly they're placed on the ceiling of a room since the coverage is quite large due to the large angle (360°). The major drawback though is that such sensors cannot see trough walls and are usually only applicable for small offices where no huge separators such as noise protectors or tall lockers are available, where direct line of sight may be broken (See figure: 27.6).

In contrast to PIR sensors ultrasonic occupancy sensors may be considered. Such sensors basically emit an inaudible sound pattern that is disrupted by any moving object altering the signal returning to the sensor. The phenomena that is applied is based on the doppler shift. As a result no direct intervisiblity with the sensor is required. Unfortunately such sensors suffer from being to sensitive. For instance a bird flying close by a window might result an environment to be interpreted as occupied. Or even air movements caused by a fan may result in false tripping (See figure: 27.7).

A good solution that has been proven to be very successful but has the disadvantage of being very expensive is a combination of PIR and ultrasonic. Hence more reliable presence can be measured since they can benefit from both strengths (See figure: 27.8).

Figure 27.6: Passive Infrared. Ceiling Motion Detector. Prize: 99.95$

Figure 27.7: Low Voltage Occupancy Sensor with Ultrasonic Technology. Prize: 119.07$

Figure 27.8: Dual Technology PIR and Ultrasonic Sensor. Prize: 160$

27.2 Energy savings

Our observations have shown that occupants are out of their offices quite often during regular working hours. Hence we started to perform energy savings in switching off the lights once the occupants were leaving their offices.

However we can do more. In the introduction we mentioned that effectors within a building are not efficiently used. Many occupants leave their lights on they are being switched on, even if the illumination is at a decent level. A further enhancement which is quite difficult to achieve, but worth to try is to start the saving process once the IB got accustomed to its occupants. Of course this provides that all occupants within the environment must have first been satisfied completely. Once this is achieved we can start to lessen the dynamic threshold to move closer to energy-saving potentials.

However the saving process must first need to be started by providing reliable blinds that can be controlled since a lot of occupants rather tend to switch on the lights instead of making use of the exterior daylight. This fact might be very useful for a IB to take advantage of. By rather moving up the blinds instead of switching on the lights. Herewith an IB might try to "teach" the occupants to get accustomed to the exterior daylight a bit better and thus would achieve better ecological results.

Additional energy saving potentials besides the most expensive resources such as heating and cooling can be realized when synthetic lights can be dimmed. Dimming is also less obtrusive to occupants than switching off or on a certain zone of an environment where other occupants remain working which is generally experienced as disturbing. If the user perceives the environment created by the system to be uncomfortable or disturbing in any way (noisy or too abrupt in its on-off switching), the system is likely to get rejected or an attempt will be made to compromise it. Energy savings are therefore directly related to a system's acceptance and proper operation by the user. Hence an inappropriate ambience will eventually be rejected by occupants.

27.3 Blinds

We already mentioned (See introduction: 2.1) that the blinds couldn't have been incorporated in the current building intelligence due to their problematic control issues.

However it might be worthwhile to mention that blinds can't be treated in the same way as the lights. Each light device agent presumes that a light status change will not affect the interior daylight to be changed by a considerable amount and hence has been neglected in its decision making. However when trying to carry the same input vector $\vec{x}$ to the blinds we'll pretty soon realize that this vector will not be sufficient enough to intelligently control the blinds. The reason is that the interior daylight now basically gets produced by the positioning of each blind within an environment that we need to control. Hence we would need something steady to work with. Steady in the sense that sensory inputs should not cause sudden influencing changes when altering any effectors in general i.e. performing blind movements, when not being able to envisage the resulting consequences.

At first glance, the exterior daylight may be an option to be added since it doesn't depend on any effector. However in someway this is also questionable when simply adding it to the overall input vector. Since the interior daylight might still carry to much noise with it i.e. When the interior daylight is high, the blinds are probably up and will logically be learnt by a blind device agent. Hence the blinds will only move up when its bright inside. At the same time the blinds will never be moved down when its bright anymore.

