UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Lauris Kruusamäe

Energy-aware Sensor Data Collection for Mobile Users

Bachelor Thesis (6 EAP)

Supervisor: Satish Narayana Srirama, PhD
Co-supervisor: Huber Flores, MSc

Author:.................................... ".....” May 2013

Supervisor:................................ ".....” May 2013

Professor:.................................. ".....” May 2013

TARTU, 2013

Abstract

Nowadays, mobile applications are becoming more context aware due to technological achievements which enable the applications to anticipate users' intentions. This is achieved through using the device's own micromechanical artifacts that can be used to perceive the environment. However, this is constrained to the hardware limitations of devices.

A proposed solution for this has been made in the thesis "Context Sensor Data on Demand for Mobile Users Supported by XMPP" by Kaarel Hanson. The solution is to use XMPP for transporting sensor data from Arduino microcontroller to the cloud. Arduino provides low-cost hardware, while the cloud offers the reliable and high- availability means for storing and processing sensor data.

This solution shows that running on a 9V battery the microcontroller lasts for 101 minutes when using an Ethernet module for communications, and 161,5 minutes with a WiFi module. These results are not good enough for remote data collection with limited access to the microcontroller.

This thesis proposes an optimisation for the system so that instead of reading and sending sensor data every 10 seconds, the cloud server would notify the controller when to start sending data and when to stop. This means implementing an algorithm for detecting similar sensor data readings and notifying the microcontroller of needed operations. With similar readings, the microcontroller could be put to an idle state for limiting power consumption, which would prolong battery life.

The aim is to optimise the sensor reading process enough to prolong the Arduino microcontroller's battery life on a 9V battery.

Contents

List of Figures

List of Tables

1

Introduction

Mobile applications are becoming more context aware and depend on perceiving the surrounding environment to provide users with the best functionality possible. This is achieved due to technological advances that enable the device to sense the surroundings, however, these abilities are limited to each device's hardware configuration.

A proposed solution has been made in the thesis "Context Sensor Data on Demand for Mobile Users Supported by XMPP" by Kaarel Hanson (2). The solution is to have special sensor modules perceive the environment and gather the measurements in a central data center, which can then be used to provide data to mobile users.

A prototype was developed based on Arduino for the sensors and XMPP for communication. This thesis proposes improvements to the existing prototype to enable sleep mode for the sensor modules and data prediction on the data server side. These improvements help prolong battery life on a 9V battery, which was measured to be 161,5 minutes with the existing prototype.

Without enough battery life, the overhead of using the modules to gather data makes the solution inefficient. Therefore, it is crucial to make the modules last longer, which is the main goal of this thesis.

1.1 Motivation

The developed prototype (2) works well when the general idea and goal is considered. However, the lack of flexibility of data transmission intervals and high power needs constrain the implementation usages. In order to take full advantage of the prototype, power usage should be reduced and more flexibility added to the communication between the server-side client and sensor modules.

1.2 Contributions

A more flexible data collection solution is developed based on HTTP. The changes made enable more energy-efficient data collection by predicting sensor data when possible. A fuzzy control system was developed to decide if sensor data is predictable and for what time period. Moreover, the prototype's Arduino-based sensor module has been improved to support sleep mode while no sensor data is sent to the server-side client. Power consumption before and after the changes was tested to indicate changes' effect on battery life.

1.3 Outline

Chapter 2: introduces the Arduino framework and modules used in the prototype. Finally, a description of fuzzy control systems and simple linear regression is given.

Chapter 3: describes the existing prototype and gives an overview of its implementation details. Later, an overview of identified areas of improvement is given.

Chapter 4: explains the improvements made to the prototype. Firstly, describes changes made to the Arduino sensor module. Secondly, changes in the communication between the sensor module and the server-side client are discussed. Lastly, an overview of the server-side fuzzy control system and data prediction techniques is given. The chapter is ended by a power consumption comparison between the initial and improved prototype.

Chapter 5: draws conclusions and summarizes the results.

Chapter 6: discusses a related paper and the thesis on which the current work is based on.

Chapter 7: points out some areas of the prototype which can be developed further to improve the results.

2

State of the Art

This chapter introduces the Arduino prototyping platform along with the components used. Secondly, as providing flexibility to a system means implementing some sort of decision making mechanism, an overview of fuzzy logic and fuzzy control systems is given. Finally, a short description of simple linear regression to briefly describe the mathematics behind linear prediction of future values based on a sample set.

2.1 Arduino

Arduino (3) is an open-source prototyping platform based on a simple microcontroller board and a development environment. It is intended for anyone interested in creating interactive solutions. Arduino can take inputs from a variety of sensors and control various actuators. The microcontroller is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino IDE enables to choose between different board models, microcontroller programmers and communication ports.

Programs (called sketches) are written and uploaded to the board using the Arduino IDE. Each sketch must have two functions- *setup*() and *loop*(). *setup*() is the first function called after Arduino is started or rebooted. It is called once and afterwards the function *loop*() is called consecutively until the board is stopped, restarted or crashes. When a crash occurs, the program is restarted, which means calling *setup*() again.

Since programs are written in C/C++, there are a lot of libraries available for use.

2.1.1 Arduino Mega ADK

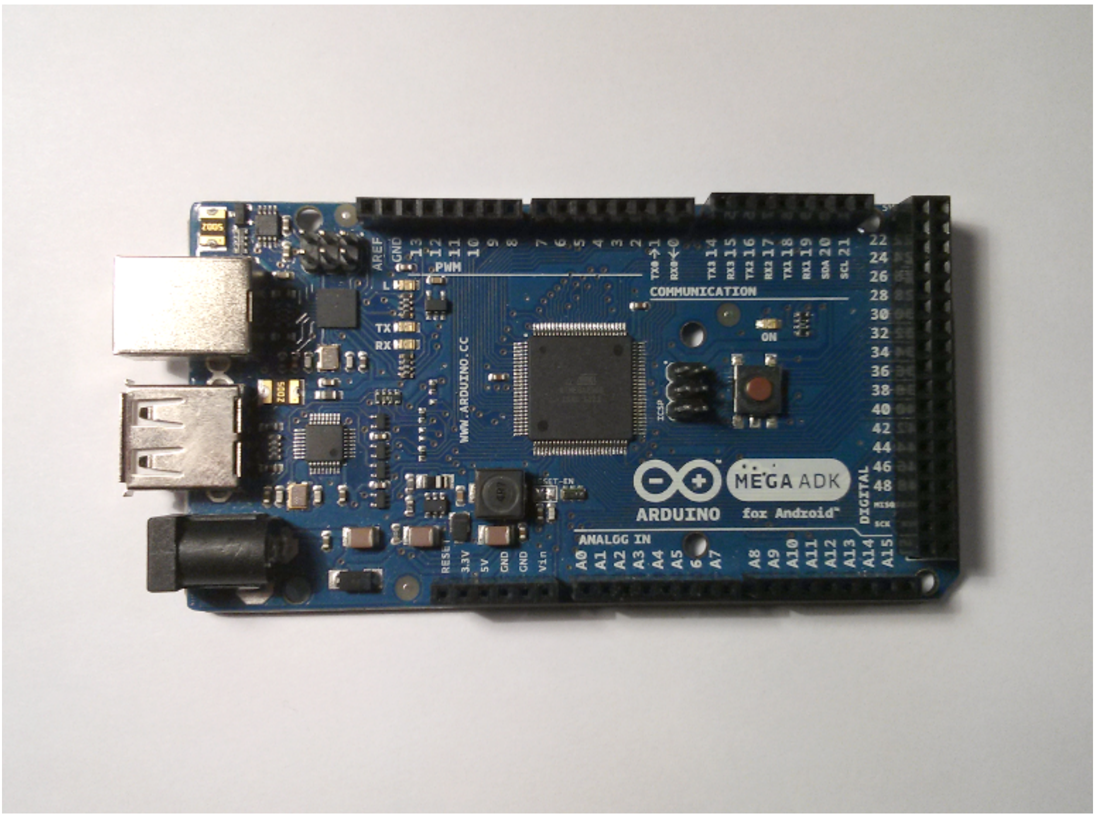

Figure 2.1: Arduino Mega ADK

Arduino Mega ADK (4) is one of the most capable boards available. The Arduino ADK is based on the ATmega2560. It has an ATmega8U2 programmed as a USB-to-serial converter, USB host interface to connect with Android based phones, 54 digital input/output pins, 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, USB B, micro B connections and a 2.1mm center-positive power jack. The ADK has 256 KB of flash memory for storing code. The Arduino ADK can be powered via the USB connection or with an external power supply. The board can operate on an external supply of 5.5 to 16 volts, recommended range is 7 to 12 volts. The board can be seen in Figure 2.1

