BABEŞ-BOLYAI UNIVERSITY

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Fuzzy logic-based IoT enable thermostat

B.Sc. Thesis

PhD student: Candale Andrei

Scientific supervisor: Lect. Ph.D. Radu D. Găceanu

2015

Contents

List of Figures

List of Tables

Introduction

The evolution of modern technologies has lead to an increased number of Do-It-Yourself (DIY) possibilities in the field of electronics and microchips; thus it has become easy and affordable to incorporate such technologies in mundane, house-hold objects that can present themselves with some data that may be relevant. This new concept turned into a movement now called the Internet of Things (IoT).

We use the term Internet of Things to refer to the general idea of things, especially everyday objects, that are readable, recognizable, locatable, addressable, and/or controllable via the Internet, wireless LAN, wide-area network, or other means. Everyday objects includes not only the electronic devices we encounter everyday, and not only the products of higher technological development such as vehicles and equipment, but things that we do not ordinarily think of as electronic at all, such as food, clothing, and shelter; materials, arts, and sub-assemblies; commodities and luxury items; landmarks, boundaries, and monuments; and all the miscellany of commerce and culture. [7]

In order for this concept to grow, technologies had to be made available to the general public, meaning that the cost at which they came had to be affordable. Fortunately, the current technological context lead to the development of cheap and powerful components which can now be used to build the infrastructure of the IoT.

With the now given possibility to acquire powerful, yet cheap, means by which one can control and gather data from different contexts, various applications can be developed, like the one described by this paper.

The field of Internet of Things has known a major boom in the recent years, adding to its infrastructure all kinds of objects, appliances etc, giving the possibility to monitor and analyze data from just about anything. The broader affect of this, is the fact that data can be stored, analyzed and made sense of it in order to increase the comfort level and to grasp a better understanding of the environment we are living in.

Starting from this idea, this paper describes a new application that makes uses of the cheap technology we have at hand and the easy access to a world wide data transport infrastructure (the Internet). The result is a device that increases home comfort levels, gathers data that is analyzed and makes of it information that is digestible by a human being, enabling one to better understand and control the context one is living in.

The chosen hardware for the proposed thermostat consist of 3 micro-controllers, temperature and humidity sensors and radio transceivers, all coming to a price comparable to a normal thermostat. The main processing unit is an ESP8266 wireless module. An in depth description of the architecture will be presented in section.

Chapter 1

Theoretical background

1.1 Introduction

Fuzzy logic is an extension of Boolean logic coined by Lotfi Zadef in 1965 [22] [24] [23]. Fuzzy logic is build on the theory of fuzzy sets which is a generalization of the classical set theory. Zadef introduced the notion of degree in the verification of a condition such that it could be in a state different from true or false. This provides a valuable extended flexibility for reasoning which makes it possible to take into account inaccurate data. One advantage of fuzzy logic is that the rules by which the reasoning is made can be expressed in natural language making it easy to take advantage of a human expert and the knowledge he/she posses.

This approach to set theory was not applied in control systems until the 70's due to the fact that technology did not yet reach the point where powerful enough computer would be developed. Professor Zadeh started from observing that people do not need precise, numerical information input to manifest the highly adaptive control they do. This lead to the idea that if computer would receive noisy, imprecise input values, using this method of control, they would be much more effective and easier to implement.

This technique is now used in a wide spectrum of applications, ranging from washing machines to car gear boxes [18].

The following sections will present the concepts surrounding fuzzy logic.

1.2 Fuzzy sets

Let X be a space of points (objects), with a generic element of X denoted by x. Thus, $X = x$. A fuzzy set (class) A in X is characterized by a membership function (characteristic) function $f_A(x)$ which associates each point in X a real number in the interval $[0, 1]$, with the value of $f_A(x)$ at x representing "grade of membership" of x in A. [**?**] Basically, a fuzzy set is

characterized by a membership function whose result, given an input x, represents how strongly does that input belong to a certain class A.

As opposed to crisp sets, where certain inputs may belong to one class OR the other, in fuzzy logic, a given input may belong to a certain class in a certain degree, belong to two classed at the same time or to none.

A crisp set is defined in the following way:

$$\mu(x) = \begin{cases} 0 & \text{if } x \in A \\ 1 & \text{if } x \notin A \end{cases} \tag{1.1}$$

As it can be observed, a given input can be classified as belonging to A OR as not belonging to A. As stated above, in fuzzy sets the belonging of an input to a class or the other is computed in the following way:

$$\mu(x) = \begin{cases} 0 & \text{if } x \in [a, b] \\ f(x) & \text{if } \in [b, c] \\ 1 & \text{if } x \in [c, d] \end{cases} \tag{1.2}$$

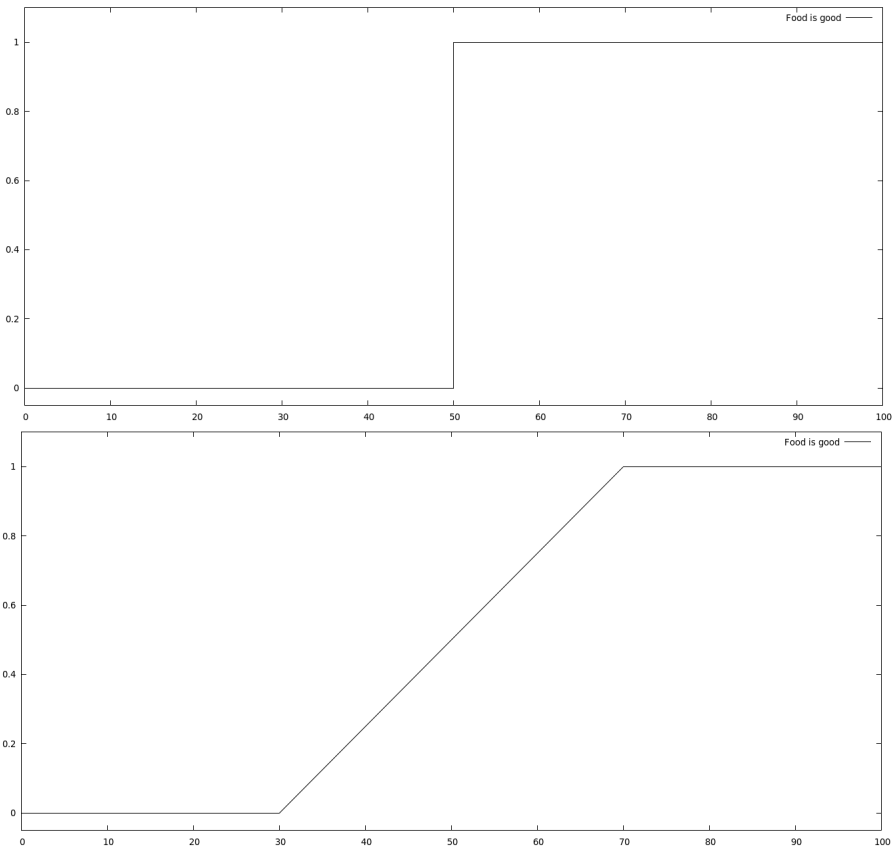

Figure 1.1: Crisp and fuzzy set graphical comparison

Figure 1.1 is a graphical representation that compares a crisp and fuzzy set.

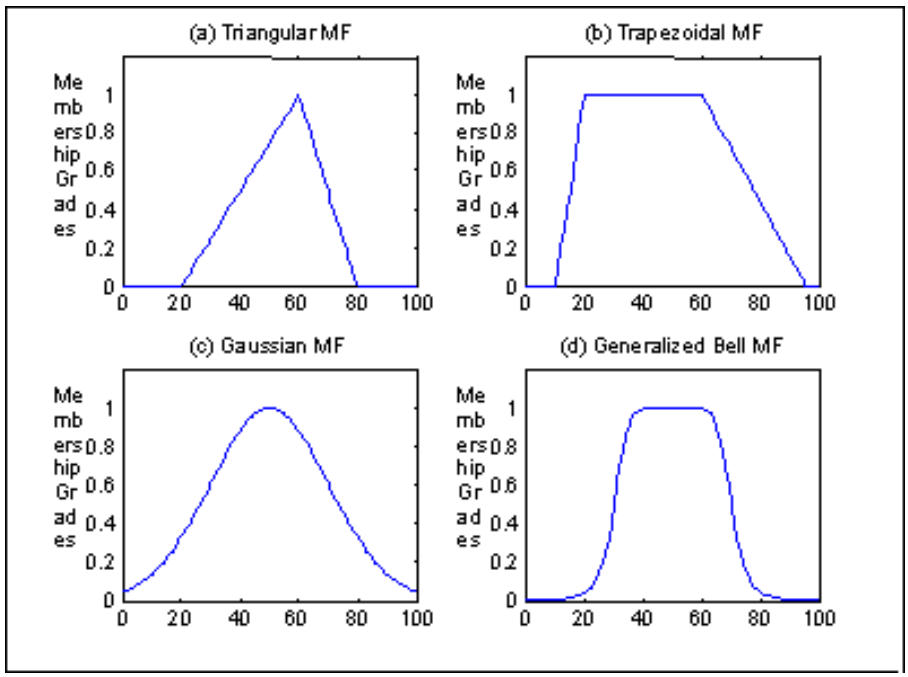

Figure 1.2: Membership function types

In the formula 1.2, $f(x)$ can be replaced with any function that may be the best for the given application. Some popular examples for the membership functions are presented in figure 1.2.

1.2.1 Fuzzy sets properties

Given the fact that fuzzy sets are an extension of crisp sets from classical mathematics, they comes with the following properties:

- Support of a fuzzy set: all elements of A with non-zero membership grade

$$supp(A) = \{x \in |\mu_A(x) > 0\} \tag{1.3}$$

- Core of a fuzzy set: all elements of A that completely belong to a class

$$core(A) = \{x \in X|\mu_A(x) = 1\} \tag{1.4}$$

- Height: maximum peak at which the function rises; a height is called normal if it is equal to 1 and subnormal if less than 1

1.2.2 Operations

Like in classical sets, fuzzy sets also define operations that can be applied.

- Union

$$U[\mu_A, \mu_B] = max[\mu_A, \mu_B] \tag{1.5}$$

- Intersection

$$I[\mu_A, \mu_B] = min[\mu_A, \mu_B] \tag{1.6}$$

- Complement

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x) \tag{1.7}$$

1.3 Linguistic variables and values

By a linguistic variable we mean a variable whose values are words or sentences in a natural or artificial language. For example, age is a linguistic variable if its values are linguistic rather than numerical, i.e., young, not young, very young, quite young, old, not very oldand not very young, etc., rather than 20, 21, 22, 23 [25]. In mode specific terms, a linguistic variables is represented by a quintuple

$$(\sigma, T(\sigma), U, G, M) \tag{1.8}$$

where σ is the name of the variable; $T(\sigma)$ is the collection of the linguistic values; U is the universe of discourse; G is a syntactic rule which generates the terms in $T(\sigma)$; and M is a semantic rule which associates with each linguistic value X its meaning, $M(X)$ where $M(X)$ denotes a fuzzy subset of U.

