ΤΕΙ Κρήτης

Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Μηχανικών Πληροφορικής

Σύστημα Καταγραφής και Παρουσίασης Καιρικών Μετρήσεων

Κοντάκης Μάνος

Acknowledgements

I would like to thank Mr. Costis Aivalis, the instructor of this bachelor thesis for supporting the subject I chose to involve. I would also like to thank my girlfriend, Efi, for supporting all my effort and mainly for her help during the testing of the wind vane and the anemometer and for her understanding for the unending hours in front of my computer during the nights!

I also feel the need to thank:

- The StackOverflow community
- SurferTim, a member of the Arduino community, for his help to solve a problem I was stuck for weeks [44]
- Darek Banas for his awesome Android Development tutorials

Abstract

The purpose of this project is to measure and gather the weather data and visualize them. The data gathering process is implemented upon Arduino Microcontroller that is programmed using C++ and a set of libraries. Arduino is an open source hardware prototyping platform, which allows an easy implementation of sensors and interactive elements. Data gathered are stored on a MySQL database via the Ethernet Shield which connects the Arduino to the internet. The manipulation of the data is done through PHP scripts from the Android Application and the Web Page in order to retrieve them and visualize them. The database and the web server are hosted on a Raspberry PI, a single board, low cost computer. Finally, the project includes a web camera mounted on a pan – tilt bracket connected on the Raspberry PI, that captures images every one minute and once per day it encodes the daily Time Lapse video, it uploads it to YouTube and it automatically embeds it on the Web Page.

Σύνοψη

Ο σκοπός αυτής της εργασίας είναι η συλλογή και η οπτικοποίηση των καιρικών δεδομένων. Η διαδικασία της συλλογής έχεις υλοποιηθεί πάνω στον μικροελεγκτή Arduino ο οποίος έχει προγραμματιστεί με C++ και ένα σύνολο από βιβλιοθήκες. Το Arduino είναι μια πλατφόρμα πρωτοτύπων ανοικτού κώδικα[1] το οποίο επιτρέπει την εύκολη υλοποίηση αισθητήρων και διαδραστικών στοιχείων. Τα δεδομένα που συλλέγονται αποθηκεύονται σε μια MySQL βάση δεδομένων δια μέσου του Ethernet Shield που συνδέει το Arduino στο διαδίκτυο. Ο χειρισμός της βάσης δεδομένων επιτυγχάνεται μέσω PHP και JavaScript σεναρίων (scripts) από την Android εφαρμογή και την ιστοσελίδα ώστε να ανακτήσουν και να οπτικοποιήσουν τα δεδομένα. Η βάση δεδομένων και ο εξυπηρετητής φιλοξενούνται σ' ένα Raspberry PI, ένα μικρού κόστους υπολογιστή σε μέγεθος πιστωτικής κάρτας. Τέλος η εργασία περιλαμβάνει μία διαδικτυακή κάμερα τοποθετημένη πάνω σε μια περιστρεφόμενη βάση και συνδεδεμένη στο Raspberry PI, το οποίο τραβάει φωτογραφίες κάθε ένα λεπτό και στο τέλος κάθε ημέρας δημιουργεί ένα βίντεο με τις φωτογραφίες αυτές. Έπειτα το μεταφορτώνει στο YouTube και το ενσωματώνει αυτόματα στην ιστοσελίδα.

[1] Όταν ο όρος 'Ανοικτός κώδικας' αναφέρετε σε υλικό σημαίνει ότι ο σχηματικό του και άλλες πληροφορίες σχετικά με το υλικό είναι δημόσια διαθέσιμα σε περίπτωση που κάποιος θέλει να το συναρμολογήσει μόνος του.