2.1.2 Wireless SD Shield

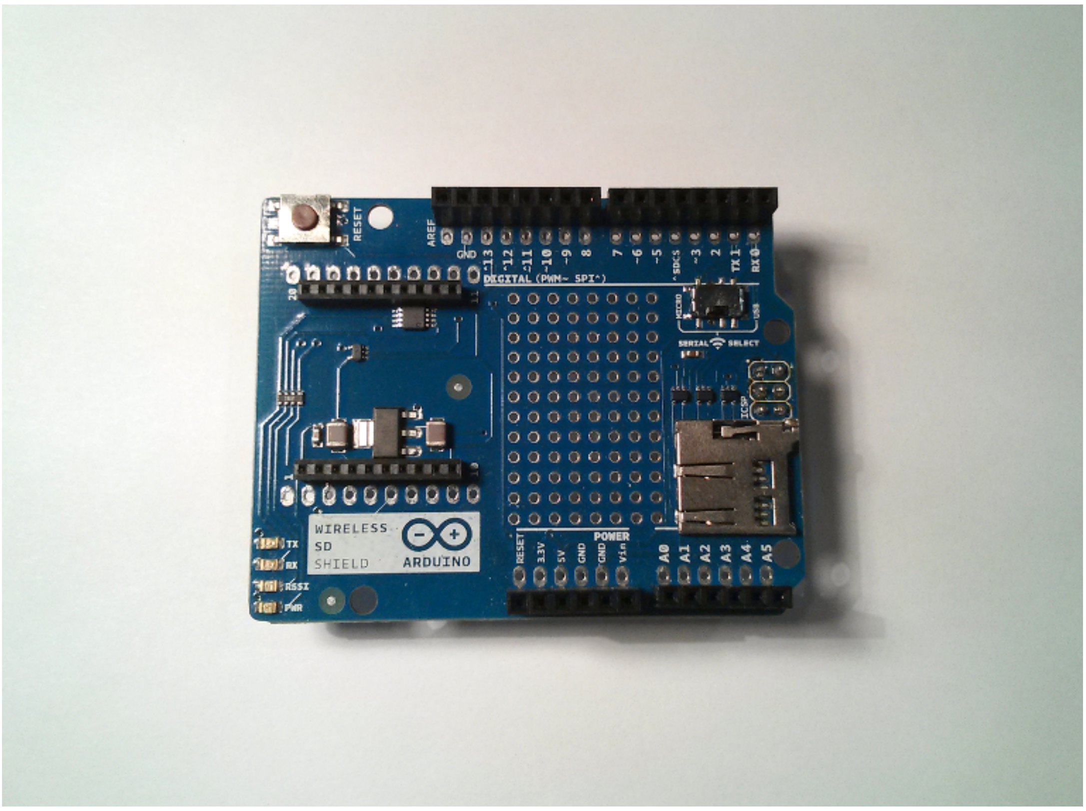

Figure 2.2: Wireless SD Shield

The Wireless SD shield (5) gives an Arduino board wireless communication capabilities. The module can communicate up to 100 feet indoors or up to 300 feet outdoors. The module can be seen in Figure 2.2

The shield has an on-board switch which allows to select between USB and Micro modes. In USB mode, the shield bypasses Arduino board's microcontroller and communicates directly to the USB-to-serial converter. In Micro mode, data sent from the microcontroller will be transmitted via USB and by the wireless module at the same time. The microcontroller is not programmable via USB in Micro mode.

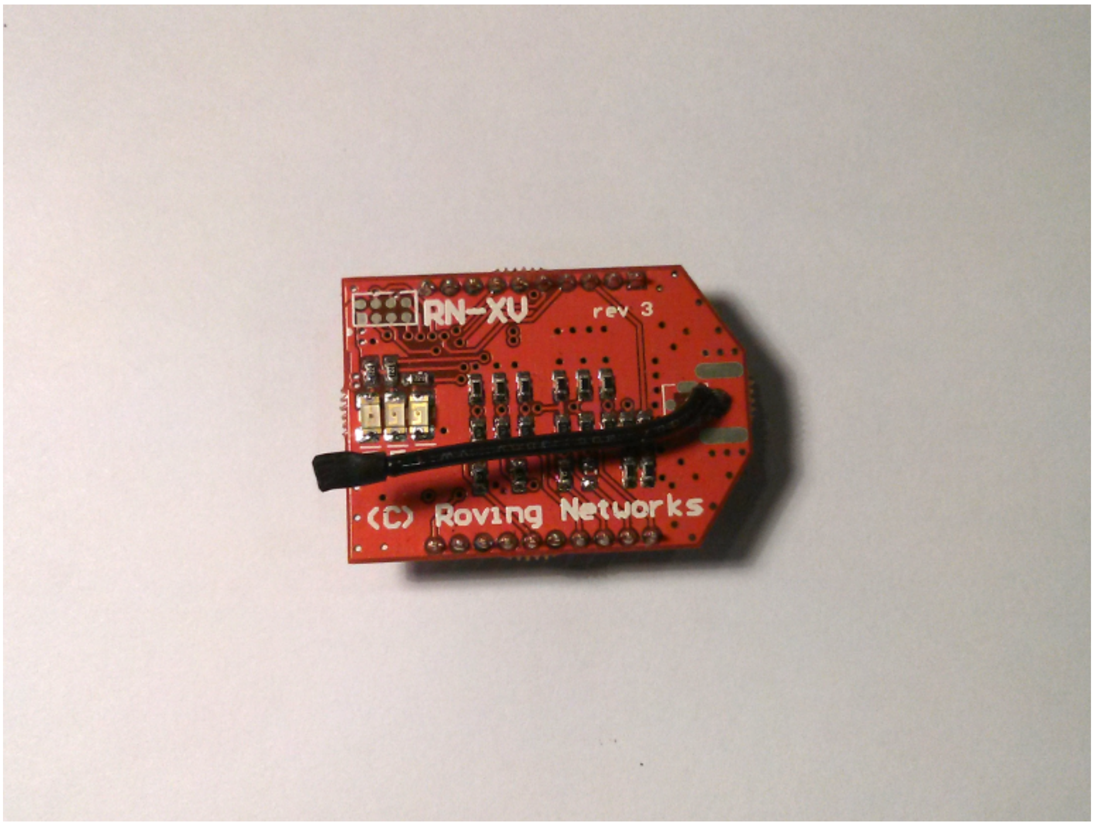

Figure 2.3: RN-XV Wifly Module

2.1.3 RN-XV WiFly Module

The RN-XV module (6) is produced by Roving Networks. It is based upon their RN-171 Wi-Fi module and has 802.11 b/g radio, 32 bit processor, TCP/IP stack, real-time clock, power management unit. It has a pre-loaded Roving firmware to enable simple configuration and usage. The module can be seen in Figure 2.3

2.1.4 TinkerKit

TinkerKit (7) is a tool to simplify interactive product prototyping using Arduino boards. It consists of modules (sensors, actuators) and a sensor shield. The tool greatly simplifies product assembly, because instead of building circuits out of low level components, all the modules can be attached to the TinkerKit sensor shield with a snapping cable.

The modules are divided into sensors and actuators. Sensors can measure their surrounding environment (provide input) and actuators perform actions (provide output). The sensors used are thermistor, LDR and Hall. The fully assembled module is shown in Figure 2.4

Thermistor sensor (8) is a resistor whose resistance changes significantly with temperature. The module's output ranges from 0V to 5V

Hall sensor (9) creates a voltage related to the magnetic field near the sensor. It can detect the presence of a nearby magnet or magnetic field induced in a coil or wire. The output ranges from 0V (no presence) to 5V (presence detected).

Light Dependent Resistor or LDR (10) is a variable resistor whose resistance is decreased when light falls on the sensor. The module's output is 5V when the module receives no light and 0V when bright light falls on it.

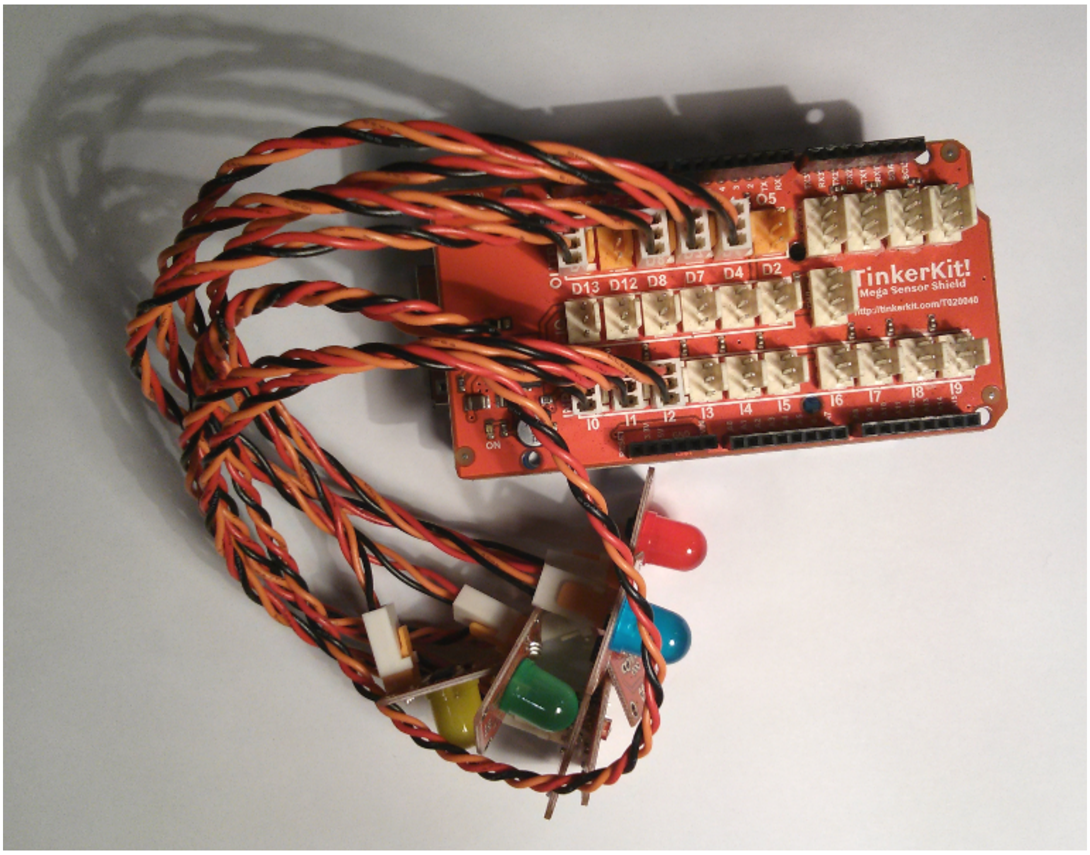

Figure 2.4: Complete Arduino Sensor Module

2.2 Fuzzy Logic

Fuzzy logic is a form of logical reasoning which is based on approximate instead of exact reasoning. When traditional reasoning has binary values like true (1) and false (0), fuzzy logic has values ranging from 0 to 1. These values represent a partial truth or in other words the degree of truth. This is similar to probability which also describes partial truth. The difference between probability and fuzzy logic is that whereas fuzzy logic describes how true a statement is, probability describes how likely it is for the statement to be true.