The concept of linguistic variable enables to approximate phenomena which are too complex or too ill-defined to be described through a mathematical model. Thus, treating a truth value as a linguistic variable with values such as true, very true, extremely true, not so true, and so on leads to what Zadeh coined as fuzzy logic [25]. By providing such a framework on which data can be processed we can get closer to human like reasoning that is more realistic and simpler to implement than the classical, mathematical model.

The fact that we can now express things as age in a more human-like manner means that systems can be defined in a human like way using values as “very young”, “young” and “not so young” which enables a system designer to make use of the knowledge that a human has.

As stated before, for each linguistic variable we have a set of linguistic values that describe the range of inputs the variable can get. The collection of values defined for a certain linguistic

variable constitutes its term-set. For each of the values found in a term-set, we define a restriction. Consider the linguistic variable "height" for which we define the linguistic values "short", "normal" and "tall". Considering that "height" can range from 30 to 250 centimeters, we need to define boundaries for each linguistic value. These represent the membership function that were discussed in section 1.2.

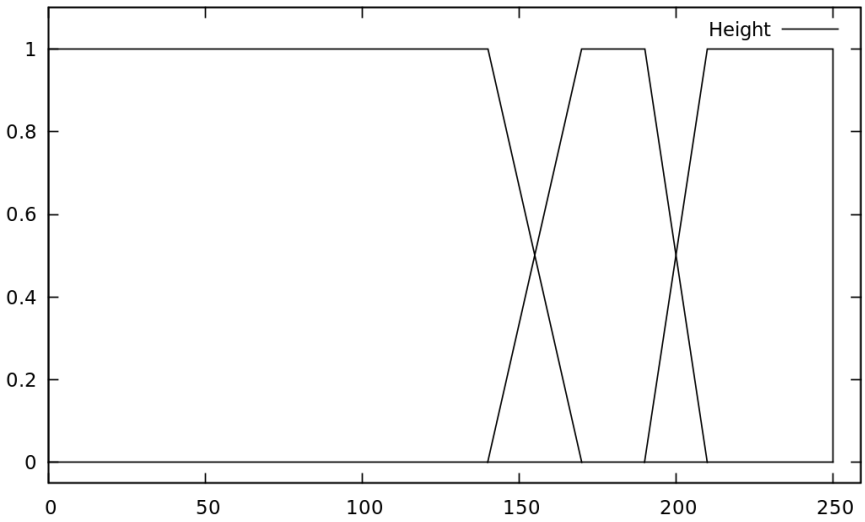

Figure 1.3: Height linguistic variable

1.4 Fuzzy relations

A relation is mathematical description of a situation where certain elements of different sets are related based on some knowledge. Fuzzy relations are an important part of fuzzy logic and fuzzy control, representing the knowledge that drives a fuzzy engine.

To explain what a fuzzy relations is, we first need to explain what a crisp relation is and extend from it. To illustrate what a crisp relation is, lets consider that we are trying to figure out how ripe a fruit is given its color. In order to reach this conclusion we need to have some previous knowledge about what color an apple turns to when it is verdant, half-ripe and ripe. We establish these rules:

- If color of apple is green then apple is verdant
- If color of apple is yellow then apple is half-ripe
- If color of apple is red then apple is ripe

Considering these rules, we can get the following table:

In the table 1.1 the values "0" and "1" represent the grade of membership of an apple with the given color to the corresponding ripeness. The crisp relation show only the absence or presence of connection but it doesn't show anything in between.

	Verdant	Half-ripe	Ripe
Green	1	0	0
Yellow	0	1	0
Red	0	0	1

Table 1.1: Crisp relation example

In contrast, for the exact same rules, the corresponding table would be 1.2.

	Verdant	Half-ripe	Ripe
Green	1	0.3	0
Yellow	0.5	1	0.5
Red	0	0.3	1

Table 1.2: Fuzzy relation example

What the tables 1.1 and 1.2 actually are, is the Cartesian product between two sets. In fuzzy logic, a fuzzy relation R is a mapping from the Cartesian space X $\times Y tothe interval[0,$ $1], where the strength of the mapping is expressed by the membership function of the relation \mu_R(x, y)$

$$\mu_R : A \times B \rightarrow [0, 1] \tag{1.9}$$

$$R = \{((x, y), \mu_R(x, y))|\mu_R(x, y) \geq 0, x \in A, y \in B\} \tag{1.10}$$

1.4.1 Composition of fuzzy rules

For composing two fuzzy rules, one might use one of the the following compositions: max-min or min-max. Lets us consider two fuzzy relations R_1 and R_2 defined on the Cartesian space X $\times Y and Y \times Z respectively.$

- max-min

$$R_1 \circ R_2 = [(x, y), max\{min\{\mu_{R_1}(x, y), \mu_{R_2}(x, y)\}\}]|x \in X, y \in Y, z \in Z \tag{1.11}$$

- min-max

$$R_1 \circ R_2 = [(x, y), min\{max\{\mu_{R_1}(x, y), \mu_{R_2}(x, y)\}\}]|x \in X, y \in Y, z \in Z \tag{1.12}$$

1.5 Fuzzy inference

Having established all the concepts surrounding fuzzy logic, we can now take it further to the actual steps that are involved in the decision making based on some given input, i.e. inference.

The inference process is based on the fuzzy sets that were defined, their membership functions and the rules which contain the actual knowledge of the system. The great thing about fuzzy logic is that the process through which the system computes a solution is by making use of some knowledge that an expert (e.g. a person) has. What is great about it is the fact that the intelligence can be expressed in a human-readable form, more specifically rules. This aspect makes it extremely easy to design such a system because a mathematical model of the problem is not necessary.

There are seven steps that to take a fuzzy logic iteration to its end:

- 1. Defining linguistic variable that will be the input for the system; e.g. height, weight etc. There are also output variables
- 2. Defining linguistic values for each of the above linguistic variables; e.g. height: short, tall, very tall etc.
- 3. Defining rules that will comprise the system's knowledge of the problem; e.g.

  "if height is short then age is young"
- 4. Fuzzify the variables and compute each one's degree of membership to its corresponding linguistic values
- 5. Compute the consequent of each rule based on its consequent variables
- 6. Combine the consequents to get an output distribution
- 7. Defuzzify the output distribution to get a crisp value

The first three steps were described in the previous sections and in the following, the next four will be presented.

1.5.1 Fuzzfication

The fuzzification process happens right after the system receives the input values for all its linguistic variables. The fuzzification is the process of taking the fuzzy input of a linguistic variable, run it through all of its membership functions and find the membership degree of to each of the linguistic values it has.

Example: Let us consider that after eating at a restaurant we may rate the food and the treatment that the customer received. And so we have the linguistic variables food and treatment

with the values bad, acceptable and tasty for food and bad, good and very good for treatment. After getting a certain rating, say from 0 to 10 for our variables and by running them through the associated membership functions of the linguistic values tasty and good, we find that the food was 0.48 tasty and the treatment was 0.57 good 1.4.

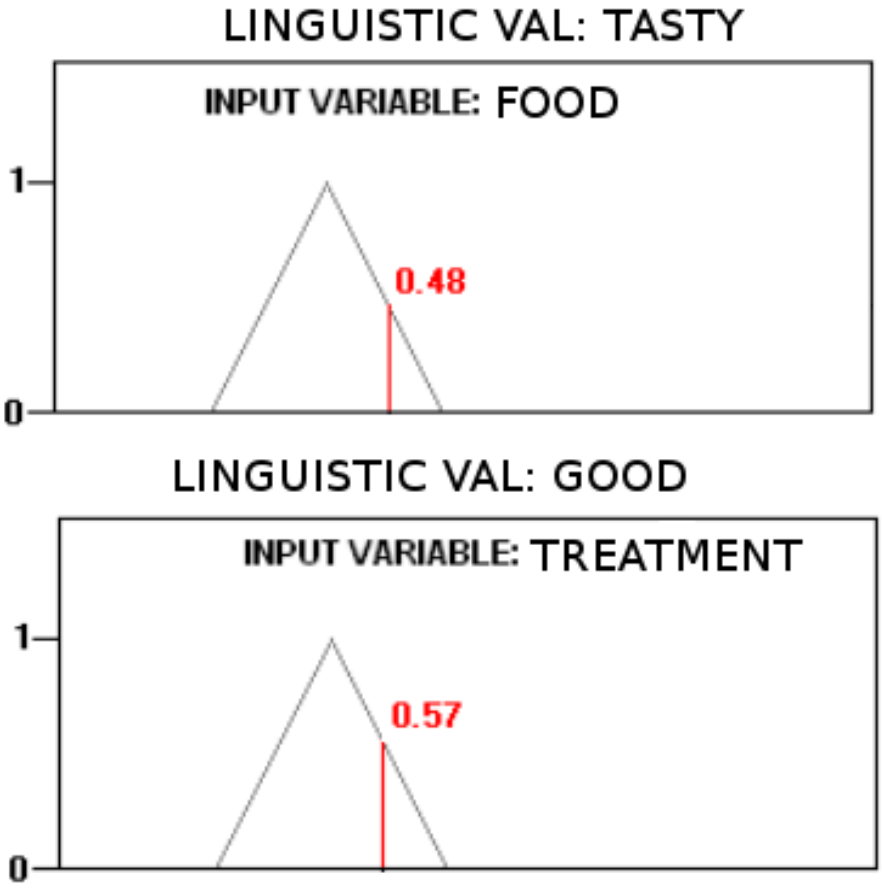

Figure 1.4: Graphical result of fuzzification

In essence, the process of fuzzification of a given set of inputs represents the computation of their membership to a certain linguistic value.

1.5.2 Rule evaluation

After having the degree of membership of all inputs to some linguistic values, the next step is going through all the rules that were defined and compute their consequent.

As stated before, because of the nature of fuzzy logic, the rules can be expressed as natural, human like rules:

$$if \quad x \quad is \quad A \quad and \quad y \quad is \quad B \quad then \quad c \quad is \quad U \tag{1.13}$$

The first part of the rule (right until “then”) is called the antecedent and the last part (after

and starting with "then") is called the consequent. The antecedent may have one or more tests that it performs and each of these tests is connected with the next through a binary fuzzy logic operator. The operators are the equivalent of the logical, mathematical operators OR, AND and NOT but have a difference definition. There are Zadeh fuzzy operators and probabilistic fuzzy operators, each with their own definition.