Contents

Nomenclature

°C	Degrees Celsius
ADC	Analog to Digital Converter
API	Application Programming Interface
CMS	Content Management System
CWR	Current Weather Report
DDNS	Dynamic Domain Name System
DHCP	Dynamic Host Configuration Protocol
EMC	Electromagnetic Compatibility
GNU	GNU's Not Unix
GPL	General Public License
GPU	Graphics Processing Unit
hPA	Hecto Pascal
I^2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
ISP	Internet Service Provider
JS	JavaScript
MC / MCU	Microcontroller
MCUSR	Microcontroller Status Register
OS	Operating System
PI	Raspberry PI
RH	Relative Humidity
RPI	Raspberry PI
SoC	System on Chip
TCO	Total Cost of Ownership
Tx / Rx	Transmitter Mode / Receiver Mode
UP	Uncompensated Pressure
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UT	Uncompensated Temperature
WDT	Watchdog Timer
WDTCSR	Watchdog Timer Control Register

1. Introduction

The observation of the weather stands for millenniums. People from ancient years were trying to predict the weather with methods usually relied on current weather conditions or observed patterns of events[2]. Even nowadays, that weather forecasting relies on computer - based models that take many atmospheric factors into account, barometric pressure, current weather conditions and weather statistics are three of them. These three factors are measured from weather stations.

A weather station is a facility, either on land or sea, with instruments and equipment for measuring atmospheric conditions to provide information for weather forecasts and to study the weather and climate. The measurements taken include **temperature, barometric pressure, humidity, wind speed, wind direction, and precipitation amounts**. Wind measurements are taken with as few as other obstructions as possible, while temperature and humidity measurements are kept free from direct solar radiation, or insolation. Automated measurements are taken at least once an hour. Weather conditions out at sea are taken by ships and buoys, which measure slightly different meteorological quantities such as sea surface temperature, wave height, and wave period.

1.1. Motivation

When I enrolled to Industrial Automation course, I was introduced to the Arduino platform. It instantly intrigued me. Within three months after my first touch with it, I bought my first Arduino and I was thinking of a project to come to grips with. After some thinking I ended up on the Weather Station leaving behind some very interested ideas, such as Quad copter and Home Alarm mainly due to their high cost.

My interested on Weather Data is inherited from my favorite sport – Sailing. For sailors observing the weather is a major part of their daily routine so this was the icing on the cake to begin!

I first thought of building the Weather Station for educational purposes and as a hobby. Later on I realized that the project could also serve as my final year’s project so I declared it.

1.2. Purposes and targets

The target of this thesis is to build a fully functional weather station that gathers weather data and visualize them. The visualization is being done on two platforms, Android and Web.

1.3. Project Structure

The project is split in two major categories that include subcategories.

[2] For example, it might be observed that if the sunset was particularly red, the following day often brought fair weather

a. Hardware Components

This part is focused on the hardware that was for the development of the weather station.

i) Raspberry PI
ii) Arduino
iii) Sensors
iv) Prototype Architecture

b. Software Analysis

In this section the software used is in – depth analyzed.

i) The Arduino part in which data are gathered.
ii) The server part that hosts the server-side PHP code that is used to store and retrieve data from the database.
iii) The Android Application part that visualizes the data.
iv) The Time Lapse Video Making
v) The Web Site part that also visualizes the data.

2. Implementation Methodology

The implementation of this thesis is based on the single-board microcontroller Arduino and on the Raspberry PI computer. A variety of sensors are connected on Arduino that gather the weather data such as BMP085, DHT22 and Sparkfun's Weather Meters [19]. The data gathered are stored on a MySQL database, hosted on the PI, that is administrated through PhpMyAdmin. The Android application is developed upon Eclipse using the Android Development Tools (ADT) plugin. The website that is also hosted on the PI using Apache HTTP Server, is built using JavaScript and PHP and the CMS used is WordPress.

3. Project Plan

The project has been implemented using the latest technologies available for its needs. The next chapter, state of the art, provides a more in depth analysis about the background of the project.

3.1. State Of The Art

- **Open – Source Hardware**

  Open - Source has become a popular expression, but mostly with regard to software. The principles and definitions of Open-Source Hardware (OSHW) are closely related to those of Open-Source Software (OSS) from the Open Source Initiative [1, 2]. Interested people can study, modify, distribute, make and sell designs based on those of OSHW products [3]. Through open development, people get the possibility to learn and understand how OSHW works, so that they are able to control and modify their technology. Machines, devices, or other physical things produced under the OSHW license must comply with the definitions of the Open Source Hardware Association [1].