2.2.1 Fuzzy Set

Fuzzy sets (11) are the main building blocks of fuzzy logic. In traditional set theory where an element can either belong or not belong to a set. With fuzzy sets, an element has a degree of membership to a set.

Fuzzy sets are represented by membership functions, which measure the degree of membership (DOM) a given value has to the set. Membership functions are usually depicted graphically and have names based on the shape of their graphical representation. Membership functions are depicted as 2-dimensional graphs, where the x-axis represents the values and y-axis the degrees of membership (ranges from 0 to 1). The degree of membership for a given value can be found by projecting vertically to the upper boundary of the membership function.

The most used membership functions are triangular and trapezoidal functions. Sample sets are shown in Figure 2.5. Based on these graphs the DOM can easily be seen. Furthermore, from this graph can be seen the difference between fuzzy sets and traditional sets. Let us assume that the values represent age, trapezoidal membership function describes young people and triangular describes middle-aged people. Now when we have an age value that is between points X_2 and X_3, this value belongs to both the young and middle-aged sets. The difference is in the value of DOM to either set. When the value is nearer to point X_3 than to X_2 the statement that it belongs to the middle-aged set has a higher truth value.

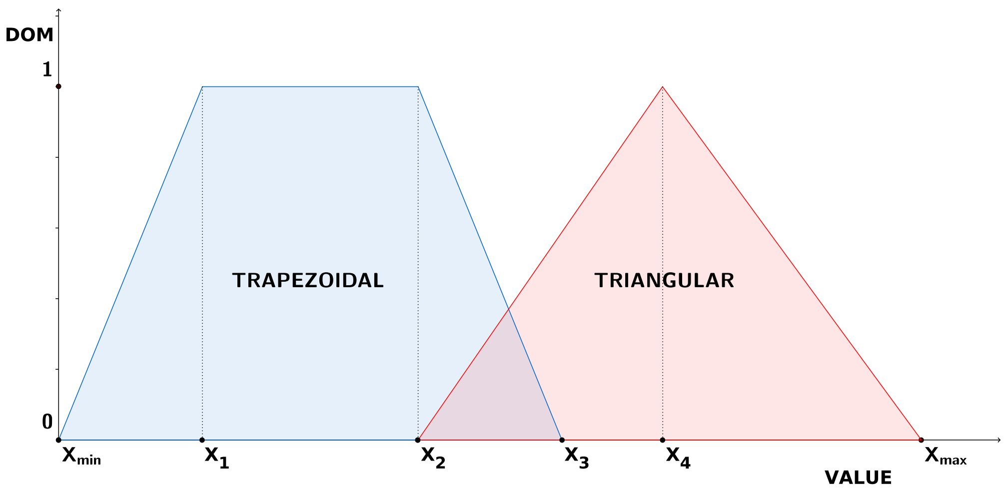

Figure 2.5: Trapezoidal and Triangular Membership Functions

2.2.2 Fuzzy Set Operations

Fuzzy set operations (12) are a generalized version of standard set operations. The most widely used generalization is referred to as the standard fuzzy set operations. This generalization includes three main operations - complement, union and intersection. Lets assume we have two fuzzy sets with membership functions A and B, then these operations can be defined as:

Complement: $\neg A(x) = 1 - A(x)$
Intersection: $(A \cap B)(x) = min\{A(x), B(x)\}$
Union: $(A \cup B)(x) = max\{A(x), B(x)\}$

2.2.3 Fuzzy Control Systems

Fuzzy control systems (11) take a heuristic modeling approach to describing the controllable domain. The heuristic model is based on a set of rules in the form: **if** $<$ *condition* $>$ **then** $<$ *action* $>$.

With the fuzzy logic part, the control system has a set of input variables and output variables, which are mapped into fuzzy sets. These variables and fuzzy sets have linguistic variables called labels.

These fuzzy sets and rules combined give us a control system which consists of a set of rules where conditions and actions are defined as fuzzy values. For example if we have an input variable *temperature* with a fuzzy set labelled *cold* and an output variable *heat* with a fuzzy set labelled *on*, we could write a following rule: IF $temperature = cold$ THEN $heat = on$. Based on this rule, the system can decide that if the temperature is cold then the heat should be turned on.

In an actual system there are multiple rules defined. As rules are activated only if their condition holds, the process of adding rules and variables has little computational overhead making the rulebase easily extendable.

Fuzzy control systems are mostly based on experience and experiments rather than actual mathematical modelling. Furthermore, as input and output variables are mapped into fuzzy sets, which have linguistic labels, the system's control rules are easily understandable. This makes fuzzy control systems usable in cases where the actual users have little knowledge of the implementation details. Instead, they can focus on the control process by defining understandable variables, their fuzzy sets and rules based on these sets.

2.2.3.1 Fuzzy control process

The decision making process has 3 main steps - fuzzification, rule evaluation and defuzzification. These steps assume that the rules, input and output variables have already been defined. An overview of a fuzzy control process can be seen in Figure 2.6

The first step in the process is fuzzification. In this step, the system's input and/or output variables are mapped into fuzzy sets. Each input variable can have one or more associated fuzzy sets. What this process does is decompose the crisp input values into fuzzy values. These fuzzy values are expressed by linguistic terms (labels) which can then be used in the rule evaluation process.

The next step is to use the fuzzified values in rule evaluation. A set of predefined rules is evaluated based on the fuzzy values created during the fuzzification process. Rules are activated when their condition holds, otherwise they are skipped. After all rules have been considered, the activated ones are gathered together. The results or actions of these rules are the basis of the next step called defuzzification.

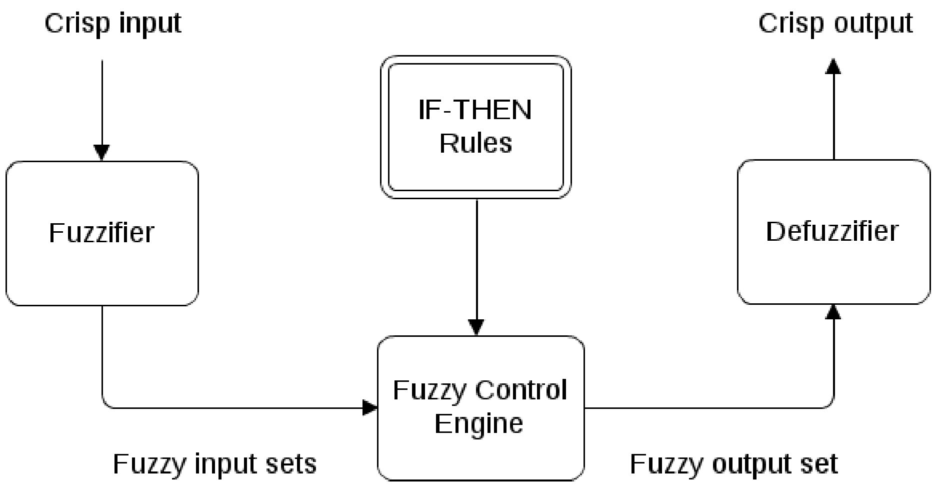

Figure 2.6: Fuzzy Control Process (1)

Defuzzification is the final step in the fuzzy control process. This part is responsible for translating the fuzzy values (linguistic terms) into a crisp quantifiable result. Usually in the previous step, rule outputs will be linguistic terms (fuzzy sets) and corresponding degrees of membership for a defined output variable. In the defuzzification part, the different fuzzy sets and their DOMS for the output variable must be added together somehow to produce a final output.

There are several defuzzification methods. The simplest of which would be to use the fuzzy set with the highest DOM as the final result. However, center of gravity is the most commonly used method. First, the different degrees of membership calculated from rules must be combined for each output variable's fuzzy sets. Then, the membership functions are depicted on a graph and for each fuzzy set, the top of the membership function is cut off in a straight horizontal line at the height of the DOM value. When this is done for all fuzzy sets of the output variable, the center of gravity for the remaining shape is calculated. The center point's projection on the value axis (usually the x-axis) is the crisp result.

2.3 Simple Linear Regression

Linear regression uses a linear model to predict future values based on a sample of previous measurements. Simple linear regression is a linear regression model that uses least squares fitting (least squares method) (13) to determine the placement of the linear function line based on the given sample values. The model has one explanatory (the x-axis) variable, based on which the future values of the predicted variable (y-axis) are calculated.