Zadeh operators are in the following form:

$$AND = min(\mu_A(x), \mu_B(x)) \tag{1.14}$$

$$OR = min(\mu_A(x), \mu_B(x)) \tag{1.15}$$

$$NOT = 1 - \mu_A(x) \tag{1.16}$$

while probabilistic operators:

$$AND = \mu_A(x) \times \mu_B(x) \tag{1.17}$$

$$OR = \mu_A(x) + \mu_B(x) - \mu_A(x) \times \mu_B(x) \tag{1.18}$$

$$NOT = 1 - \mu_A(x) \tag{1.19}$$

The antecedent of every rule is taken, computed and the consequent receives the resulted value. The case can present itself when we have multiple rules for which the consequent is identical.

For example:

"if food is bad and treatment is bad then tip is low"

and

"if temperature is good and treatment is bad then tip is low"

In such a case, the value that is chosen is the one for which the resulting membership function is the greatest. For example if the resulting membership function for the first rule is 0.45 and the resulting membership function for the second rule is 0.7, for the linguistic value "off" of variable heating_status will be 0.7.

Example: Considering the example from section 1.5.1, we had that food is 0.48 "tasty" and "treatment" is 0.57 good, lets consider the following rule:

$$if \quad food \quad is \quad tasty \quad and \quad treatment \quad is \quad good \quad then \quad tip \quad is \quad big \tag{1.20}$$

Given our antecedent "if food is tasty AND treatment is good" we can see that the operator is AND, thus we apply the *min* operator on the resulted membership degrees. The result is that "tip" is now 0.48 big 1.5.

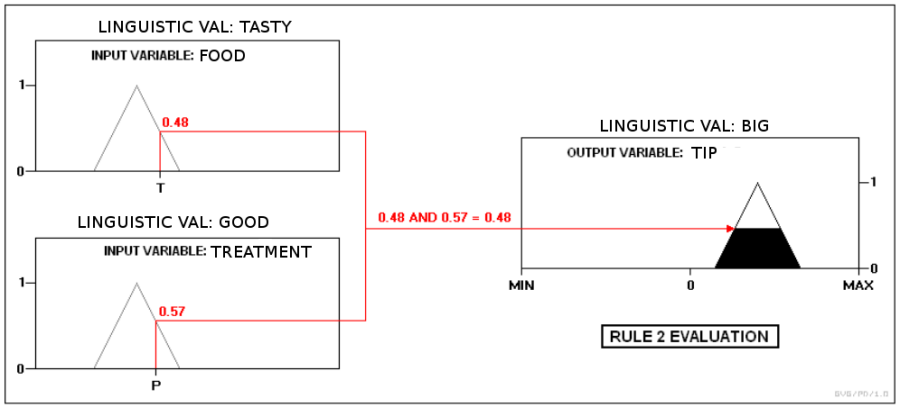

Figure 1.5: Graphical representation of rule evaluation

1.6 Defuzzification

After having computed the consequent of all the rules, it's time to take those values and turn them into a crisp value that can be used.

The result of each consequent is either a clipped version of the output linguistic variable's membership function or the membership function itself if the result of antecedent is one. To get a crisp value out of the output distribution, the resulting output membership functions are combined to form a polynomial for which the center of gravity will be calculate. The abscissa of the resulting point will represent the crisp value that is the final result of the system.

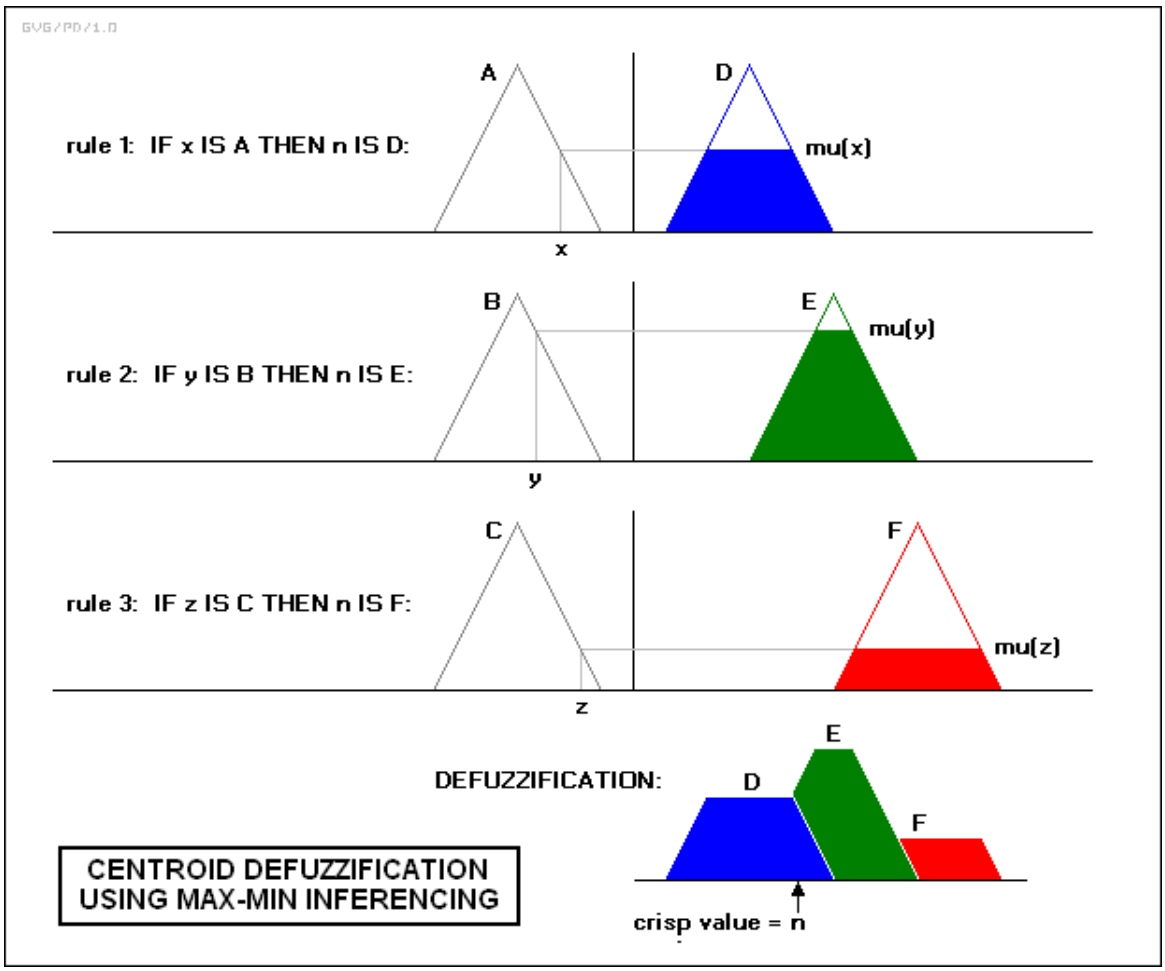

Figure 1.6: Final fuzzy engine result

Chapter 2

Conceptual model

2.1 Motivation

The motivation behind this proposal is three-fold: affordability, efficiency and connectivity.

The time we are living in presents itself with rapidly occurring technological breakthroughs that help human kind in setting up for themselves a better, more comfortable and efficient world. The greater problem concerning this is that it does not come cheap. Advanced electronics were not affordable to the general public so the vast majority of people were unable to get access to them, let alone develop "things" having them as a platform. This situation has changed in the past few years and now we have easy access to such devices enabling the contribution of everybody to the Internet of Things.

The recent years have marked an amazing rate at which new technologies are discovered and at which new technological processes of manufacturing are getting more efficient and more affordable. This created a self sustaining environment in that the newly invented electronics and machinery lead to more efficient development methods which in turn provided a more feature packed set of tools that is driving the observed rapid evolution.

Because of the above described advance, it is now possible to get access to powerful devices that are relatively cheap thus giving the opportunity to everybody to build gadgets that can be incorporated in the infrastructure of the Internet if Things.

Having access to diverse pieces of technology lets anybody create mechanisms through which data is gathered from the environment and shared on a wide network of other devices where is can be observed. The main idea behind the Internet of Things is having a large number of gadgets that collect, store and analyze data about everything and anything with the purpose of grasping a better understanding about them. The Internet of Things enables the world to research itself and find patterns that were otherwise invisible, thus leading to a greater understanding of the context we are living in, which in turn lets us control that context

in a more efficient manner.

All these connected devices mean that data can be shared and used in a collaborative manner and the possibility to control processes from afar arises. For the application in discussion, this fact enables one to monitor and control one's house temperature setting from anywhere thus increasing the comfort level. For example, going away from home, one would turn the temperature down but when coming back, it is desirable to have the house already at a comfortable temperature level. This can be achieved by being connected to your home thermostat.

2.2 Fuzzy engine overview

This section will present the way the fuzzy engine of the proposed thermostat was designed, and its various components.

2.3 Linguistic variables

The problem this application is facing is not a trivial one. The factors that surround this matter of home comfort are many and there are relationships amongst them that affect and make it harder to predict what is going to happen or decide what action should be taken at a specific moment in time.

The factors that were considered for this application are the following

- Difference between the temperature that the user desires (temperature set point) and the current, environment temperature (temperature error)
- Humidity of the environment
- Rate of heating
- Rate of cooling

2.3.1 Temperature error

Most basic thermostats that are used for controlling in-doors temperature use as their single factor of controlling a central heating unit, the difference between the temperature set point and the actual temperature of the environment. This is the basic and fundamental aspect to take into consideration when trying to keep a habitat at a constant temperature.

In the context of fuzzy logic, the target was to define for this linguistic variable (temperature error) a set of linguistic values and their corresponding membership functions that would

be relevant to the problem in question. The main idea is that the greater the temperature error is, the more drastic are be the measure that need to be taken to keep the environment in a constant state. For example, the user might want to reach a certain temperature in a short period of time, so he might set the desired temperature higher than the actual target in order to reach it faster. In contrast, the user might make a repeated, small modifications to the temperature set point in a short period of time which might demote the fact the a finer control is desired.

Another factor that was taken into consideration while designing the membership functions for temperature error was the limit at which a person might feel a difference in temperature. Studies have shown that temperature differences that are below 0.5 C are usually not felt by a person [16].

Taking everything from above into consideration, it was decided the the linguistic variable temperature error will have three linguistic, each defined by a trapezium:

- "low": trapezium defined by the abscissa coordinates: 0, 0, 1.75, 2.75
- "moderate": trapezium define by the abscissa coordinates: 1.5, 2.75, 3.75, 5
- "high": trapezium defined by the abscissa coordinates: 3.75, 5.5, 100, 100

A graphical representation of the membership functions of temperature error linguistic variables is shown in Figure 2.1.