However we'll be trying to incorporate the exterior daylight as the major measure and might also get rude of the interior daylight. The flip side upon trying to control the blinds without the interior daylight is that we might need additional inputs to consult i.e. the daytime as well as the altitude of the sun since sunbeams or instant light reflections probably couldn't be recognized elsewise.
Other solutions would involve slopes to be trained which is quite more complicated then dealing with simple states. Thus other machine learning techniques might need to be applied which can deal with changing

outputs which impact certain inputs again.

27.4 Algorithms and Parameter optimization

In the learning part we annotated that this thesis rather emphasizes in exploring different learning algorithms or approaches then testing and optimizing each algorithm.
The time period of the diploma thesis was just to short to experiment around with other machine learning algorithms in detail and paralelly extend the ABI System in order to possible any learning tasks in the first place.

The learning (See: VI) and partly also in result part (See: VII), clearly showed us that power of the G-Means and the GNG algorithm. Although we also illustrated that the G-Means suffers from evicting long-term knowledge that has been trained from subjects which suddenly start to change their habits or might have even left the environment for good and new occupants have moved in. The GNG does a fairly good job at the average. However its too bad that the GNG yields a persistent storage of sparse long-term data.
Hence possible future extensions might want to perform some adaption on them. Most algorithms actually skipped the consideration of direct user inputs since most algorithms tried to overcome this problem by incorporating a certain memory limit i.e. limiting clusters. It would be thinkable to extend each algorithm with more user centric behaviors. Several possible future extension examples have already been covered in the corresponding algorithm chapters. However to retrospect a few:

- In the statistical learning chapter: 18 we consider the alteration of the momentum value as a function of user inputs (similar to the IBF algorithm).
- Although the G-Means algorithm maintains a clusters size limit, some chunks of laziness might still be resolved by considering a local instead of global regression since local clusters might be able to carry an adaptive aging that could be altered as a function of user input also. A global aging could also be thinkable but would rather tend to end-up in a GNG like behavior.
- The GNG algorithm could be adapted to consider an aging process on the basis of user interactions as well. Similar to the G-Means algorithm a local or a global regression would then be thinkable. A global would surely be easier to accomplish then a local one since we could skip a statistical test on each neuron.

Since each algorithm limits the size of its data (in form of clusters or elsewise), a temporary but online knowledge base has already been applied and therefore may be questionable to even consider user inputs in the first place. On the top of that it is quite hard to test such situations since user behavior changes within an environment barely take place. Not to mention that in a simulated environment such behaviors are very hard to imitate. Quite often we noticed that IB predictions where in fact correct but just a little too late. Particularly in such cases, a long during real environment test would be necessary.

Furthermore certain environments might better be satisfied by just allow manual control over those areas, instead of trying to intelligently control them. Manually, either by the provided user interaction agents (See part: III) or mechanically by using the wall switches.
We might meet such environments upon device or network failures. However such environments are very hard for an IB to detect and deal with since controlling is quite often interrupted or sensory inputs suddenly start to fail or return misleading values that makes decent controlling impossible. For those situations we should maintain a dynamic overall satisfaction or comfort level that should be integrated in the IBF algorithm. If a device controller starts to annoy its occupants by undergoing a maximal pre-defined comfort level the controlling of that device should be stopped for a while. Gradually though, a possible "reanimating" procedure should also be reassessed to check whether controlling is possible again.

Part IX

Glossary and Bibliography

Chapter 28

Glossary

Abbreviation	Explanation / Comment
ABI	Adaptive Building Intelligence
ABI System	The ABI System implemented using an OSGi Framework
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
Bundles	Java JAR archive
DAI	Distributed Artificial Intelligence
DC	Decision Controller
DLU	Decision and Learning Unit
DHCP	Dynamic Host Configuration Protocol
DSP	Discovery Service Protocol
EDIFICIO	Efficient design incorporating fundamental improvements for control and integrated optimisation
EPFL	Ecole polytechnique federale de Lausanne
ETH	Swiss Federal Institute of Technology
FIFO	First in first out
GNG	Growing Neural Gas
GUI	Graphical User Interface
HSR	University for applied sience Rapperswil
HTTP	Hypertext Transfer Protocol
IB	Intelligent Building
IBF	Intelligent Building Framework
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
INI	Institute of Neuroinformatics
JAR	Java Archive
JDBC	Java Database Connectivity
LCS	Learning Classifier Systems
LNS	Lon Network Server
LON	See LonWorks
LonWorks	Field bus network protocol / standard
LTM	Log Term Memory
MAS	Multiple Agent System
MISO	Multiple Input Single Output
NG	Neural Gas
PCA	Principal Component Analysis
OSGi	Open Service Gateway initiative
PIR	Passive Infrared Sensor