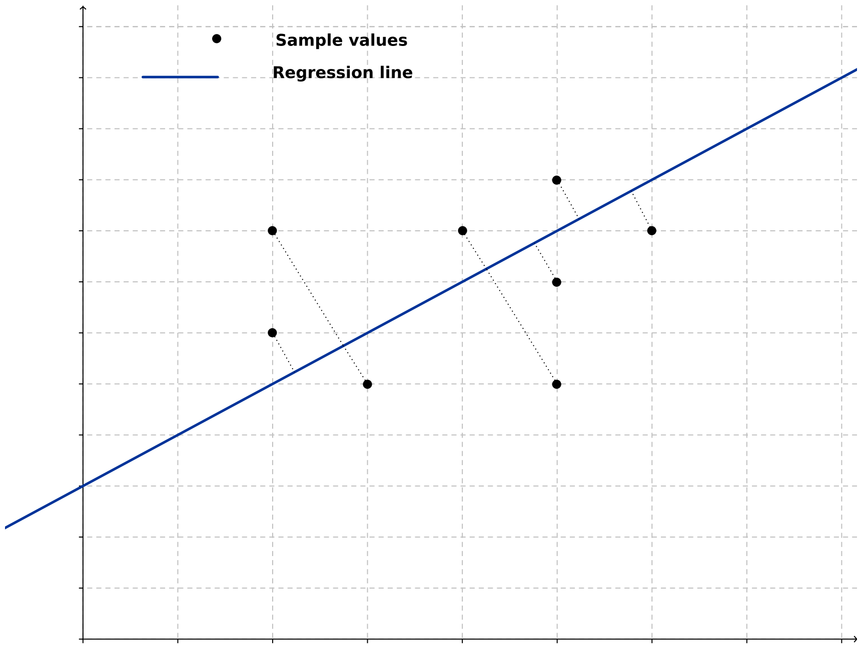

Figure 2.7: Least Squares Fitting Regression Line

Least squares fitting uses a simple approach to determine the placement of the regression line. The line placement is determined by minimizing the sum of the squares of the residuals. Residuals are the difference of the predicted values and actual values in the sample. A sample regression line can be seen in Figure 2.7. From there it can be seen, that the regression line fits in the middle of the sample points so that their residuals (depicted by the dotted lines) would be minimal.

3

Problem Statement

In this chapter, the prototype developed in Context Sensor Data on Demand for Mobile Users Supported by XMPP (2) is described. The prototype was successful, but had some areas of improvement. Secondly, an overview of these possible improvements is given.

3.1 Current Solution

The current solution (2) has three main components:

1. Arduino sensor module
2. OpenFire XMPP server
3. Data collection server in the cloud

There were two separate configurations described in Context Sensor Data on Demand for Mobile Users Supported by XMPP - one using Wi-Fi and the other Ethernet for network communication. Only Wi-Fi configuration is considered in this thesis due to the fact that the availability of an Ethernet connection (a cable) usually means that there is a power outlet nearby.

3.1.1 Arduino

The first component is the Arduino sensor module. The hardware configuration is based on the Arduino Mega ADK board. Wireless Shield with RN-XV WiFly module is mounted on top of the board. Wireless Shield and Arduino board communicate over UART (hardware serial). TinkerKit Mega Sensor Shield is mounted on top of the Wireless Shield with 7 modules attached to it: 4 LED indicator lights, Hall, thermistor and LDR sensors. The external power source used is a 9V battery.

On the software side, the implementation mainly relies on a XMPP library and WiFly module library called WiFlyHQ. The WiFlyHQ library is responsible for creating a TCP connection and communication over the network, the XMPP library handles all the XMPP implementation details.

The sketch itself is fairly straightforward - in *setup*() a wireless network is joined, a TCP connection established and lastly an XMPP session is initialized. In *loop*() all connections are checked and reestablished if needed. Then the last transmit time variable is compared to the current time and if the report step amount (currently 10 seconds) has passed, data is collected from the sensors, formatted to a JSON string and sent to the server-side client. A sample data string:

```
SensorData{"location": 1, "data": [{"type": 1, value: 5.00},
{"type": 2, "value": 500.00}, {"type": 3, "value": 312.00}]}
```

3.1.2 XMPP Communication

Both the data collection server and Arduino module are XMPP clients. The XMPP OpenFire server runs in the cloud and provides XMPP communication to both clients. Clients connect to the same chat where the server listens for messages from the sensor module. When a message is received by the server, sensor data is parsed from it and saved in a database.

XMPP session lifecycle can be seen in Figure 3.1. With the current implementation, steps 1 - 5 are done once when establishing the connection or when the connection drops. Data transmission step is done every 10 seconds and the last 2 steps are done when the connection is closed.

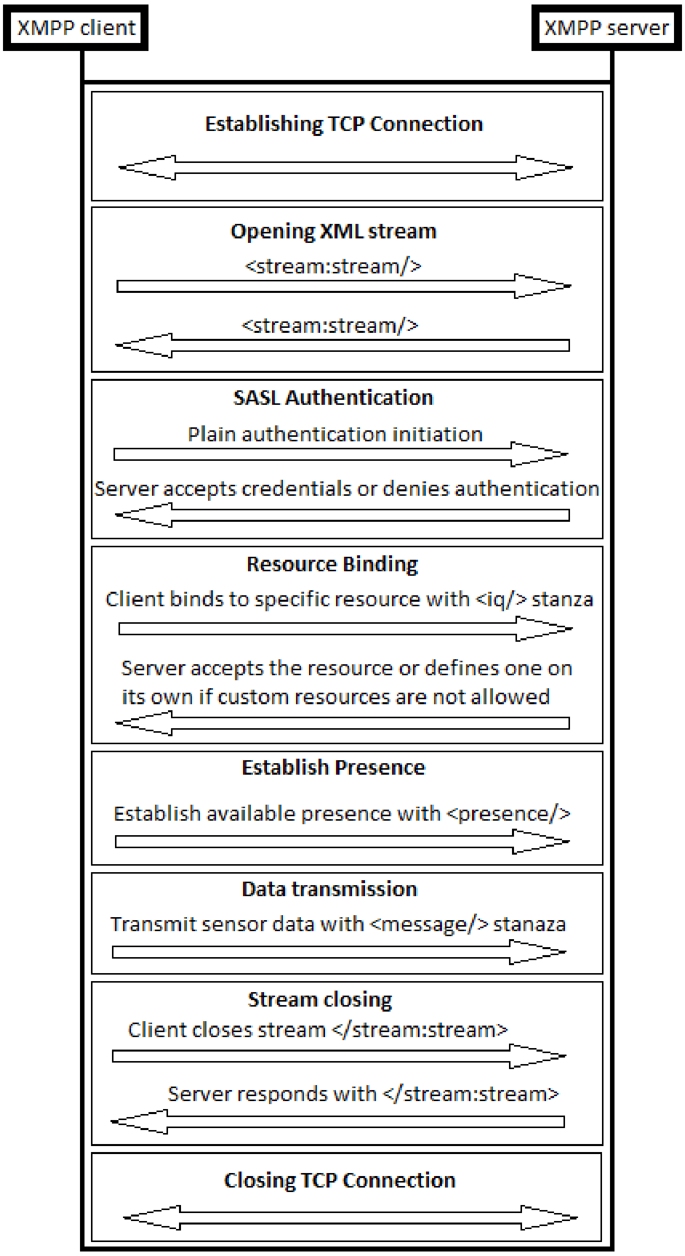

Figure 3.1: XMPP Session Lifecycle (2, p. 40)

3.1.3 Data Collection Server

The data collection server is responsible for gathering sensor data and saving it in a database. Its implementation is written in Java and uses Smack XMPP API to communicate over XMPP. Data is stored in an H2 database because of its simplicity and suitability for prototyping.

The database has three main tables - sensors, data and locations. Locations table has different sensor module locations, sensors table has different types of sensors and data has measurements gathered from different locations and sensors. The data model can be seen in Figure 3.2.

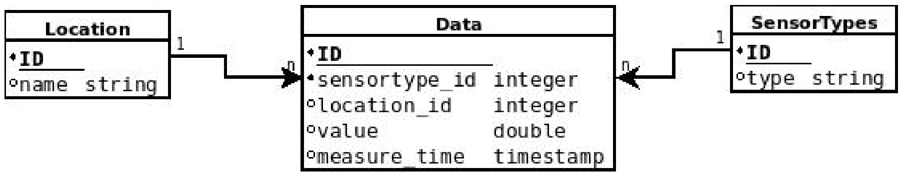

Figure 3.2: Server-side Data Model (2, p. 46)

3.2 Problems

Previously mentioned implementation has a few problems which will be discussed in this section. The two main points are power consumption and data collection flexibility.

According to the tests run in the previous thesis, the battery is able to power the module for 161,5 minutes using Wi-Fi. (2, p. 50-51) This, however, is not sufficient to enable actual data collection from a remote location. The problem can be addressed by using a larger battery, but the actual power consumption should be optimized, too. Additionally, the 10 second data transmission interval is hard coded into the Arduino sensor module, which does not provide enough flexibility. The following chapter describes changes to the hardware, software and communication methods which improve on these error points.

3.2.1 Hardware

The hardware configuration has two main problems. Firstly, communication over the wireless module can be unstable at times as parts of the messages might be missing or scrambled when read from the UART (2, p. 47). Since XMPP session initialization is quite verbose, the possibility of receiving scrambled or incomplete messages effects the process.

Secondly, all parts of the hardware configuration are run in full power mode. As there is a 10 second gap between data transmission and sensor reads, there is a time period when most components are idling. This, in turn, means that some parts of hardware could use less power during these intervals and therefore reduce overall power consumption.

3.2.2 Software

With the existing implementation *loop*() is called consecutively and time differences are checked to determine when 10 seconds has passed using the internal *millis*() function, which gives the current Unix time. Since we have a 10 second interval, we do not need to waste cycles on time difference checks we now will fail for the next 10 seconds. A way to delay program execution could be used to stop the execution until we know the defined amount of time has passed.

In addition, as messages from the Wireless Shield might not be complete and XMPP session initialization is a verbose process, the current XMPP implementation hangs when scrambled messages are received during session start up. The library checks from complete XML tags, but when the attributes are incomplete, connection will not be successful and the session initialization hangs.