- **Arduino Platform**

  Arduino is a prototyping platform containing a microcontroller board (MC) as core element, a programming language and an integrated development environment (IDE) [4]. OSHW MCs from Arduino are based on ATmega8, ATmega168, ATmega2560 and latest boards on ATmega32u4, Sitara AM335x. They can be used to develop interactive prototypes, which are using input from sensors to control output devices connected to the same board. The capabilities of the MCs allow several methods for in- and outputs of signals.

  The simplified IDE with a wire-based programming language also allows beginners to realize complex projects in short time. Documentation of the IDE and the language reference can be found on the Arduino homepage [5]. All code samples are released into the public domain. Additionally, the language can be extended with C++ libraries to enable more functionality. All official software tools from Arduino are published under the OSS license and are platform-independent.
  The well documented Arduino hard- and software makes it even possible to rebuild MCs by oneself. However, pre-assembled Arduino boards are relatively inexpensive compared to other commercial MC platforms available on the market

- **Raspberry PI**

  The Raspberry PI is a credit – card – sized single board computer developed mainly for educational purposes. As core element it has a Broadcom BCM2835 system on a chip (SoC) [9] which includes an ARM11 family 700MHz processor, a VideoCore IV GPU and 512MB RAM.
  The Raspberry PI foundation provides a lightweight version of Debian (Raspbian) and Arch Linux ARM distribution as official operating systems. Tools are available for Python as the main programming language, C, Java and Perl.
  For the needs of the project, the RPI is used as server, hosting the script files that are used to store and retrieve weather data in the database as well as hosting the web page and the database. Later it is explained why the RPI is the perfect small server (Advantages of Raspberry PI being a Web Server).

- **Automated Weather Stations**

  Automated weather stations (AWS) are meteorological stations, which perform climatic observations and data transmission automatically [6]. Main advantages compared to human weather observations are, that measurements, read out by a central data-acquisition unit, are more reliable and can be performed much more frequently. Sensors of an AWS are operated by a microprocessor. It allows exact sampling of sensor data and processing for averaging or filtering of the samples. The result will be a series of observations which are representative over a limited area.

- **Android Platform**

  Android is a mobile operating system based on the Linux Kernel that is currently developed by Google. It powers hundreds of millions of mobile

devices around the world. It is the largest installed base of any mobile operating system platform.
Android’s source code is released by Google under open source licenses (Apache License 2.0 and GNU General Public License 2.0 [7, 8]).
Its open nature has encouraged a large community of developers and enthusiasts to use the open – source code as a foundation for community-driven projects, which adds new features for advanced users.

- **Google Chart API**
  The Google Chart API [10] is a tool that lets people easily create a chart from some data and embed it in a web page. Google creates a PNG image of a chart from data and formatting parameters in an HTTP request. Many types of charts are supported, and by making the request into an image tag, people can simply include the chart in a web page.

- **Android GraphView.**
  GraphView is a library for Android to programmatically create flexible and nice-looking diagrams. It is easy to understand, to integrate and to customize it. At the moment there are two different types:

  - Line Charts
  - Bar Charts

4. Basic Part

4.1. Hardware Components

The Microcontroller controls the data storage process and sampling frequency. Sampling rates of sensors are depending on the time constant value that we set. Sensors and Microcontroller functionality are explained in depth in the upcoming section 0.

A server is a running instance of a Software capable of accepting requests from the client and giving responses accordingly. In the case of the project the Single – Board Computer Raspberry PI operates as the server. It is a critical part for the project being functional, since it is responsible for storing the data to the database it hosts, for distributing the data where requested and hosting the website (web server). In the next section (4.1.1) more about the PI can be found.

4.1.1. Raspberry PI

The Raspberry PI holds a critical place in the project. It has to perform several tasks that without them the system is useless. These tasks are the following.

- Web Server
- Website Hosting
- WordPress Hosting
- Database Hosting
- Dynamic DNS Update Client
- Time Lapse Photography

Web Server

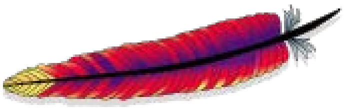


Figure 1 - Apache Logo

The primary function of a web server is to store, process and deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to text content.

