Design

- Develop the overall software architecture
    - Identify major system components
    - Specify interactions between system components
    - Specify system components
        - interfaces
        - semantecs
- The Design section should allow a skilled programmer to implement the system

In this chapter we will describe how the system is designed, and discuss the major decisions we have made in regard to the system design. Since the system is research minded, and since the purpose of the project is to analyze the possibilities of developing an intelligent home control system using machine learning technology, we had to make some adjustments to the development process. The traditional waterfall model for software development dictates that after finishing the project analysis, we would start designing how the system should handle the problems found in the analysis, along with the system architecture. Finally we would then implement the designed solution. With this project we were however faced with an additional challenge. When using machine learning you generally end up with a system that does not have an intuitive execution flow. This means that it can be almost impossible to predict the execution outcome because of the vast amounts of data that form basis for the systems decision making. This means that we have no way of verifying the validity of our proposed solution before implementing the system, or at least parts of it. Therefore we decided to approach the project by using incremental development instead.

In order to successfully apply this development model we must first divide the

project into smaller parts, that can be implemented with each cycle. This design approach also inspired our final system design. Just like the development had several phases, where each phase had to be concluded in order to activate the next, the system will have similarly have different stages of operations. These stages are determined by the amount of data the system have collected on the user.

The system will have three different stages of operation.

- In **The untrained stage**, the system is running, but it has not yet collected enough data to make intelligent decisions.
- The system enters **The learning stage** when there's enough data to attempt to manipulate the switches in the house. We call this the learning stage, because it provides us with a unique opportunity for the system to learn from the user. If the system makes a mistake and the user corrects it, e.g., the system turns off the lights and the user turns it back on, we can use that interaction to train our system further. In this case we can see it as the user punishing the system for making a mistake. The system will then adjust its decision scheme.
- After the system has been in the learning stage, it will enter its final stage, which we call **The evolution stage**. Here the system constantly updates its decision scheme with new data both from monitoring the user, and from being punished for its mistakes. In this stage there is a symbioses between the user and the system where the system reacts to the user and vice versa.

Each stage the system enters is based on the information the system as acquired in the previous stage. Thereby in the evolution stage we can attempt to address some of the shortcomings that are present int the previous stage. Because of this approach we were able to design, implement and evaluate each stage of the system, before continuing to the next.

In this chapter we will discuss the different stages of the system, the problems that are presint in each stage, and the solutions designed to solve these problems.

In the section "Theory" we will present the mathematical and statistical theory, that forms the basis for our machine learning algorithms.

In the untrained stage the system simply stores all sensor and switch events generated by the system, ordered by time of occurrence. This data collection is very simple, and will not be discussed in this chapter. In the chapter "Implementation" this process will be discussed further.

In the sections "Event pattern", "Training the system" and "Zones"we will discuss how the system analyses the collected user data in the learning stage. We will also provide a brief evaluation of the system in this stage, which will form the basis for the design of the final evolution stage.

In the final sections and "Switch and sensor correlation" we will discuss the processes at work in the final stage of the system.

Theory

Stand back! Im going to try science! -Randal Munroe

In the core of our system lies a series of machine learning algorithms. In this section we will explain some of the basic concepts of machine learning, along with the statistical theory that it is based on.

Machine learning

The purpose of machine learning is to have the system evolve behaviors based on empirical data, rather than programming a specific behavioral pattern. By using the supplied data as examples of relationships between data events, the system can recognize complex patterns, and make intelligent decisions based on the data analyzed[# wiki-machinelearning].

With **supervised learning** the system is give labeled data consisting of examples of correct behavior. Because of both the human factor, and the imperfection of the motion sensors, the system will generate a certain amount of invalid data called noise. The algorithm will have to extinguish between what is proper training examples and what is noise.

Markov chains

A Markov chain is a mathematical system that under goes transitions from one

stage to an other [#markov chains]. In a Markov system each step taken in a Markov chain is represented by a certain probability, based on the current state that the system is in. Formally:

\[P(X_{n+1} | X_n) \]

Here \(X_{n+1}\) represents the next state, and \(X_n\) represents the current state. And the entire notion is defined as the probability of the event \(X_{n+1}\) given that event \(X_n\) has just occurred.

By arranging these values in a matrix you can create a lookup table for future reference.

	\ **(X_1\)**	\ **(X_2\)**	\ **(X_3\)**	\ **(X_4\)**	\ **(X_5\)**				
\ **(X_1\)**	P(\ (X_1	X_1)\)	P(\ (X_1	X_2)\)	P(\ (X_1	X_3)\)	P(\ (X_1	X_4)\)	
\ **(X_2\)**	P(\ (X_2	X_1)\)	P(\ (X_2	X_2)\)	P(\ (X_2	X_3)\)	P(\ (X_2	X_4)\)	
\ **(X_3\)**	P(\ (X_3	X_1)\)	P(\ (X_3	X_2)\)	P(\ (X_3	X_3)\)	P(\ (X_3	X_4)\)	
\ **(X_4\)**	P(\ (X_4	X_1)\)	P(\ (X_4	X_2)\)	P(\ (X_4	X_3)\)	P(\ (X_4	X_4)\)	
\ **(X_5\)**	P(\ (X_5	X_1)\)	P(\ (X_5	X_2)\)	P(\ (X_5	X_3)\)	P(\ (X_5	X_4)\)	

Each cell in the table represents the probability of entering the state represented by the cells row, assuming the system is currently in the state represented by the cells column.