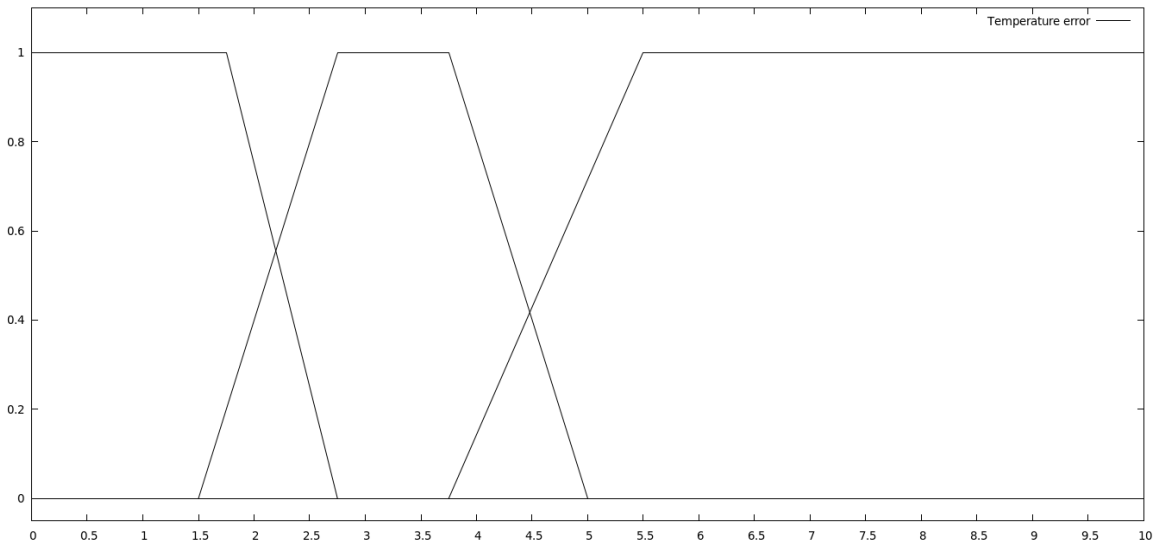

Figure 2.1: Temperature membership function

2.3.2 Humidity

Human beings are sensitive to slight temperature changes, yet cannot perceive differences in relative humidity levels within the range of 25% and 60%, which is the primary reason that this range is often cited as the baseline. If relative humidity falls outside this range, there are

notable effects. When relative humidity gets too high, discomfort develops, either due to the feeling of the moisture itself, 8 which is unable to evaporate from the skin, or due to increased friction between skin and clothing with skin moisture.

When relative humidity gets too low, skin and mucous surfaces become drier, leading to complaints about dry nose, throat, eyes, and skin. In particular, discomfort in working environments, which are prone to significant eyestrain, such as an office with a computer, is exacerbated [16].

To measure the comfort level in correlation to the temperature and humidity of the environment, an index was created that reflects this and it is called humidex.

Humidex is a measure of how hot we feel. It is an equivalent scale intended for the general public to express the combined effects of warm temperatures and humidity. It provides a number that describes how hot people feel. A rule of thumb is presented in Table 2.1 and a graphical representation of the correlation between temperature and relative humidity is shown in Figure 2.2 [5].

Humidex range	Degree of comfort
20 - 29	comfortable
30 - 39	some discomfort
40 - 45	great discomfort; avoid exertion
above 45	dangerous; heat stroke possible

Table 2.1: Humidex rule of thumb

Taking the discussed factors into consideration, the following linguistic values membership functions were defined for “humidity” linguistic variable as trapeziums:

- “low”: trapezium defined by the abscissa coordinates: 0, 0, 20, 40
- “moderate”: trapezium defined by the abscissa coordinates: 20, 47, 47, 70
- “high”: trapezium defined by the abscissa coordinates: 50, 70, 100, 100

A graphical representation of the membership functions of humidity linguistic variables is shown in Figure 2.3.

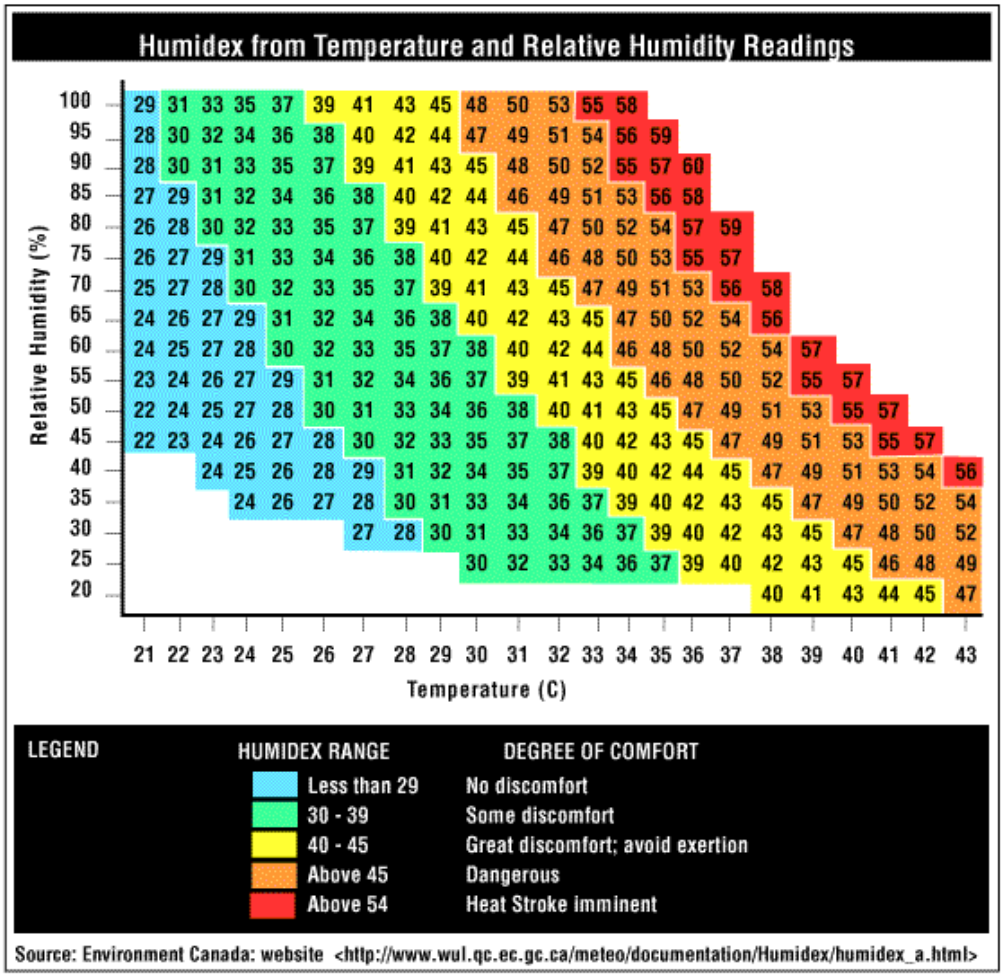

Figure 2.2: Humidex in as a relation of temperature and humidity.

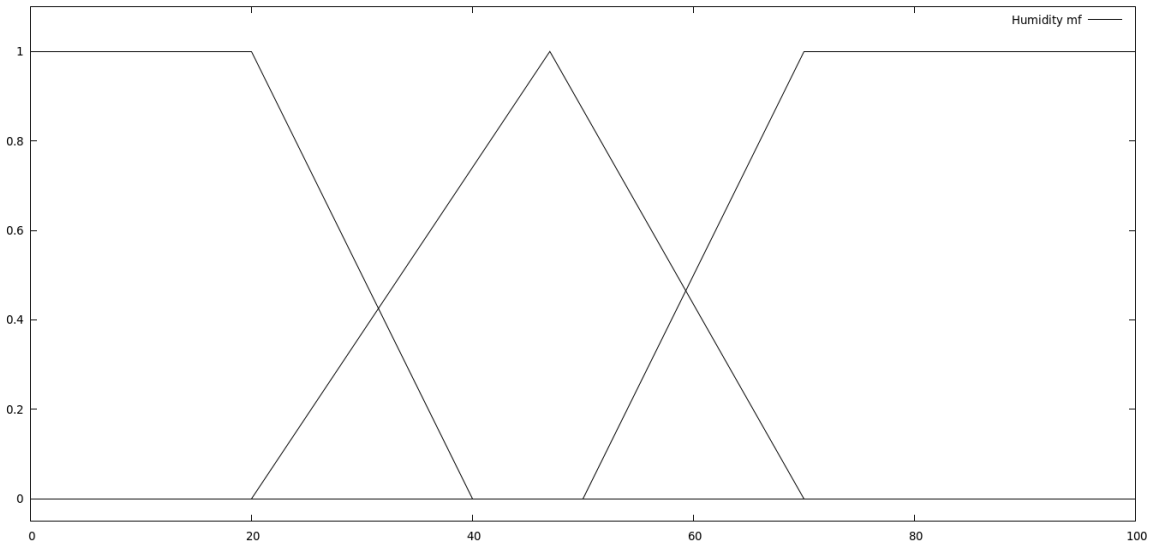

Figure 2.3: Humidity membership function

2.3.3 Rate of cooling and heating

"Rate of cooling" and "rate of heating" are two factors that predict the temperature of the environment based on the history that the thermostat experienced in the last five minutes. The inputs of both linguistic variables derive from the same computation that uses an approximation method to get a temperature trend.

While the inputs for temperature and humidity are read from the sensors, the rate of cooling and heating has to be calculated in order to get the trend of the temperature. This was done by storing and analyzing data from a certain time interval in the past. Several approaches were considered for getting a useful value from the data points:

- the difference between the last and first data point from the dataset

$$Rate of cooling = t_m - t_1 \tag{2.1}$$

- computing the average temperature difference at every measurement

$$Rate of cooling = \frac{\sum_{i=1}^{m}(t_{i+1} - t_i)}{(m-1)} \tag{2.2}$$

- using least squares regression line method to approximate a function that best describes the dataset and use that function's slope as input for the engine

$$Rate of cooling = \frac{\sum_{i=1}^{m}(x_i - \overline{X})(y_i - \overline{Y})}{\sum_{i=1}^{m}(x_i - \overline{X})^2} \tag{2.3}$$

where

$$\overline{X} = \frac{\sum_{i=1}^{m} x_i}{m} \tag{2.4}$$

and

$$\overline{Y} = \frac{\sum_{i=1}^{m} x_i}{m} \tag{2.5}$$

Experiments (Figure 2.4) showed that the last method, least squares, proved to be the most steady and relevant with the fluctuations of temperature. Even with extreme differences, the values drop back to normal in a fair amount of time.