3.2.3 Power Consumption

Power consumption is the main problem this thesis focuses on. As a 9V battery has an average capacity of 400 mAh at 100 mA current, the resource is quite limited. With the existing configuration, the average current drawn is around $111mA$ as seen in **??**. Battery lifetime tests in the previous thesis showed that the module runs for 161,5 minutes on a 9V battery (2, p. 50). However, 161,5 minutes is not enough to gather contextual data for the proposed data collection system. The batteries need to be changed too often for it to be a viable solution.

4

Towards an Energy-aware Solution

In this chapter improvements to the existing implementation are discussed. First, changes to the Arduino sensor module are described. Secondly, an overview of changes in the communication between the module and data collection server. Lastly, an overview of the implemented fuzzy control and data prediction systems is given.

4.1 Overcoming Power Consumption Issues

The first problem addressed is power consumption. There are two main ideas behind reducing it - put the sensor module to sleep mode when not transmitting data and reduce the need for data transmission. For this, the server-side client was improved to predict sensor data when possible and notify the sensor module of the next data transmission time. This enables the module to enter sleep mode for the given time period and thus reduce power consumption.

4.1.1 Sleep

The module consists of 3 components - Arduino Mega ADK, Wireless SD Shield with WiFly module and TinkerKit Sensor Shield. Both the Mega ADK and WiFly module support sleep modes, however, TinkerKit Sensor Shield does not.

4.1.1.1 Watchdog Timer

A watchdog timer (14) is an electronic timer used to recover from computer malfunctions. They are found in automated systems where human interference is not possible and therefore the system must be able to recover from malfunctions on its own. A watchdog timer essentially performs a timing function producing a delayed response to an input trigger. The most common implementation has a digital counter that counts from a specified value down to a terminal value. Usually the initial value is programmable. When the counter reaches the terminal value, the timer timeouts and triggers a timeout signal. Usually this means restarting the program from the start. A program can restart the watchdog timer at any time. The act of restarting is usually referred to as "kicking the dog". In this way, a program can be written which never lets the counter reach the terminal value.

4.1.1.2 Arduino Mega ADK

In case of an Arduino board, when the watchdog timer timeouts, the sketch is restarted (new call to *setup*()). Furthermore, a watchdog timeout signal is sent and this signal can be captured by the sketch. In the Arduino sketch, this is implemented by the JeeLib library (15). JeeLib is a library written for experimenting with JeeLabs products, however, some parts of the library are written for Arduino boards and can be used with them. Specifically, the Ports class (16) is the one used in this implementation to put the Arduino Mega ADK into sleep mode. The sketch execution stops for the specified sleep time and afterwards continues from the call to the sleep function. The implementation code can be seen in GitHub [1].

4.1.1.3 WiFly module

The RN-XV WiFly module can be put to sleep in two ways - sleep timer or sleep command. With the sleep timer, the shield will enter sleep mode after a specified time period has passed since all TCP active connections have closed. With the

[1] https://github.com/huberflores/ArduinoXMPP/tree/dev

sleep command the module will enter sleep mode immediately, unless an active TCP connection exists. (17).

This means that in order to put the WiFly module to sleep, all active connections must be closed. For the XMPP session, this means closing the active stream and the underlying TCP connection. Once this has been done, the module can successfully enter sleep mode.

The module can be waken up by either sending characters of the UART or by using the wake timer. In our implementation the activity on the UART wakes the module up when the sketch execution continues after the Mega ADK wakes up from sleep mode and establishes a new TCP connection in the start of *loop*() method call. This effectively means going through the first 5 stages of XMPP session lifecycle on every wake up. Since XMPP session initialization is quite verbose and the communication over Wi-Fi is unstable, the possibility of receiving scrambled or incomplete messages creates a problem.

4.1.2 Communication

Because all TCP connections and therefore the XMPP session have to be closed after every data transmit, XMPP session lifecycle steps 1 - 5 shown in Figure 3.1 are executed multiple times. When testing the XMPP implementation with Arduino Mega ADK and WiFly module sleep modes enabled, a troubling fact was discovered - the XMPP session negotiation fails at least once for every 30 minute test. The reason for these connection failures are scrambled authentication or stream opening stanzas.

From the tests, it could be seen that the average XMPP session start up time was 15 seconds. Data needs to be transmitted every 10 seconds, which means that the module can never be put to sleep as it will not be able to go through the sleep and wake up cycle during the available time period. Moreover, data transmission took on average 1 second, meaning that 15 seconds spent on wake up would result in a second of actual work, which is not efficient.

During testing the sleep and wake up cycles with XMPP, it was found that XMPP session negotiation will hang approximately once in 15 minutes. As the maximum sleep time in our implementation is 65 seconds, this means that there are at least 13 separate session negotiations. Of course, this is the problem of the XMPP library in use and its lack of error handling. Problems with the library could be addressed with a better implementation, however, there was another factor discovered during the tests.

As a result, XMPP as a means of communication was not viable when trying to minimize power consumption. As an alternative, web sockets, raw sockets and HTTP was considered. Web sockets were left out because a web socket connection is opened with an HTTP request. Thus, using that one request to actually send the data is more efficient. Therefore, HTTP was preferred to web sockets. Raw socket implementation in Arduino would add needless complexity to the sketch and was therefore not implemented. Here is a sample of used HTTP request:

```
POST /data/ HTTP/1.1
Host: ec2-54-224-196-78.compute-1.amazonaws.com
Connection: close
Content-Type: application/json
Content-Length: 105

{"location":1, "data":[{"type":1, "value":  7.87}, {"type":2, \
"value":512.00},{"type":3, "value":166.00}]}
```

The main advantages of HTTP are connection initialization speed and simplicity. The average time to wake up from sleep mode, send an HTTP request and receive response from the server, was measured to be around 5 seconds. In the previous scenario of 10 second transmission interval, it would mean 5 seconds could be spent in sleep mode, 5 seconds to wake up and transmit the data. This implementation would be more energy efficient, which can be seen in Figure 4.1. The data was gathered while running the prototype for 30 minutes with a data sending interval of 30 seconds. The sleep times were adjusted for both configurations based on their wake up times. As seen in the figure, the HTTP configuration consumes approximately 20% less power than the one using XMPP.

4.2 Server-side Client

With the move from XMPP to HTTP, the server-side client's implementation changed. Instead of using XMPP Smack library, a web server was needed. Jetty

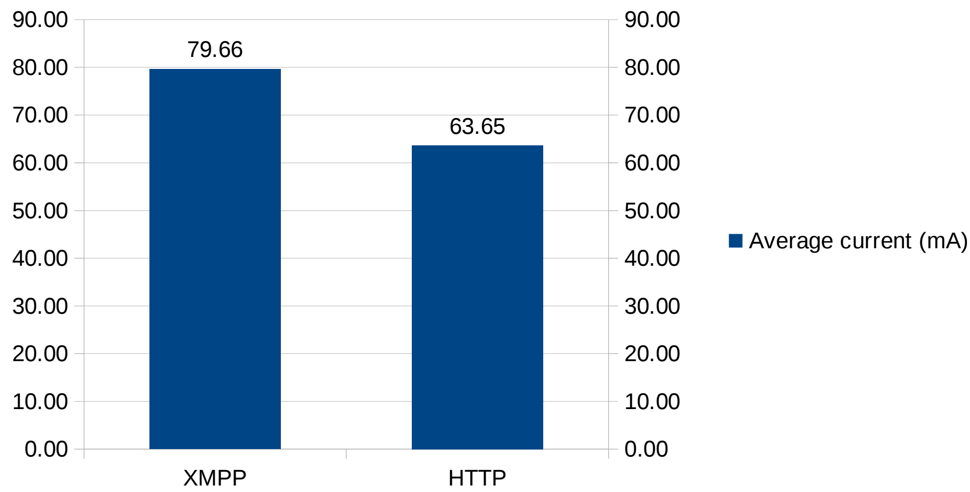

Figure 4.1: Power Consumption at 30 Second Interval

was selected because of its simplicity and possibility to embed it into the application. A web server embedded in an application is useful when prototyping, because it saves time on configuration and deployment.

Furthermore, to take full advantage of the newly developed sleep mode functionality, the client was further developed to predict sensor values for some time. Two modules were added the server for this - simple linear regression and fuzzy control engine. In addition, the data model was modified to suit the new modules. The final web server consists of 4 modules and a figure of the architecture can be seen in Figure 4.2

1. HTTP Request handler
2. Prediction module (Linear regression model)
3. Fuzzy control engine
4. Data storage

The new data model can be seen in Figure 4.3. *Data* table now has *measured* field, which indicates if the value has been measured or predicted. *SensorTypes*

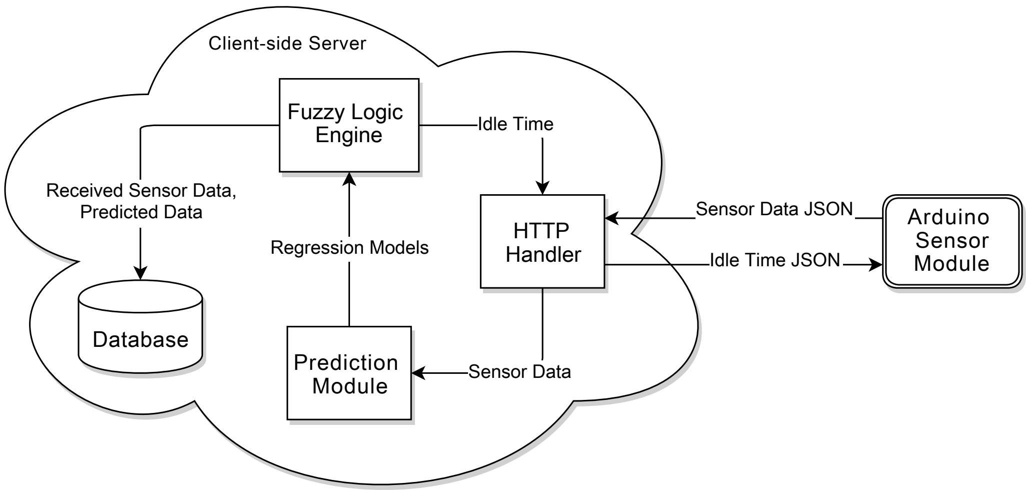

Figure 4.2: Final Solution Architecture

table has two new fields - *regression_error* and *measure_error*. *regression_error* is the acceptable regression model error for future predictions. *measure_error* indicates the acceptable variance for measured values.