Markov chains of order

One of the most iconic features of Markov chains is the fact that they are memoryless. The probability of entering a new state is only based on the current state of the system. The states prior to the current have no effect on this

probability. With “Markov chains of order m” the system has memory of the last m steps in the chain, and these affect the probability of entering future states. the probability can be written as: \[P(X_{n+1}|X_{n},X_{n-1}, …, X_{n-m})\] Now the probabilities are calculated based on the pattern of steps made through the system rather than just the current state.

Since our probabilities are calculated based on collected data, we will not have to perform any complex statistical calculations.

Event patterns

We want to be able to trigger the switches, based on more than just where the user is right now. We want to be able to look at where the user is coming from, and try to predict where the light needs to be turned on or off. So the light is already on when the user enters a room, and is turned off where it isn’t needed.

We want to determine the series of sensor events, or pattern, that leads up to a user turning the lights on or off, e.g. which sensors are triggered when a user goes from the couch to the restroom. If a series of sensor events, are less than some time interval appart, we consider them to be part of a event pattern. The time interval needs to be long enough, that a user moving around normally is seen as a continuous event pattern, and not broken into fragements. The time interval also needs to be short enough, that different user action, is seen as separate event patterns. For instance, a user going the the kitchen to get a snack, and then returns to the living room, should ideally be seen as two separate event patterns.

With the idea of an event pattern, we can look at what patterns lead up to a switch event. And by extension of that analysis, when we observe an event pattern, we can determine the probability that it would lead to a switch event.

Training the system

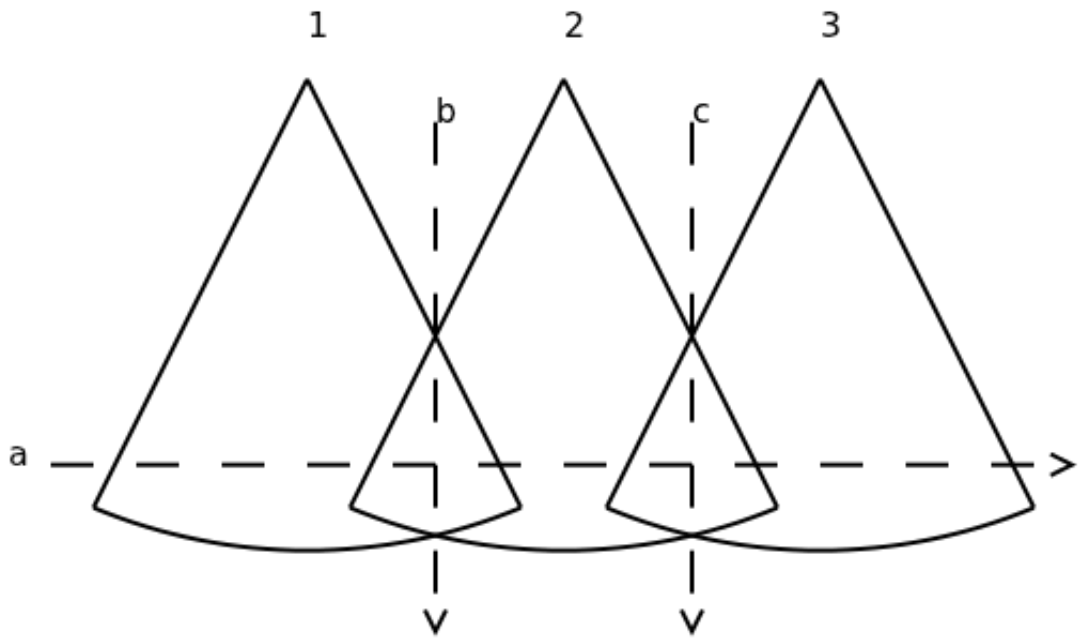
Decision Table

Zones

In many cases to cover an entire room with sensors, the sensors end up overlapping in some areas. These overlaps can be used to increase the precision

of the sensors. If two sensors triggers shortly after each other, then the user is in the zone where the two sensors overlap. In cases where multiple sensors triggers at the same time, it can be seen as one zone event.

Take as an example, of three sensors which overlap a bit, and three paths past the sensors a, b and c. The paths b and c should only be observed as zone events by the system. While path a should look something like 1, zone 1 & 2, 2, zone 2 & 3, 3. depending on the cooldown of the sensors each event may be multiple times in the pattern.