As stated before, the inputs for both "rate of cooling" and "rate of heating" are the slope of the approximated line. The difference between the two is whether or not the slope is ascending or descending.

Figure 2.4: Values of slope with regard to temperature change

2.4 Fuzzy rules

The fuzzy engine contains a set of 64 rules that were defined using an expert's knowledge of temperature control. This section presents o few of them.

- if temp_err is low and humidity is moderate and rate_of_heating is high heat_status is off

  This is a general case where the heating should be off because the temperature error is low, meaning that the temperature that the environment exhibits is close to the temperature the user has set. Also the humidity begin moderate, the conditions are such that the humidex 2.2 would be in a comfortable range. The only parameter that is high is the rate of heating. This means that the temperature is rising at a fast rate. If all of these tests are passed, the heating should be off.

- if temp_err is moderate and humidity is low and rate_of_heating is moderate heat_status is on

  This rule states that although the temperature error is moderate and the rate of heating is moderate, which means that the temperature is slowly rising to the user's desired point, the heating status should be on due to the fact that the humidity is low which leads to a low humidex. A low humidex expresses the fact that the comfort level is low and that the temperature or the humidity should rise.

- if temp_err is low and humidity is low and rate_of_cooling is high then heat_status is on

This situation expressed the case when although the temperature error is low, the humidity is also low and the rate at which the environment cools down is high. This situation requires the heating to be on in order to remain at a constant temperature.

- if temp_err is high and humidity is high and rate_of_heating is high then heat_status is on

  It may happen that despite the fact that the humidex is high and the rate of heating is also high, the temperature error is high as well, making it necessary to turn the heating on to compensate.

All of the 64 rules are based on the same reasoning as the one that were presented above.

2.5 Defuzzyfication

After having computed the consequent of every rule from rule set of our engine, the systems goes on to compute the result, the crisp value which will be the actual decision.

The defuzzyfication process is done by computing the center of gravity for the polynomial that results from the combinations of rule consequents.

Chapter 3

Architecture

This chapter will describe the architecture of the thermostat: the components that make it up and how all of them are interacting with each other.

3.1 Overview

The thermostat architecture is made up of at least two modules: a main module, that is the central part of the setup and a second module. From now on we will refer to the main module as the central unit and the second module as reporter. When referring to the thermostat, the whole setup is considered, central unit and reporters.

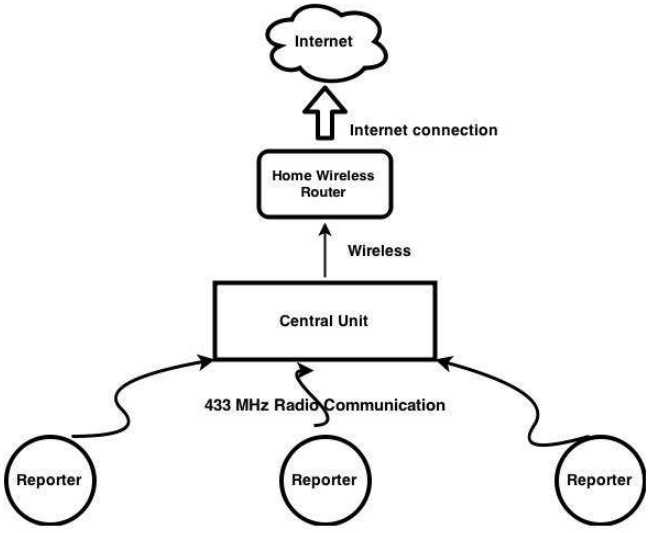

Figure 3.1: Agent based model architecture.

The thermostat setup follows an agent based model having one or multiple agents that are reactive to the environment, and that report their findings to a centralized module. In accordance to the agent based model, we have the central unit of the thermostat which acts as a central agent that receives, stores and analyses the data and one or multiple reporters that read

temperature and humidity values from the environment they are in and send it to the central unit.

3.1.1 Central unit

The central unit is the module where the data is centralized, analyzed and where the actions of turning the heating on or off happen. It is home to an Attiny85 microcontroller, a 433 MHz radio receiver, a relay, an ESP8266 and a LF33CV voltage regulator. These components will be described in-depth in the next sections.

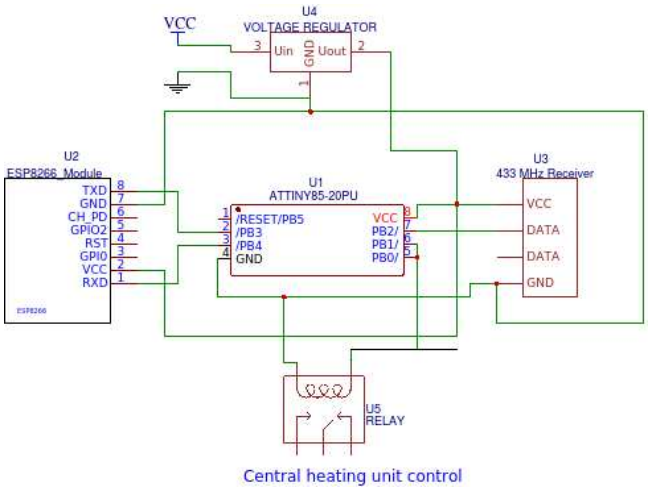

Figure 3.2: Central unit electrical diagram.

Most heating system in Northern Europe are based on a gas central heating unit that has two states: on or off. The control mechanism they employ is a standard one: there are two wires that need to make contact in order for the heating to be set to on. When the wires no longer make contact, the heating will be off. The two wires that provide the mean to control the central heating will be connected to the relay that's on board of the thermostat's central unit so it will be able to gain control over it.

The central unit of the thermostat fits on a small PCB and have a size of 7 by three point five centimeters because of it's low number of parts that come in a small package. It's size can be furthered reduced by using SMD components and by improving their layout.

The whole assembly of components can be powered with a voltage that ranges from 6 to 40 volts. That is due to on board LF33CV voltage regulator that limits the the number volts to 3.3. This voltage was chosen because the ESP8266 works at a maximum rating of 3.3 volts and the other parts were selected so they would also run at this rating. Given it's minimum and maximum ratings, the central unit may be powered by batteries or an adapter that converts 110/220 volts alternative current to 6-40 volts direct current.

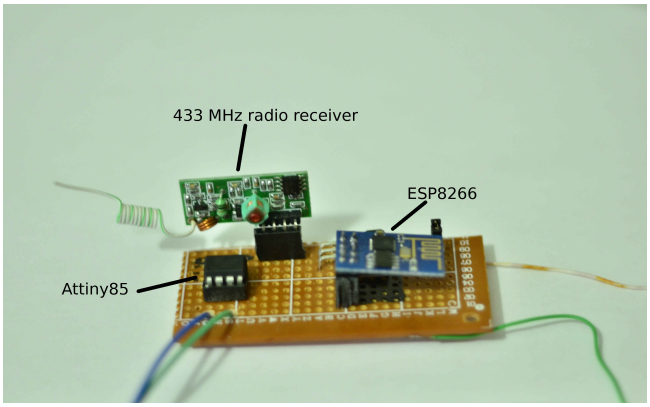

Figure 3.3: Central unit prototype on PCB.

3.1.2 Reporter

The second module of the thermostat, the reporter, it's what collects that data from the environment and sends it to central unit. The reporter is an independent module that consists of an Attiny85 (See section 3.2.3) a 433 MHz radio transmitter, a DHT22 temperature and humidity sensors and an L7805CV voltage regulator.

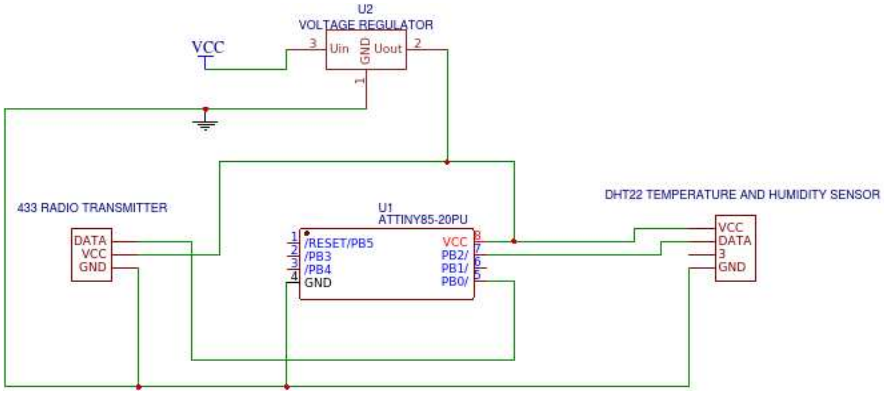

Figure 3.4: Reporter electrical diagram

The reporter is made up of components that could run on 3.3 volts but on them, the temperature and humidity sensors needs at least 5 volts for normal functioning. Also, the 433 MHz radio transmitter may work in a range of 5 to 12 volts. The greater the voltage, the greater the signal's strength will be, thus covering a greater area. The Attiny85, as stated above can work on 3.3 volts but given the ratings that the other components require, on board of the reporter there is an L7805CV voltage regulator that limits allow no more than 5 volts to pass. Given the above specifications, the following configuration was chosen: the reporter is powered by a 9 volt battery. The 433 MHz radio transmitter is powered directly from the battery while the other components (Attiny85 and DHT22) will be powered by 5 volts that is supplied through the voltage regulator.

The 433 MHz radio transmitter is well equipped to provide a range that is more than necessary for medium sized houses and it can even be placed outside, given that it is weather proof. The

9 volt battery that is powering the reporter give the transmitter enough power to have a range of about 300 meters with no obstacles in the way [21].

As stated above, the components of the reporter are in a small number thus giving the possibility to put everything in a small package. The dimensions of the PCB which holds the parts is five by four centimeters but it can be greatly reduced if SMD components are used and a better layout is chosen.