A web browser or web crawler, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so. The resource is typically a real file on the server's secondary storage, but this is not necessarily the case and depends on how the web server is implemented.

While the primary function is to serve content, a full implementation of HTTP also includes ways of receiving content from clients. This feature is used for submitting web forms, including uploading of files or receiving data to be stored on a database.

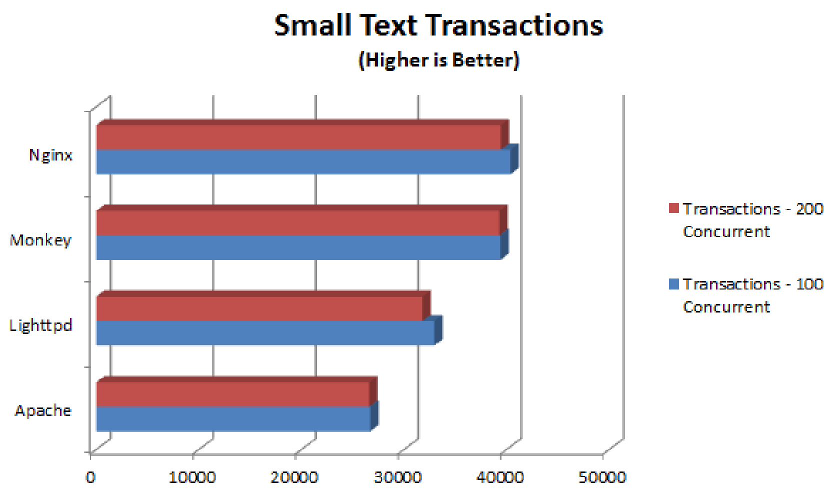
Many generic web servers also support server-side scripting using Active Server Pages (ASP), PHP, or other scripting languages. This means that the behavior of the web server can be scripted in separate files, while the actual server software remains unchanged. Usually, this function is used to create HTML documents dynamically ("on-the-fly") as opposed to returning static documents. The former is primarily used for retrieving and/or modifying information from databases. The latter is typically much faster and more easily cached but cannot deliver dynamic content.

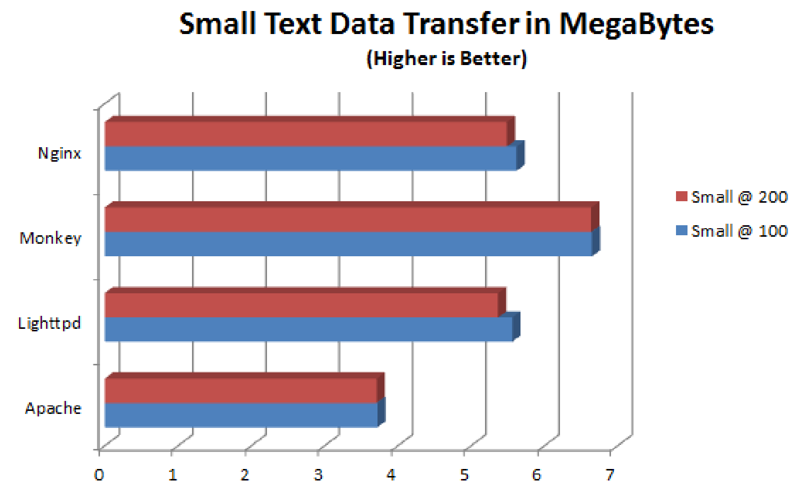
Web servers are not always used for serving the World Wide Web. They can also be found embedded in devices such as printers, routers, webcams and serving only a local network. The web server may then be used as a part of a system for monitoring and/or administering the device in question. This usually means that no additional software has to be installed on the client computer, since only a web browser is required (which now is included with most operating systems).