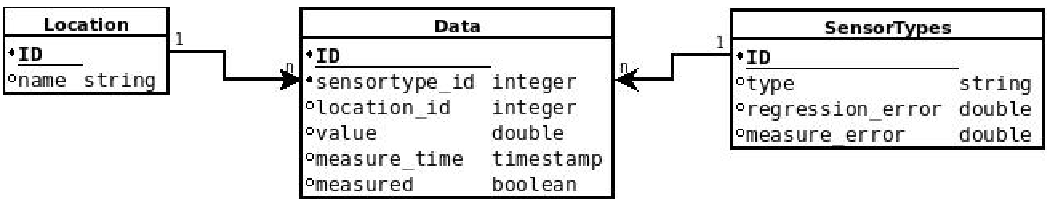

Figure 4.3: Updated Server-side Data Model

4.2.1 Prediction Module

A simple linear regression model was selected to predict future data, which uses the least squares method to calculate the future values. A linear regression model is sufficient to predict sensor data (18, p. 1066). The model has a single explanatory variable - Unix time. This variable is used to predict the future values of sensor readings based on previous measurements.

The model sample is taken from previous measurements during the last 4 minutes. The 4 minute interval is selected to balance out errors caused by extreme values. This is necessary because a single value with big enough deviation can cause the regression model to become inaccurate.

To measure the accuracy of the model, an confidence level of 90% was introduced. If the sample for last 4 minutes provides an accurate enough regression model, then the predictions can be used. Otherwise, fresh data should be queried and added to the model until the error threshold is satisfied. The equation to calculate regression model's confidence level α (18, p. 1066):

$$\alpha = 2\Phi\left(\frac{\epsilon}{RMSE}\right) - 1.$$

Here Φ is the CDF (Cumulative Distribution Function) of residuals, ϵ is the allowed error (in the database model named as $regression\_error$ and $RMSE$ is the root mean square error (also known as standard error of estimate) of the regression model. The original equation used standard deviation of errors instead of RMSE, but RMSE is used here because of computational ease. RMSE is the square root of the mean squared deviation (19) which is assumed to show the same thing as standard deviation (deviations from the mean). Furthermore, as the actual value received is the ratio of ϵ to the calculated RMSE value, the ϵ value is selected with respect to RMSE.

Regression models are created separately for each sensor. Each model has a sample of previous measurements and the allowed error ϵ defined in the database. The confidence level α is calculated for each model and has to be over 90% which is the selected threshold. If all regression models satisfy the confidence level, then the fuzzy control system is initialized with data from previous measurements and calculated confidence levels.

4.2.2 Fuzzy Logic Engine

The next step in predicting future sensor values is to calculate the time interval for the next measurement request. To calculate the time, a fuzzy control system was introduced to provide flexible decisions based on multiple input variables. The input variables are:

1. *temp*
2. *temp predictability*
3. *light*
4. *light predictability*
5. *hall*
6. *hall predictability*

Here each sensor has a pair of input variables - measured sensor value and sensor value predictability (regression model confidence level α). All these variables are mapped into fuzzy sets, a procedure call fuzzification. The predictability variables are mapped into sets which all have the same membership functions shown in figure Figure 4.4.

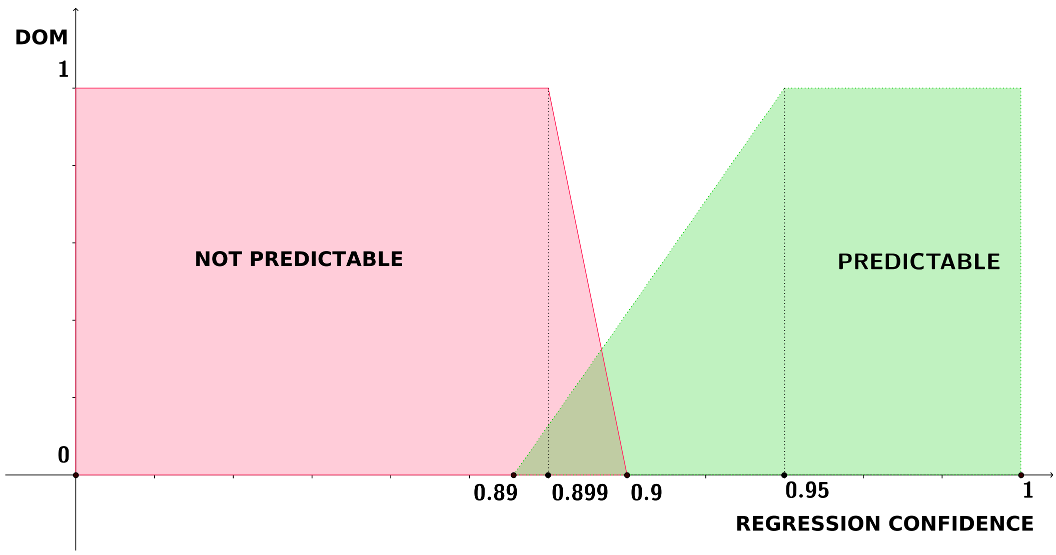

Figure 4.4: Regression Confidence Fuzzy Sets

The fuzzy control system is initialized with a previous measurement for each sensor. This effectively sets the bounds of each sensor's fuzzy sets as seen in Figure 4.5. The $PREDICTABLE$ fuzzy set's DOM (degree of membership) reaches maximum at a confidence level of 0.95 or 95% because for our prototype anything above that level is highly predictable.

The measured sensor values are fuzzified into sets, which each have different membership functions. The membership functions are calculated on the last measurement received as seen in Figure 4.5. Here X in set labels notes the sensor type currently used as the sets are the same relative to each sensor's previous measurement. The center point O is the previously measured value. $||X_2X_3||$ is the predefined sensor measurement error (*sensor_error* field in the data model). This means that if the new measurement is between the points X_2 and X_3, there is no change in the measurement for the fuzzy system. The lengths $||X_0X_2||$, $||X_1O||$, $||OX_4||$ and $||X_3X_5||$ are defined by the same variable *slope_width*. Currently, slope width is the same as the selected *regression_error* values which are 5 times the value of *measure_error*. The selected error values can be seen in Table 4.1.

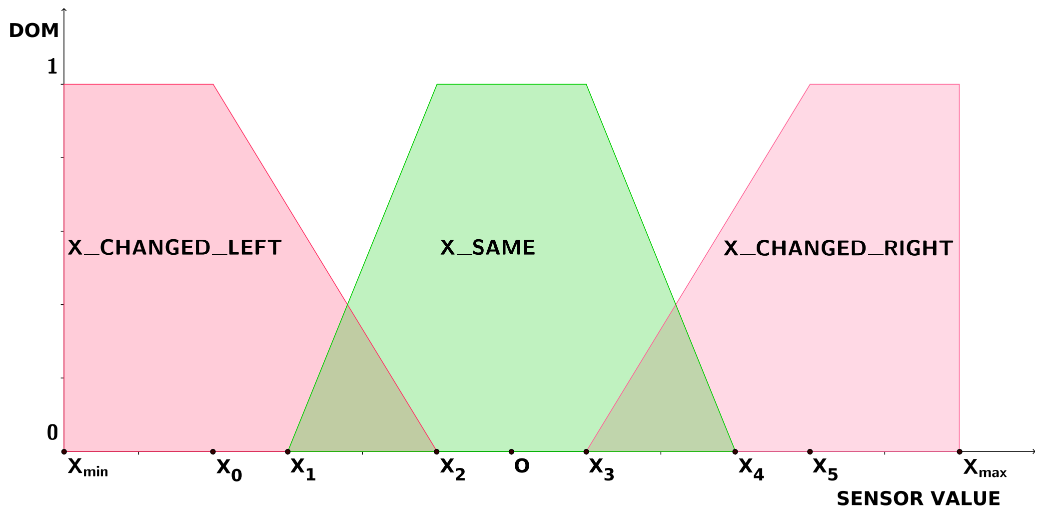

Figure 4.5: Sensor Value Fuzzy Sets

The output variable is defuzzified from the idle time sets which have membership functions described in Figure 4.6. The sets *request* and *predict* are symmetrical to enable easy defuzzification. The maximum idle time is 65 seconds, which is the maximum length of one sleep cycle for the sensor module.