Abbreviation	Explanation / Comment
RBC	Remote Building Control (Protocol)
RBC API	The protocol abstraction layer API that implements the RBC Protocol
RBC Server	The Server Bundle of the ABI System
RTS	Real Time Simulation
RMI	Remote Method Invocation (Java Technology)
RMS	Root Mean Square
RPC	Remote Procedure Call
RSI	Remote Service Invocation
RSP	Remote Service Protocol
SOM	Self Organising Map
STM	Short Term Memory
Service	Interface exported by the service provider bundle
UML	Unified Modeling Language
UNIZH	University of Zurich, Switzerland
WLS	Weighted Least Square

Bibliography

[BG04a] Patrick Brunner and Simon Gassmann. Adaptive building intelligence based on the open services gateway initiative, diploma thesis. Technical report, University of Applied Sciences Rapperswil, Switzerland and Institute of Neuroinformatics, Swiss Federal Institute of Technology, Zurich, Switzerland, 2004.

[BG04b] Patrick Brunner and Simon Gassmann. Adaptive building intelligence based on the open services gateway initiative, term work. Technical report, University of Applied Sciences Rapperswil, Switzerland and Institute of Neuroinformatics, Swiss Federal Institute of Technology, Zurich, Switzerland, 2004.

[Bis95] C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.

[BUI] Intelligent buildings. http://www.arch.hku.hk/teaching/bs2/sbs-08.pdf.

[Bul05] Larry Bull. Learning classifier systems: A brief introduction. Technical report, Faculty of Computing, Engineering and Mathematical Sciences University of the West of England, 2005.

[CCSZ21] J. L. Castro, J. J. Castro-Schez, and J. M. Zurita. Learning maximal structure rules in fuzzy logic for knowledge acquisition in expert systems. *Fuzzy Sets and Systems*, 101:331–342, 1999/2/1.

[CP01] F. Sperduto C. Priolo, S. Sciuto. Efficient design incorporating fundamentals improvements for control and integrated optimisation. Technical report, Federal Office of Education and Science, Switzerland, 2001.

[CZ16] J. L. Castro and J. M. Zurita. An inductive learning algorithm in fuzzy systems. *Fuzzy Sets and Systems*, 89:193–203, 1997/7/16.

[CZ65] Juan Luis Castro and Jose Manuel Zurita. A generic atms. *International Journal of Approximate Reasoning*, 14:259–280, 1996/5.

[DHZS02] Chris Ding, Xiaofeng He, Hongyuan Zha, and Horst D. Simon. Adaptive dimension reduction for clustering high dimensional data. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, page 147, Washington, DC, USA, 2002. IEEE Computer Society.

[Fab] Vance Faber. Clustering and the continuous k-means algorithm.

[Fen04] Andreas Fenkart. Personalized blind control based on handcrafted adaptive controller. Technical report, Institute of Neuroinformatics, Swiss Federal Institute of Technology, Zurich, Switzerland, 2004.

[Fri95] Bernd Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA, 1995.

[Fri97] Bernd Fritzke. A self-organizing network that can follow non-stationary distributions. In *ICANN*, pages 613–618, 1997.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[GNG] Growing neural gas. http://www.neuroinformatik.ruhr-uni-bochum.de.

[HE04] Greg Hamerly and Charles Elkan. Learning the k in k-means. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[Hol] Jaakko Hollmen. Principal component analysis.

[Hol86] John H. Holland. Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In Mitchell, Michalski, and Carbonell, editors, *Machine Learning, an Artificial Intelligence Approach. Volume II*, chapter 20, pages 593–623. Morgan Kaufmann, 1986.

[iLO] LonWorks Networking Platform. http://www.echelon.com/products/internet/ilon1000.htm.

[Ins] Institute of Neuroinformatics. http://www.ini.unizh.ch/.