Confidence

A key transition of the system, is when does it go from the untrained stage to the learning stage? When is the system confident enough to take over control of the home. In the "placebo" setup, the system couldn't enter the learning stage, since it couldn't control the lights. Therefor this functionality wasn't implemented, but is still key feature of the system, and should be discussed in this report.

There are two main metrics we believe should determine when the system is confident enough: The system should start attempting to control the home, once it is confident enough, to act upon the decision schemes it has learned. But the system needs to have some quantifiable metric to determine it's confidence, before it start to take over control of the home:

1. The probability in the decision scheme must be above some threshold. \(P(switch_i | pattern_j) > \varphi \)
2. The specific \(pattern_j\) must have occurred atleast some number of times.

Exactly what the threshold should be, is up to speculation and could be determined through experimentation, once the system in ready to enter the learning stage. The second rule is to make sure, the system doesn't start acting based on patterns only observed once.

Switch and sensor correlation

It is beneficial to get a sense of which sensors are near which switches. And we have a lot of statistical data too look at. When a user turns a which on, it's most likely because there isn't light where the user intends to be in the immediate future. So it is possible to get an idea of which sensors are near a which, by looking at the interval shortly after a switch is turned on.

When flicking a switch off, the user may be leaving the room, or just have entered the room to turn the switch off. Each of the two cases are just as likely as the other, but the sensor events in the interval leaving up to the off event is completely opposite.

Based on the statistical data it is possible to generate a table of probability that a sensor is triggered shortly after a switch is turned on, and by extension of that give a idea of wich sensors are in the same room as a switch

\[P(sensor_i | switch_j , \Delta t) = \frac{\sum 1_{sensor_i} (switch_i, \Delta t) } {\sum switch_j \ events } \]

The indentity function \(1_{sensor_i} (switch_i, \Delta t) \) is 1 if the sensor is triggered within \(\Delta t\) after \(switch_j\) is triggered, and is not therefor not counted twice, in the sensor triggeres multiple times after the same switch event.

So to reiterate \(P(sensor_i | switch_j , \Delta t) \) is the probability that \(sensor_i)\) fires within \(\Delta t\) after \(switch_j\) fires.

Correlation table

Correlation table

	sensor 1 \ ((se_1)\)	**sensor 2 \ ((se_1)\)**	**…**	**sensor n \ ((se_n)\)**			
switch 1 (\ (sw_1)\)	\(P(se_1	sw_1, \Delta t)\)	\(P(se_2	sw_1, \Delta t)\)	…	\ (P(se_n	sw_1, \Delta t)\)
switch 2 (\ (sw_2)\)	\(P(se_1	sw_2, \Delta t)\)	\(P(se_2	sw_2, \Delta t)\)	…	\ (P(se_n	sw_2, \Delta t)\)
\ (\vdots\)	\ (\vdots\)	\ (\vdots\)	\ (\ddots\)	\ (\vdots\)			
switch m (\ (sw_m)\)	\(P(se_1	sw_m, \Delta t)\)	\(P(se_2	sw_m, \Delta t)\)	…	\ (P(se_n	sw_m, \Delta t)\)

Correlation based timeout

Ideally the system will turn off the light by detecting off patterns, but in the learning stage or if the user changes behavior, this isn't reliable. We want to avoid is the light being on longer than it needs to, even if the system doesn't detect the off pattern. The user leaves a room and doesn't realize the light is still on, or expect the system to turn off the light on it's own, causing an necesary waste of energy. We wanted to make a catch-22, a situation where no matter what happens the light is eventually turned off. The system has a timer for each switch, and as the user is detected by the sensors, the timer is extended based on the correlation to the switch. In a real scenario it's very like for any sensor to have atleast some correlation to any switch, however low it might be. So the system has to avoid having all sensors extending the timeout ever so slightly, essentially keeping the light on foreven, as long as sensors events keep firing somewhere. Therefor the correlation has to be above some threshold in order to extend the timeout. Ideally only sensors in the same room as the switch are extending the timeout.

Timeout adjustment

The problem with a timeout based solution, is people sitting still. Most people

have experience controllable or programmable smarthouse solutions, where motion sensors keep the light on for some amount of time. And it tend to work great in spaces where people are passing through, hallways, carports, et cetera. But in places where people some times sit still, be it working or relaxing, motion sensors won't be triggered, and user end up having to get up or wave their arms to keep the lights on. So we allow the system to keep the light on for longer duration in some areas, based on which sensors are triggered, and also a way for the system to learn where these areas are. As already stated the base timeout is based on the correlation, which means sensors close to the switch will keep the light on longer. A common senario in a home would be a user laying on a couch watching TV. So we want the system to be able to keep the lights on longer, if it detects that the user is on the couch. If a timer runs out, the system turns the switch off, and if the user immediately turns it back on again, the system takes that as a punishment for it's behavoir. The system reacts by increasing the timeout those sensors have to the switch. Adversely if a timer runs out, and the user doesn't take any action, it assumes it's behavior was correct, and decreases the timeout.