	Temp	Hall	LDR
$measure\_error$	0.1	0.5	10
$regression\_error$	0.5	2.5	50

Table 4.1: Selected Error Values

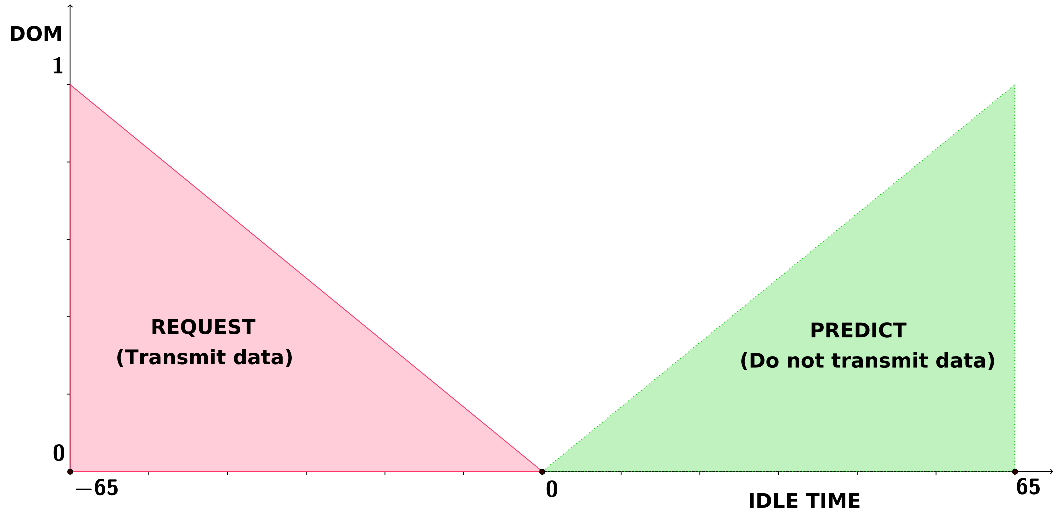

Figure 4.6: Idle Time Fuzzy Sets

When all input variables have been fuzzified and degrees of membership calculated, then the next step is to fire all predefined rules. For each sensor, there is a collection of 3 rules, making a total of 9 rules (here $x \in \{hall, temp, light\}$):

1. IF x IS $x\_changed\_left$ OR $x\_changed\_right$ AND $x\_predictability$ IS $not\_predictable$ OR $predictable$ THEN $request$

2. IF x IS x_same AND $x\_predictability$ IS $not\_predictable$ THEN $request$

3. IF x IS x_same AND $x\_predictability$ IS $predictable$ THEN $predict$

The previously calculated degrees of membership are fed into these rules. For AND relationship, the minimum value is selected, for OR relationship, the maximum value is selected. For each rule, the output variable receives a DOM equal to the DOM of the premise. When all rules have been evaluated, the results are defuzzified into a single crisp value, which is the idle time decision.

During defuzzification the DOMs of result sets *request* and *predict* for each rule are combined into a single DOM for the specified set. For this, *request* output has an *OR* relationship (maximum value) and the *predict* set has an *AND* relationship (minimum value). These relationships mean that the most inaccurate sensors would be represented in the final result. For example if *hall* has a DOM of 1.0 to *predict*, but *temp* has a DOM of 0.5 to *predict*, then the outcome would be a DOM of 0.5, because we need the output for least accurate sensors. For *request* this logic is reversed as the maximum DOM to *request* represents the output for the least accurate sensor.

The next step is to get the crisp result value. For this process, the idle time values for *predict* and *request* set membership functions at the specified DOM are calculated. x is found the equation $f(x) = DOM$, where x is the idle time and f is the membership function. *request* and *predict* membership functions are symmetric with the axis being in the point $x = 0$ as seen in Figure 4.6. The final defuzzified idle time is equal to $max\{x_{predict} - x_{request}, 10\}$. Meaning that if the DOM to *request* was higher that to predict, then the default idle time of 10 seconds is returned. Otherwise the value at the center point of the two results is returned.

When the crisp result is received from the fuzzy control system and idle time is longer than 10 seconds, future data is predicted for the idle period. This means that using the regression models, future values with 10 second intervals are inserted into the database for each sensor.

Finally, the idle time with a minimum value of 10 seconds is written as a JSON format string to the response body of the HTTP request. A sample response would look like:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: 12
Connection: close
Server: Jetty(7.2.0.v20101020)