[JAV] J2se. http://www.sun.com.

[JH04] Sandra Blakeslee Jeff Hawkins. *On Intelligence*. Times Books, 2004.

[KNO] Knopflerfish. http://www.knopflerfish.org/index.html.

[LES] Solar energy and building physics laboratory (leso-pb). Antoine Guillemin, Dr. Nicolas Morel.

[LF] Hartmut S. Loos and Bernd Fritzke. Demogng v1.3.

[loc] LonWorks Networking Platform. http://www.echelon.com/products/lonworks/.

[LTM] Long-term memory (ltm). http://coe.sdsu.edu/eet/articles/ltmemory/start.htm.

[Lyt02] William W. Lytton. *From Computer to Brain*. Springer-Verlag New York, LLC, 2002.

[MAT] Anderson-darling test. http://www.itl.nist.gov.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGRAW-HILL, 1997.

[MUL] Techniques: The multi-layer perceptron. http://www.qub.ac.uk/mgt/intsys/backprop.html.

[NB05a] Stephan Kei Nufer and Mathias Buehlmann. Intelligent, learning system. Technical report, University of Applied Sciences Rapperswil, Switzerland and Institute of Neuroinformatics, Swiss Federal Institute of Technology, Zurich, Switzerland, 2005.

[NB05b] Stephan Kei Nufer and Mathias Buehlmann. Remote building control api – an api that supports custom abi client applications. Technical report, University of Applied Sciences Rapperswil, Switzerland and Institute of Neuroinformatics, Swiss Federal Institute of Technology, Zurich, Switzerland, 2005.

[NB05c] Stephan Kei Nufer and Mathias Buehlmann. Remote building control protocol – a protocol that is used to transfer building data. Technical report, University of Applied Sciences Rapperswil, Switzerland and Institute of Neuroinformatics, Swiss Federal Institute of Technology, Zurich, Switzerland, 2005.

[OSG] Osgi alliance. http://www.osgi.org.

[Per] The origin of personas. http://www.cooper.com/content/insights/newsletters/2003_08/Origin_of_Personas.asp

[PM00] Dan Pelleg and Andrew Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, San Francisco, 2000. Morgan Kaufmann.

[RJD04] U. Rutishauser, J. Joller, and R. Douglas. Control and learning of ambience by an intelligent building. *IEEE Transactions on System,Man and Cybernetics Part A, Submitted*, 2004.

[RS02] Ueli Rutishauser and Alain Schaefer. Adaptive home automation – a multi-agent approach. Technical report, University of Applied Sciences Rapperswil, Switzerland and Institute of Neuroinformatics, Swiss Federal Institute of Technology, Zurich, Switzerland, 2002.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical J.*, 27:379–423, 1948.

[SOM] Kohonen networks. http://www-cse.stanford.edu/classes/sophomore-college/projects-00/neural-networks/Architecture/.

[TZ03a] Jonas Trindler and Raphael Zwiker. Adaptive building intelligence – an approach to adaptive discovery of functional structure. Technical report, University of Applied Sciences Rapperswil, Switzerland and Institute of Neuroinformatics, Swiss Federal Institute of Technology, Zurich, Switzerland, 2003.

[TZ03b] Jonas Trindler and Raphael Zwiker. Adaptive building intelligence – parallel fuzzy controlling and learning architecture based on a temporary and long-term memory. Technical report, University of Applied Sciences Rapperswil, Switzerland and Institute of Neuroinformatics, Swiss Federal Institute of Technology, Zurich, Switzerland, 2003.

[UL03] Jürgen Cleve Uwe Lämmel. *Künstliche Intelligenz*. Fachbuchverlag Leipzig, 2003.

[WIK] Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Main_Page/.

[YWN05] Kalai Mathee Yong Wang, Chengyong Yang and Giri Narasimhan. Clustering using adaptive self-organizing maps (asom) and applications. Technical report, Bioinformatics Research Group (BioRG), School of Computer Science, Florida International University, Miami FL 33199, USA, 2005.

[ZK04] Xiao-Jun Zeng and John A. Keane. Approximation capabilities of hierarchical fuzzy systems. *IEEE TRANSACTIONS ON FUZZY SYSTEMS*, 2004.