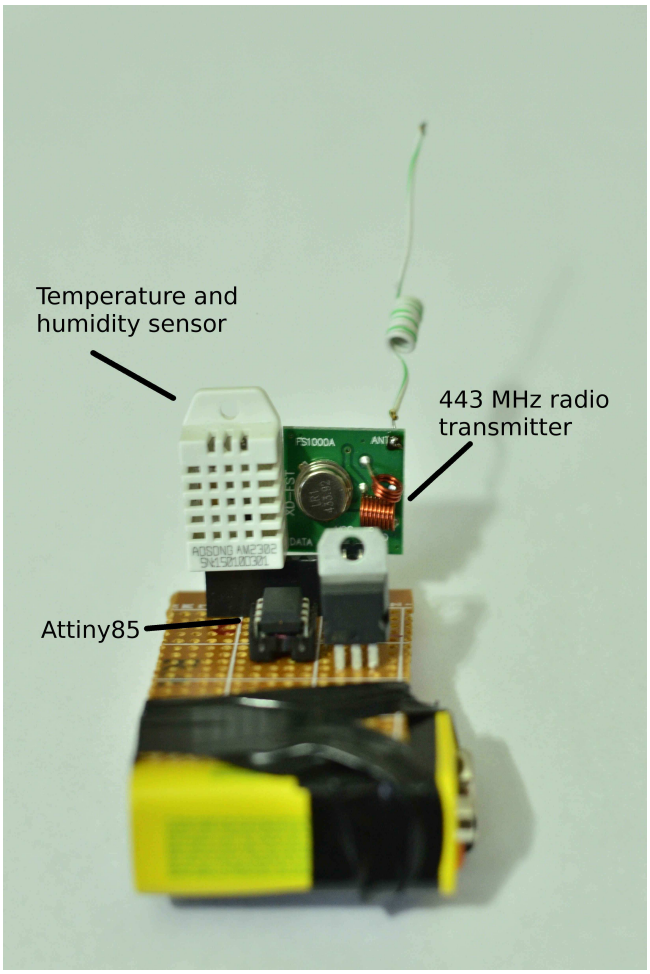

Figure 3.5: Reporter prototype on PCB.

Because the communication between the reporter and the central unit is made by radio channels, one can place such a reporter outside giving chance to a new range of possibilities including having the outside temperature as factor by which the inside temperature can be tuned.

3.2 Hardware

3.2.1 Sensors

The thermostat in discussion uses a single sensors that reports temperature and humidity. The DHT22 is temperature compensated and calibrated in accurate calibration chamber and

the calibration-coefficient is saved in OTP memory; when the sensor is detecting, it will cite coefficient from memory [8].

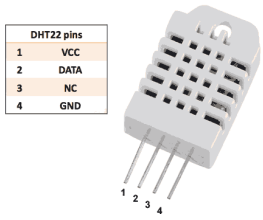

Figure 3.6: DHT22 Temperature and humidity sensor.

The DHT22 is a basic, low power sensors but precise enough to be reliable in the conditions the thermostat is designed for. The precision with which it reports the data is of 0.1 for both temperature an humidity having a rage of -40 C to 80 C for temperature while for humidity it has a range that starts from 0 and ends at 100, measured in percentage, representing the relative humidity of the environment.

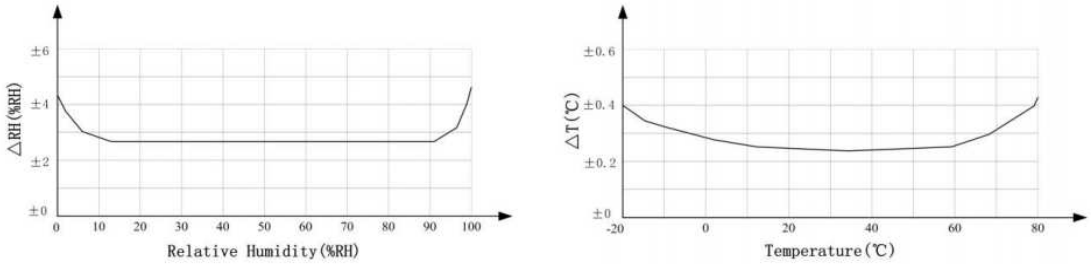

Figure 3.7: DHT22 Temperature and humidity sensor errors.

The interval at which data can be collected from this sensor is of about two seconds, which is more than enough for the purposes of this application. It uses a single-bus communication protocol that only needs one data-line. This represents an advantage since the number of I/O pins is small on the microcontroller responsible for reading the data.

3.2.2 Radio communication

The communication protocol and hardware was chosen based on two factors: low power and affordability. Given the fact that the agent-based architecture that was chosen for this applications has agents that only report data but do not need to receive any, the solution chosen comes in two packages: a transmitter and a receiver.

The radio hardware is the MX-05CV receiver/transmitter pair 3.8 that works on 433 MHz frequency. It is a low power, cheap solution that is intended for application where data needs to be carried over short distances. The radio communication uses AM (Amplitude modulation) modulation technique, meaning that the amplitude of the radio signal is varied in order the represent the data one is transmitting.

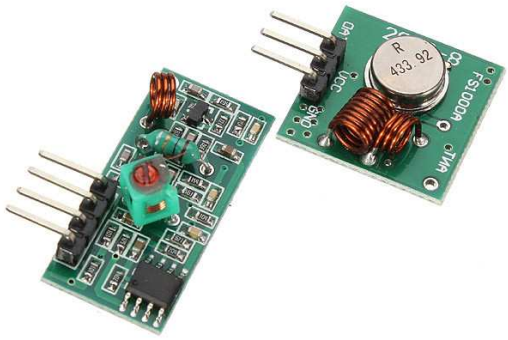

Figure 3.8: MX-05CV Radio transmitter and receiver. Image source [17]

The radio receiver works on 5 volts, direct current and takes up about 4 milliamperes of current when on stand-by which makes it suitable for low energy footprint application such as the one we're discussing. The receiver is fitted with a seventeen centimeter antenna that boosts its signal capturing capabilities.

The radio transmitter works in the range of 3.5 to 12 volts. The signal range varies proportionally to the voltage applied to the module having a transmitting power is of 10 milliwatts and a maximum data transfer rate of 4 kilobytes per second.

The cost of such a pair of radio devices comes extremely cheap and in combination with the above mentioned specifications, it makes it the perfect communication solution.

3.2.3 Attiny85

The ATtiny25/45/85 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATtiny25/45/85 achieves throughputs approaching 1 MIPS (million instructions per second) per MHz allowing the system designer to optimize power consumption versus processing speed [10].

The Attiny85 is small microcontroller that has 8 kilobytes of In-System Programmable Flash, 512 bytes of EEPROM (non-volatile memory that holds across power-ups) and 256 bytes of SRAM and it work at a frequency of 8 MHz. It has 6 general purpose I/O lines that may be used for either output or input.

Given its low power consumption and the relatively high processing power, the Attiny85 can perform non-trivial tasks like generating pulses to drive a radio transmitter to send data, collect data from a radio receiver and read various information from sensors. The Attiny85 comes in a small package, it requires no external components like a crystal clock, having one

Figure 3.9: Attiny85 AVR microcontroller. Image source [11]

internally that runs on 8 MHz, which makes it suitable for the central unit of the thermostat as well for the reporter due to its small PCB footprint.

The microcontroller comes with a sleep mode feature that enables it to go in a state where the power consumption is negligible. It features three sleeping modes, each of them disabling a further layer of its architecture thus lowering the power draw to a point where is almost non-existing. This is a nice feature that can be used in the thermostat's reporter since it runs on battery. The way if functions is that at power-up, after everything has been initialized in the memory, it tries and reads the data from the temperature and humidity sensors. After completing this task, it sends the data over radio to the central unit an it goes to deep-sleep thus preserving and prolonging the life of the battery by a significant amount. Tests [20] have shown that with normal operation on 8 MHz clock speed, the Attiny85 consumes about 10 to 12 milliamperes. Considering that a CR2032 [watch] battery contained about 200 milliamperes hour, that gives us ($200mAh/20mA$) ten hours of runtime. With the sleeping mode enabled, the microcontroller draws about 1 microamp of current, thus being able to reach four thousand hours of runtime. That is an amazing figure for the battery in discussion.

The Attiny85 can be programmed with the well known platform, Arduino by using the Arduino IDE [4]. As in all Arduino boards, the program for such a microconctroller has the following structure: a setup and a loop function. The setup function is called only once when the Atinny85 starts up and here is the place for code that initializes the runtime environment. After the function setup finishes, the function loop is called repeatedly for as long as the microcontroller has power. This pattern of programming follows the idea that your code has to do something periodically, like checking the data on some sensors or checking for the pressing of some buttons.

n the presented thermostat, the Attiny85 is used in both the central unit and reporter. In the central unit the, the microcontroller has the responsibility to read the data from the radio receiver, send that data to the ESP8266 (where data processing happens) and control the relay at the request of the ESP8266. The receptor has only two tasks, reading the temperature and humidity from the sensors and sending it via radio.

In order to accomplish all of this, the Attiny85 uses several libraries. For reading the data that the sensor sends, an Arduino secific library was used that implements the communication protocol that the DHT22 uses [1]. This library is easily can be easily ported so it can work with our setup.

For the radio communication, the Manchester encoding library [15] was used. It provides a reliable implementation of an encoding protocol that works regardless of the frequency of the microcontroller it operates on.

On the central unit of the thermostat, the Attiny85 is constantly communicating with the ESP8266 data that is receiving from the reporters and itself is receiving data from the ESP8266 regarding the state of the heating. Unlike other microcontrollers, as the one that lies on the Arduino board (Atmel Atmega328P-PU [9]), the Attiny85 does not have a hardware implementation of serial communication. In order to bring serial communication to this microcontroller, a SoftwareSerial library [6]. It enables our microchip to talk to the main processing module that lies on the central unit.

3.2.4 ESP8266

The ESP826 is a small, 32 bit wireless enabled microcontroller build onto ARM architecture. It offers a complete and self-contained Wi-Fi networking solution, allowing it to either host the application or to offload all Wi-Fi networking functions from another application processor [12]. The chip comes packed with 200 kB of ROM, 32 kB ofSRAM and 80 kB of DRAM which is more than enough to give it the role of the main processing unit of the thermostat. The module also provides the wireless connectivity through which it can be accessed and the means by which data can be published on a remote server. The chip supports 802.11 b,g and n standards and has TCP and UDP stack implemented on it.

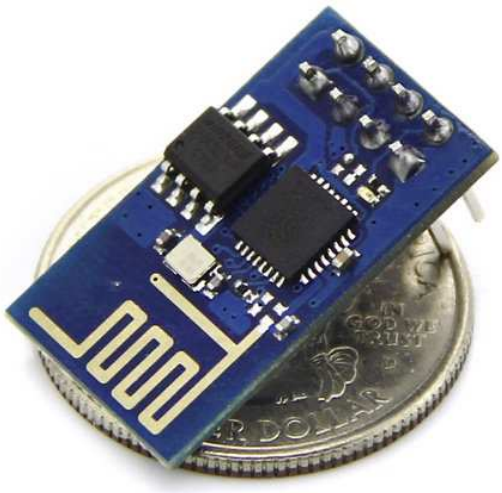

Figure 3.10: ESP8266 Wireless module. Image source [3]

Since its release, the open-source community has developed a number of programming languages

support, including Lua [19] and JavaScript [14] but, while testing, it has been found that currently the C SDK that the producer has released is the most stable and it is the one used.