{"idle": 34}
```

4.3 Results

4.3.1 PeakTech 1890 Power Supply

Figure 4.7: PeakTech 1890

A PeakTech 1890 programmable power supply was used when testing the power consumption of the Arduino sensor module. The power supply enables to set variable output voltage and current values and has RS-232 and RS-485 interfaces for connecting with external devices. The RS-232 interface was used to collect drawn current values with 1 second interval. The power supply can be seen in Figure 4.7

4.3.2 Overview of Tests

All average current tests were carried out using the PeakTech 1890 power supply. The power supply collected data with 1 second interval for 1800 seconds (30 minutes).

When testing with the server hosted in an Amazon EC2 micro instance, it was discovered that the previously used 3G wireless connection was not stable enough to provide consistent connections. Previous tests were made in a local area network where 3G was sufficient. To fix this, a switch to local Wi-Fi connection was made, the connection speeds were measured at 17.68 Mb/s download and 23.44 Mb/s upload. The latency of HTTP response for the we server was measured with Google Chrome web browser's developer tools console and produced an average of 120ms.

4.3.3 Test Results

Firstly, the average current drawn in a second was measured for the original XMPP implementation, the final HTTP implementation with a hard coded 10 second idle time (same as the original XMPP version) and finally, the HTTP implementation with variable idle times enabled. The results can be seen in Figure 4.8 and Table 4.2. As seen from the table, there was a 23% improvement for the HTTP version with the same fixed idle time. With variable idle time, the improvement was 42% with the average idle time measured at 37.15 seconds.

	Average current (mA)	Percentage	Improvement
XMPP 10s interval	111.74	100%	
HTTP 10s interval	86.14	77%	23%
HTTP final	64.39	58%	42%

Table 4.2: Overview of Current Measurement Results

Secondly, the module's lifetime on a 9V battery was tested. The result time was the time difference between the first and the last request received by the server-side client after powering the sensor module from a 9V battery. Two separate tests were carried out - one later in the day and one earlier to show behaviour

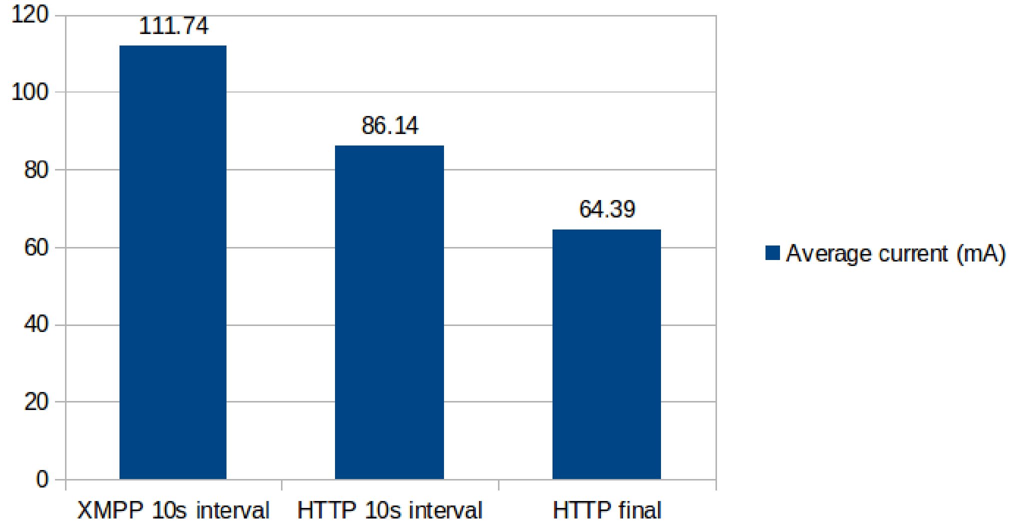

Figure 4.8: Current Measurements Comparison

in different conditions. The difference mainly involved the LDR sensor which is quite sensitive and slight fluctuations in lighting affect the measurements.

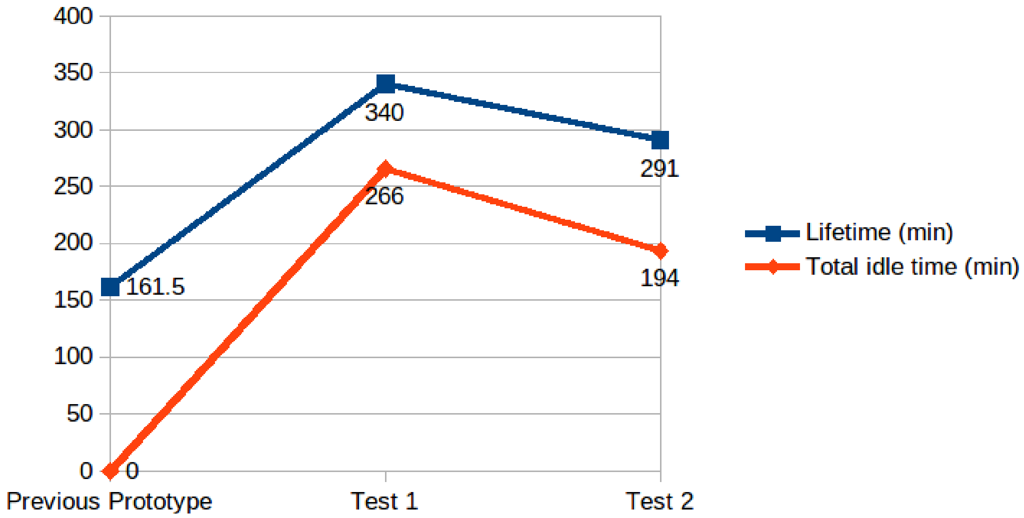

Figure 4.9: Battery Lifetime and Idle Time Comparison

	Lifetime (min)	Improvement (%)	Total Idle Time (min)
Previous Prototype	161.5	-	0
Test 1	340	111	266
Test 2	291	80	194

Table 4.3: 9V Battery Lifetime Test Results

From Table 4.3 it can be seen that the two test results differ a bit. This is because Test 1 was done in the evening and Test 2 was done during the day, which affects the LDR sensor's readings. During the day the sensor readings change a lot due to the variable lighting conditions. However, both tests confirm that the fuzzy control engine and prediction system work well together and that the overall battery lifetime has improved. The decrease in power consumption is dependent on the actual time spent in idle mode, because the more the module sleeps, the less power it consumes in total.

As can be seen in Figure 4.9, change in total idle time and battery lifetime have almost identical lines, which indicates that they are closely correlated. This, in fact, is quite logical because the main decrease in power consumption comes from utilizing idle mode as much as possible.

The server-side client was not separately load tested, because the average request handling time was measured to be 53ms, which is quick enough to handle sensor module loads. Furthermore, since Jetty web server is a Java application server, which has proven itself in real commercial implementations, the solution was trusted to handle larger number of requests.

All in all, the outcome is positive with the daytime test giving an 80% and the evening test giving 111% improvement. The different results are caused by the selection of acceptable error values, which can be fine tuned based on the actual measurement characteristics needed.

5

Conclusions

Context aware applications have become more common in recent years and therefore mobile applications need to take advantage of the possibilities they offer. To provide this contextual information, a prototype solution was developed, which registers sensor data from an Arduino-based module and saves it in a data server.

The prototype solution was further developed in this thesis to increase battery performance and therefore the usability of the prototype. For this, a variable sensor reading interval solution was developed, which consists of a fuzzy control engine and a simple linear regression model. Moreover, the sensor module was improved to enable sleep mode during periods of inactivity to fully utilize the variable sensor reading times.

A switch from XMPP to HTTP was made on the communication protocol part. This switch was necessary to enable sleep mode in the sensor modules, since XMPP session negotiation is a verbose process and therefore takes longer than making a simple HTTP request. The move to HTTP decreased connection initialization times, which enabled the module to utilize sleep mode for longer periods.

Several tests were carried out to measure the actual benefits of the proposed improvements and the results were positive. Two tests, with improvements of 80% and 111% over the previous prototype's battery lifetime, were carried out. These tests show that the proposed solution of enabling sleep mode in the sensor module for idle periods decreases power consumption and enables the module to perform its tasks for a longer period of time.

6

Related Work

A solution for energy-efficient data collection for clustering-based wireless sensor networks has been suggested (18). This solution has wireless sensor clusters with a head node, which actively decides if new sensor data should be requested from the nodes or if they can be predicted. The communication with the central data collection point goes through cluster heads only, which reduces transfer overhead. This is similar to the current thesis in a sense that data is predicted or requested and then forwarded, with the exception that this prototype does not have clusters and extra complexity that comes with clustering. The general implementation idea has been taken from this solution and prediction algorithm details, as in both cases simple linear regression is used to predict sensor data.

The previous prototype (2) is the main basis of this work as it was the starting point. The previous implementation used XMPP as the communication protocol and a fixed interval data collection. XMPP has several advantages over simple HTTP requests, mainly because the protocol supports message queues and security. However, actually useful security options (SSL/TLS) are unavailable due to their overhead and implementation complexity for Arduino boards. On the other hand, XMPP protocol has communication overhead which reduces its usability when fast connection establishment and data transmission is required.

7

Future Research Directions

The changes made in this thesis are done with the notion of improving on a proof of concept prototype. This means that the changes are done to prove a concept as well. Hence, several aspects of the changes implemented can be improved upon.

Firstly, starting with the Arduino sensor module, the hardware side of the module can greatly affect the power consumption of the device. Currently, Arduino Mega ADK is used, which is one of the most power-hungry boards available. Currently, the board on average draws $55mA$ in sleep mode, which is very close to the average current drawn as seen in Figure 4.8. When this number is lowered, the average current drawn will decrease and thus improve battery lifetime.

Secondly, the server-side client's fuzzy control engine and prediction models can be improved. At the moment, the fuzzy control engine checks if the measured value has changed from a previous measurement by a degree x. If it has, a new measurement is requested. However, this configuration does not cover the case when the measurement values changes steadily by a degree which is larger than the acceptable $measure\_error$. In this case, the changes in value might be large, but if they are steady, then the future values could still be predictable.

Thirdly, the prediction model can be improved upon to better handle extreme values, because currently, a large enough deviation can cause the model to become inaccurate. The selected $measure\_error$ and $regressin\_error$ values can be adjusted to better suit the selected sensors and their data ranges. At the moment, the values are based on a trial and error tuning of the models to suit the needs of the prototype level device. However, in an actual implementation environment, these values should be based of the type of information needed from the sensors. For example, a small change in the light sensor measurements might indicate that an object (a bird) has flown past the sensor, but if the goal is to measure the cloudiness of the sky or the time of day, the acceptable deviations in the sensor values can be larger to ignore small fluctuations.

Finally, as XMPP was replaced by HTTP for the sleep mode to be usable in the Arduino sensor module, a large component of the previous prototype was changed. This switch gave faster connection establishment and data transmission times. On the other hand, several advantages of XMPP have gone missing, most notably the possibility to queue offline messages and status updates for sensor modules. Therefore, a more complex communication solution can be developed to provide these possibilities while still providing fast connection times.

Mobiilsetele kasutajatele sensoritelt kogutud keskonnapõhiste andmete energiateadlik edastamine

Bakalaureusetöö (6 EAP)
Lauris Kruusamäe

Tänapäeval levib järjest rohkem rakendusi, mis tajuvad ümbritsevat keskkonda ning pakuvad sellele põhinevalt kasutajale lisavõimalusi. Selliste võimaluste pakkumiseks on arendatud prototüüp, mis kogub keskkonna kohta andmeid kasutades Arduino platvormil põhinevad sensorite moodulit ning keskset andmet kogumise serverit.

Käesolevas töös arendati antud prototüüpi edasi, et tõsta aku vastupidavust ning seeläbi parandada lahenduse kasutatavust. Selleks loodi varieeruva sensoriandmete saatmise intervalliga lahendus, mis koosneb hägusloogikat kasutavast kontrollsüsteemist ning lihtsa lineaarse regressiooni mudelist. Lisaks loodi lahendus, mis lubab sensorite moodulil vabadel hetkedel minna puhkerežiimi.

Töö käigus asendati seni kasutuselolev XMPP protokoll HTTP protokolli vastu, et parandada ühenduse loomise ajakulu ning lubada sensorite moodulil kauem puhkerežiimis olla.

Parandatud lahenduse tulemusi mõõdeti mitme testi käigus. Kaks põhilist testi, mille käigus sensorite moodul sai voolu 9-voldiselt patareilt, andsid vastavalt tulemusteks 80 ja 110 % pikema eluea. Sellest tulenevalt saab eeldada, et pakutud puhkerežiimi ja muutuva intervalliga andmete kogumist kasutav lahendus parandab aku vastupidavust ning seega ka prototüübi kasulikkust.

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Lauris Kruusamäe (date of birth: 30/12/1989), herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Energy-aware Sensor Data Collection for Mobile Users
supervised by Huber Flores and Satish Narayana Srirama,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 13.05.2013

Bibliography

[1] H. Flores, S. N. Srirama, Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning, Tech. rep., Institute of Computer Science, Mobile Cloud Lab, University of Tartu (2013). v, 12

[2] Kaarel Hanson, Context Sensor Data on Demand for Mobile Users Supported by XMPP (2012). v, 1, 2, 14, 16, 17, 18, 35, 42

[3] Arduino - HomePage, http://www.arduino.cc/, 2013-04-16. 4

[4] Arduino - ArduinoBoardADK, http://arduino.cc/en/Main/ArduinoBoardADK, 2013-04-18. 5

[5] Arduino - ArduinoWirelessShield, http://arduino.cc/en/Main/ArduinoWirelessShield, 2013-04-18. 6

[6] RN-XV WiFly module - wire antenna - SparkFun electronics, https://www.sparkfun.com/products/10822, 2013-04-18. 7

[7] Tinkerkit introduction, http://www.tinkerkit.com/introduction/, 2013-04-19. 7

[8] Thermistor sensor, http://www.tinkerkit.com/thermistor/, 2013-04-27. 8

[9] Hall sensor, http://www.tinkerkit.com/hall/, 2013-04-27. 8

[10] Light sensor, http://www.tinkerkit.com/ldr/, 2013-04-27. 8

[11] Marcelo Godoy Simoes, Introduction to Fuzzy Control, 1610 Illinois Street Golden, Colorado 80401-1887 USA. 9, 10

[12] Lotfi A. Zadeh, Information and Control, 1965, Ch. Fuzzy Sets, p. 338–353. 10

[13] Least squares fitting – from wolfram MathWorld, http://mathworld.wolfram.com/LeastSquaresFitting.html, 2013-05-04. 13

[14] Jim Lamberson, Single and Multistage Watchdog Timers (2012). 20

[15] JeeLib: introduction, http://jeelabs.net/pub/docs/jeelib/, 2013-04-23. 20

[16] JeeLib: port class reference, http://jeelabs.net/pub/docs/jeelib/classPort.html, 2013-04-23. 20

[17] WiFly user guide - WiFly-RN-UM.pdf, http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/WiFi/WiFly-RN-UM.pdf, 2013-04-23. 21

[18] H. Jiang, S. Jin, C. Wang, Prediction or not? an energy-efficient framework for clustering-based data collection in wireless sensor networks, Parallel and Distributed Systems, IEEE Transactions on 22 (6) (2011) 1064–1071. doi:10.1109/TPDS.2010.174. 24, 25, 35

[19] Root-mean-square – from wolfram MathWorld, http://mathworld.wolfram.com/Root-Mean-Square.html, 2013-05-04. 25

Appendix A

The Arduino sensor module sketch and server-side client implementation code can be accessed from https://github.com/huberflores/ArduinoXMPP/. The *master* branch contains the original prototype (2). The current implementation is in the *dev* branch.

The code, collected data and thesis LaTeX files are also available on a CD attached to the thesis.