The SDK provided by Espressif, the maker of the ESP8266, exposes a low-level C interface to manipulate the various features of the wireless chip [13]. The way the code is structured in the case of this chip is similar to the one from the Attiny85. There is user_init function where the system is initialized and the environment prepared for runtime and the user can create a loop for himself where the periodic tasks may take place. This is done my creating a system task that is fired in user_init and then it sustains itself by triggering the same task over and over.

```
static void ICACHE_FLASH_ATTR procTask(os_event_t *events)
{
    system_os_post(procTaskPrio, 0, 0 );
    printf( "Idle Task\n" );
}

void user_init(void)
{
    system_os_task(procTask, procTaskPrio, procTaskQueue,
                   procTaskQueueLen);
    system_os_post(procTaskPrio, 0, 0 );
}
```

In the following subsections the basic API of the ESP8266 wireless module will be described.

3.2.4.1 Operation Mode

As stated before, the ESP8266 wireless module has the sufficient features to either run a complete application on itself or to offload all the communication work of another microcontroller thus the chip can be configured in three ways: in STATION mode, SOFTAP (soft access point) mode or both STATION+SOFTAP mode. This setting can be modified by the API function $wifi\_set\_opmode(uint8opmode)$ where, for STATION, opmode has the value 0x01, for SOFTAP 0x02 and for STATION+SOFTAP 0x03.

The STATION mode allows the module to connect itself to an existing access point (AP), host a webserver that can respond to requests which are made through the connected AP and make requests of its own. When using the chip in station mode, one can use its mDNS feature to publish the IP address that the router to which it connected to assigned to it. This is a nice feature that enables the user to acquire easy access to the device by finding it on the network he is already used to by a domain name.

```
// mDSN works only in STATION mode
wifi_set_opmode(0x1);
// free the informtion related to SOFTAP
wifi_softap_free_station_info();

// allocate memory for mDNS information
struct mdns_info *info = (struct mdns_info *)os_zalloc(
    sizeof(struct mdns_info));

// set parameters accordingly
info->host_name = "thermostat";
info->ipAddr = station_info.ip.addr;
info->server_name = MDNS_SERVER_NAME;
info->server_port = 80;
os_strcpy(info->txt_data, "version = now");

// initialize, enable and register the device
espconn_mdns_init(info);
espconn_mdns_enable();
espconn_mdns_server_register();
```

Now, the webserver that is running on the ESP8266 will be available from within the network to which it connected, by accessing the URL htpp://thermostat.local. This offers a great deal of flexibility because one does not have to change APs just because one wants to get access to the thermostat

By enabling the SOFTAP mode, the ESP8266 is able to create an AP to which anybody can connect. The AP can be secured using the protocols WPA / WPA2 or WEP. A webserver can listen to requests made from this mode as well.

```
struct softap_config ap_conf;
wifi_softap_get_config(&ap_conf);

os_memcpy(ap_conf.ssid, AP_SSID, os_strlen(AP_SSID));
os_memcpy(ap_conf.password, AP_PASS, os_strlen(AP_PASS));
ap_conf.authmode = AUTH_WPA_WPA2_PSK;
// set the configuration
wifi_softap_set_config(&ap_conf);

// start DHCP server
wifi_softap_dhcps_start();
```

In STATION+SOFTAP mode the wireless module can receive request through the access point (SOFTAP) it created and at the same time it is able to sent requests by also being in STATION mode. The module will no longer be accessible, from the network it connected to, through a domain name but only through its designated IP address.

3.2.4.2 Running a webserver

The ESP8266 wireless module API comes with a basic implementation of a webserver. The way it works is that first one sets the type of connection he wants (TCP, UDP) and the port on which the server will listen and then a callback is registered that is called whenever a connection is received. The said callback is responsible for setting other callbacks: for receive data event and for disconnect event.

```
void http_recieve(void *arg, char* data, unsigned short length);
void http_disconnect(void *arg);

void ICACHE_RAM_ATTR
webserver_listen(void *arg) {
    struct espconn *conn = (struct espconn *)arg;

    espconn_regist_recvcb(conn, http_recieve);
    espconn_regist_disconcb(conn, http_disconnect);
}

void setup_tcp_listener()
{
    // wipe clean memory
    os_memset(server, 0, sizeof(struct espconn));

    LOCAL esp_tcp esptcp;

    server.type = ESPCONN_TCP;
    server.proto.tcp->local_port = WIFI_SERVER_LISTEN_PORT;
    // register on connect callback '
    espconn_regist_connectcb(&server, webserver_listen);

    // create the TCP server
    espconn_accept(&server);
    // register connection timeout in seconds
    espconn_regist_time(&server, 15, 0);
}
```

The server will accept any TCP connection that comes on the specified port wither from the AP it setup or from the AP it connected to and it will call the corresponding callbacks that were registered.

3.2.4.3 Webserver wrapper

The server implementation provided by the API is a basic one and has functionality limited to fetching the request body and its size. In order facilitate the processing of requests, a wrapper around the API's webserver was developed that makes it easier to get and interpret a request's body. The wrapper has the following API:

```
/*
 * Returns the method of a HTTP request given its request body
*/
uint8 get_http_method(char* request);

/*
 * Returns the full URL of a request given it request body
*/
char* get_request_url(char* request);

/*
 * Returns a url that has not GET parameters in it
*/
char* get_url_without_params(char* url);

/*
 * Registers a calllback function that is to be called
 * when a request comes on the URL specified by the
 * first parameter.
 * process_request_func:
 *  typedef void (* process_request_func)(
 *      struct espconn * conn, char* data,
 *      unsigned short data_length);
/*
void register_url(char* url, process_request_func func);

/*
 * Returns the value of the request parameter whose key is passed as
 * the function parameter for the given method.
*/
char* get_param(char* name, uint8 method);
```

3.2.4.4 Making HTTP requests

The API that is provided for the ESP8266 comes with functions that perform HTTP requests, given the IP of a server, and that make DNS queries in order to resolve the IP of a domain name. The presented thermostat uses a library [2] that is written around the mentioned API to make it easier to perform requests and not have to deal with the rather complicated native process. The way a request is made is by simply calling a function that has as parameters

the URL to which the request is made and a callback that will process the response.

While testing this library, it has been found that the DNS resolver cannot get the IP of the target server so modifications were brought to the library such that one could use an IP to make a request. In this configuration both the IP address of the server and the URL that would normally follow the domain name have to be supplied.

```
ip_addr_t addr;
IP4_ADDR(&addr, 184, 106, 153, 149);
http_get(url, &addr, http_callback_example);
```

3.3 Fuzzy engine implementation

Given the fact that the proposed application uses microcontrollers as its power horse, there were concerns and challenged regarding the way of code developing. The greatest concern was regarding the memory usage, both instruction code memory and heap memory, when choosing how the fuzzy logic engine should be designed. Eventually, the decision has been made to develop a new fuzzy engine, from scratch, that would be tailored for the specific problem in discussion such that no redundancies would find way into the code.

The rest of this section will provide a description of the created API.

3.3.1 Data structures

Due to the fact that the application had to written in C programming language, the building block of the fuzzy engine is the *struct* directive. All the structures that were needed to represent the data are defined as C structures and by using the *typedef* directive, are turned into types.

The following data structure were created:

- Linguistic variable

  Holds the all the data corresponding to a linguistic variable.

```
typedef struct ling_var_struct {
    char* name; // name of the linguistic variable
    double value; // the input value
    int id; // identificator
    linked_list* values; // list of linguistic values associated
                         // with the current variable
    ling_var_type type; // the type fo the variable:
                        // input or output
```

```
} ling_var;
```

- Linguistic value

  Holds all the data that associated with a linguistic value.

```
typedef struct ling_val_struct {
    char* name; // name of the liguistic value
    double a, b, c, d; // abscissa coordinates corresponding
                       // to the trapezium that the membership
                       // function describes
} ling_val;
```

- Condition

  Holds a single test that is part of a rule antecedent.

```
typedef struct condition_struct {
    ling_var* variable;
    ling_val* value;
    fuzzy_op op;
} condition;
```

- Rule antecedent

  Represents a list of objects of type condition and represents a list of tests that form an antecedent. It is inherited from linked$_list$.

- Rule consequent

  Represents the data that is associated with the second part of a rule.

```
typedef struct rule_consequent_struct {
    ling_var* variable; // output variable
    ling_val* value; // chosen value
    double result; // consequent result
} rule_consequent;
```

- Fuzzy rule

  Holds the antecedent, consequent and the resulted value after rule evaluation.

```
typedef struct rule_struct {
    rule_antecedent* antecedent;
    rule_consequent* consequent;
    double result;
} fuzzy_rule;
```

- Fuzzy engine

  This structure holds all the data from the fuzzy engine (linguistic variables, rules etc). The attribute "consequents" of the fuzzy_engine is the collection of added consequents. Each consequent is a singleton.

```
typedef struct fuzzy_engine_struct {
    linked_list* ling_vars; // linguistic variables
    linked_list* consequents; // consequents singletons
    linked_list* rules;
} fuzzy_engine;
```

3.3.2 Fuzzy engine API

In order to be able to manipulate all the structures 3.3.1 an API was defined.

This is the public API of the fuzzy engine:

```
/*
 * Allocates a fuzzy_egine structure that holds all the components
 *   of the system
*/
fuzzy_engine*
create_fuzzy_engine();
/*
 * Adds a linguistic variable to a fuzzy_engine
*/
void
add_ling_var(fuzzy_engine*, ling_var*);
/*
 * Adds a rule to a fuzzy_engine
*/
void
add_rule(fuzzy_engine*, fuzzy_rule*);

/*
 * Allocates memory for a linguistic variable and sets its name,
 * id and variable type (INPUT, OUTPUT)
*/
ling_var*
create_linguistic_variable(const char* name, int id, ling_var_type);
/*
```

```
 * Adds a linguistic value to a linguistic variable
*/
void
add_ling_val(ling_var*, ling_val*);

/*
 * Allocates memory for a linguistic value and sets the name
 * and triangular shape boundaries for it
*/
ling_val*
create_linguistic_value(const char* name,
                        double a, double b, double c, double d);

/*
 * Allocates memory for a rule and sets its antecedent and consequent
*/
fuzzy_rule*
create_rule(fuzzy_engine*, rule_antecedent*, rule_consequent*);

/*
 * Allocates memory for a rule antecedent
*/
rule_antecedent*
create_rule_antecedent();
/*
 * Adds a condition to a rule antecedent
*/
void
add_condition_to_antecedent(rule_antecedent*, condition*);

/*
 * Allocates memory for a rule consequent and sets its linguistic
 * variable and its linguistic value
*/
rule_consequent*
create_rule_consequent(ling_var*, ling_val*);

/*
 * Allocates memory for a condition and sets its variable, value and
 * fuzzy operator that will be applied between the current condition
```

```
 * and the one that will follow it
 * Example:
 *  create_condition("temp", "high", AND)
 *  create_condition("humidity", "low", NONE)
 * translates to "if temp is high AND humidity is low"
*/
condition*
create_condition(ling_var*, ling_val*, fuzzy_op);

/*
 * Outputs the engine's linguistic variables with their corresponding
 * linguistic variables and the rules
*/
void
dump_engine(fuzzy_engine*);

/*
 * Sets an input value for a linguistic variable that is identified
 * by its name
*/
uint8_t
register_value_by_name(fuzzy_engine* engine, char* name,
                       double value);

/*
 * Sets an input value for a linguistic variable that is identified
 * by its id
*/
uint8_t
register_value_by_id(fuzzy_engine* engine, int id, double value);

/*
 * Runs the whole fuzzy process
*/
 point*
 run_fuzzy(fuzzy_engine* engine);
```

3.3.3 API examples

First of all, a fuzzy$_{e}ngineobjectneedstobecreatedthatholdsallthecomponentstogether$

After initializing the fuzzy engine structure, is time to create the linguistic variables.

Example:

```
// create variable with name "tmp_err", id 1 and type INPUT
ling_var* temp_err = create_linguistic_variable(
    "temp_err", 1, INPUT);
// create values for the variable
ling_val* low_temp = create_linguistic_value(
    "low", -100, -100, 1.75, 2.75);
ling_val* moderate_temp = create_linguistic_value(
    "moderate", 1.5, 2.75, 3.75, 5);
ling_val* high_temp = create_linguistic_value(
    "high", 3.75, 5.5, 100, 100);
// add the values to the variable
add_ling_val(temp_err, low_temp);
add_ling_val(temp_err, moderate_temp);
add_ling_val(temp_err, high_temp);
// add the linguistic variable to the engine
add_ling_var(engine, temp_err);
```

After creating the linguistic variables, the only remaining step is create the rules.

A rule is created in the following way:

```
// create the antecedent
rule_antecedent* antecedent_4 = create_rule_antecedent();
// create the test conditions for the antecedent
condition* cond_41 = create_condition(temp_err, high_temp, AND);
condition* cond_42 = create_condition(humidity, high_hum, NONE);
// add the tests to the condition
add_condition_to_antecedent(antecedent_4, cond_41);
add_condition_to_antecedent(antecedent_4, cond_42);
// create the rule consequent
rule_consequent* consequent_4 = create_rule_consequent(heat_status,
                                                       heat_off);
// create the rule
fuzzy_rule* rule_4 = create_rule(engine, antecedent_4, consequent_4);
// add the rule to the engine
add_rule(engine, rule_4);
```

After the building of the fuzzy engine is done, a second step before running it is to register input values for the linguistic variables that were defined.

Example:

```
register_value_by_name(engine, "temp_err", 2.1);
register_value_by_name(engine, "rate_of_heating", 0.1);
register_value_by_name(engine, "humidity", 50);
```

after which the engine may be ran:

```
point * centroid = run_fuzzy(engine);
printf("Centroid: X: %.4f  Y: %.4f\n",
       centroid->x, centroid->y);
```

The printed result will be the point that resulted from running the fuzzy engine.

3.4 Application

Every component that we have described so far plays a crucial role in the functioning of the thermostat and they are all necessary for this purpose. All the pieces of hardware that were chosen were done so considering most of all affordability, processing power and connectivity.

In the following, this section will describe how all of the above parts are combined together to make up a functioning thermostat and how the user is interacting with it.

3.4.1 Thermostat high level components interaction

As stated before, the thermostat is composed of a central unit and one or more reporters. The central unit's purpose is to collect data from reporters, process the data and make a decision about whether or not to turn the heating on or off and it is also responsible for running the necessary software that provides the user interaction with the system.

There are two types of connections that need to be alive for the thermostat be operating as intended. First of all, the radio connection through which data is send from the reporters to the central unit. This is a key aspect that the system cannot do without. The second connection, though not mandatory, is with the home wireless router. The central unit needs to connect to an access point through which the Internet is accessible so that data can be sent to remote servers. Even without having an Internet connection, if the central unit is linked with an AP, the access to it is much more easy to achieve given the mDNS feature that the ESP8266 has 3.2.4.1. As implied, these are not necessary conditions for the thermostat to work but having them present greatly increases the user experience.

3.4.2 User interaction

After first starting up the thermostat, the user needs to set the SSID and password of his/shes home wireless router in order for the ESP8266 to connect to it. In order to do that one must connect to the AP that the central unit has created an go to the address 192.168.4.1/*wifi_setup* to set the credentials.

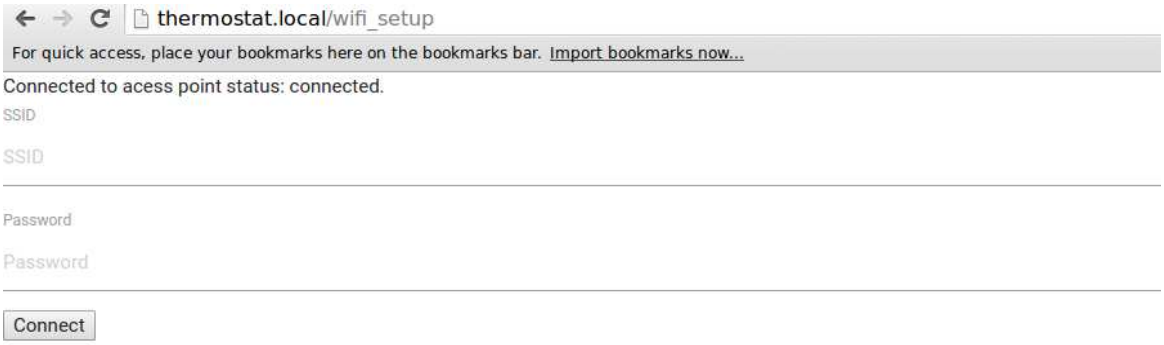

Figure 3.11: Connect to wireless router access point

After doing so, the thermostat will connect to the router, the AP it created will disappear and the ESP8266 will be accessible through the network to which it connected by going to the URL http://thermostat.local.

The thermostat provides means by which one can monitor the temperature that the reporters are recording and set a desired temperature. This is done by going the URL http://thermostat.local/control A webpage will appear where one can configure the temperature that one desires.

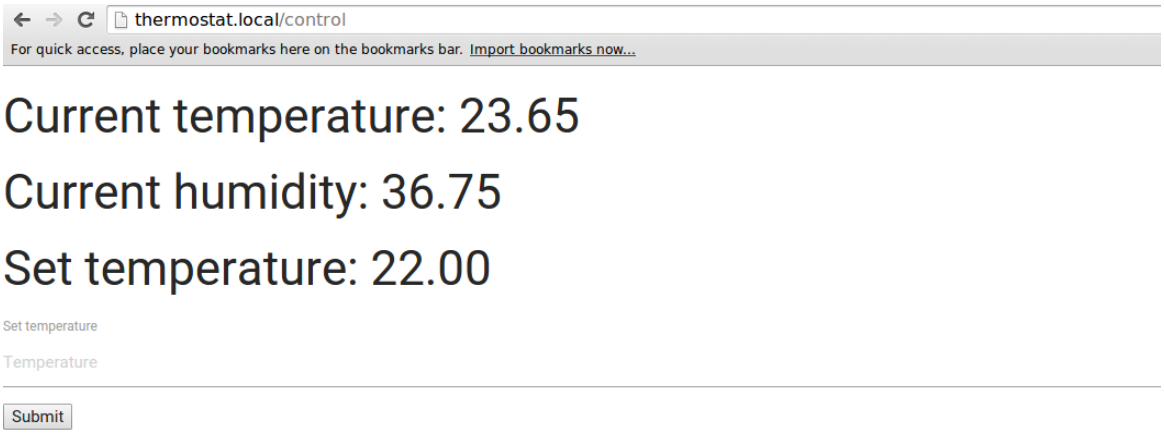

Figure 3.12: Thermostat information and control page

Chapter 4

Conclusions

Conclusions here...

Bibliography

[1] Dht-sensor-library, https://github.com/adafruit/dht-sensor-library.

[2] Esp8266 esphttpcliennt, https://github.com/caerbannog/esphttpclient.

[3] Esp8266 wifi serial module, http://is.gd/jiv6jk.

[4] How to program an attiny85, http://highlowtech.org/?p=1695.

[5] Humidex, http://is.gd/llhnw3.

[6] Softwareserial library, http://www.arduino.cc/en/reference/softwareserial.

[7] Six technologies with potential impacts on us interests out to 2025. In *Disruptive Civil Technologies*. National Intelligence Council, 2008.

[8] AOSONG, http://akizukidenshi.com/download/ds/aosong/AM2302.pdf. *DHT22 Temperature and Humidity sensor*.

[9] Atmel, http://is.gd/ef7Fyq. *Atmega328P-PU microcontroller*.

[10] Atmel, http://is.gd/wV8aIA. *Attiny85 microcontroller*.

[11] Attiny85. http://thomastesla.com/mcu_class/.

[12] ESP8266. http://espressif.com/en/products/esp8266/.

[13] Espressif, http://is.gd/H0otjK. *ESP8266 Wireless module API*.

[14] Espruino. http://www.espruino.com/.

[15] Manchester. http://mchr3k.github.io/arduino-libs-manchester/.

[16] "Warren Fincher" "Michael Boduch". Standards of human comfort.

[17] MX-05CV. http://is.gd/eubb0h.

[18] Constantin Negoita. Fuzzy sets and applications: Selected papers by l. a. zadeh, r. yager, s. ovchinnokov, r. tong, and h. nguyen (eds.), wiley, new york, 1987, 684 p. *International Journal of Intelligent Systems*, 3(2):203–204, 1988.

[19] NodeMCU. http://www.nodemcu.com/.

[20] Nathan Seidle. Low power attiny85, https://learn.sparkfun.com/tutorials/h2ohno/low-power-attiny.

[21] Texas Instruments, http://www.ti.com/lit/ds/swrs037b/swrs037b.pdf. *Low Power Sub-1 GHz RF Transmitter*.

[22] Lotfali Askar Zadeh. Fuzzy sets. *Fuzzy Logic*, 1965.

[23] Lotfali Askar Zadeh. Fuzzy algorithms. 1968.

[24] Lotfali Askar Zadeh. Outline of a new approach to the analysis of of complex systems and decision processes. *Fuzzy Logic*, 1973.

[25] Lotfali Askar Zadeh. The concept of a linguistic variable and its application to approximate reasoning-i. 1975.