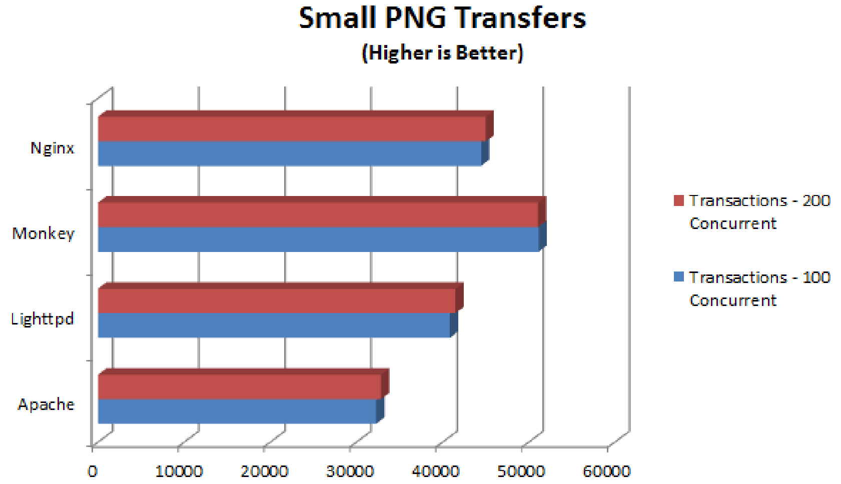
These was some general information about Web Servers. More specifically, for the project the Apache HTTP Server is used. Colloquially called Apache is notable for playing a key role in the initial growth of the World Wide Web. It is the most popular HTTP server in use since April 1996, serving more than 100 million Web sites.

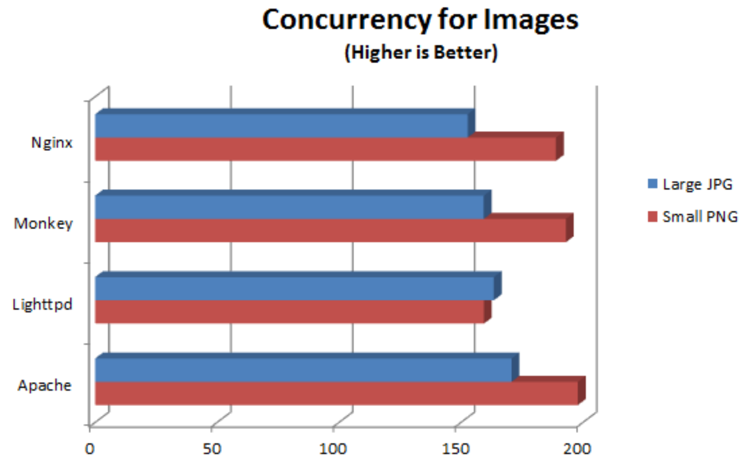
Apache was chosen among its main competitors Nginx, LightHttpd and Monkey, even if it lacks on speed as shown in the tests, because of its variety of features and the huge community.

The speed tests in the next page compare how each Web Server behave on the rPI.









Advantages of Raspberry PI being a Web Server

The Raspberry PI is ideal for being a Home Server. Obviously its ARM11 processor cannot handle a lot of traffic but the advantages are so many that overlay that.

Below there is a list of some of the advantages of the PI being a Server.

- **Power consumption** - The Pi draws about five to seven watts of electricity. This is about one tenth of what a comparable full-size box can use. Since servers are running constantly night and day, the electrical savings can really add up. I have calculated that the basic Pi kit (Pi board, case, and power supply) will pay for itself with about one year's worth of electricity savings if it's left to run 24x7x365.

- **No moving parts** - The Pi uses an SD card for storage, which is fast and has no moving parts. There are also no fans and other things to worry about. A Class 10 SD card is usually the best performing compared to lower class cards, but this will mainly only affect boot time where there is the most I/O.
- **Small form factor** - The Pi (with a case) is smaller than an adult’s palm. A comparable full-size box is a lot times bigger. This means the Pi can be integrated inside of devices, too.
- **Status lights** - There are several status lights on the Pi's motherboard. With a clear case one can see NIC activity, disk I/O, power status, etc.
- **Huge community support** - The Pi has phenomenal community support. Support can be obtained quite easily for the hardware and/or GNU/Linux software that runs on the Pi mainly in user forums, depending on the GNU/Linux distribution used.
- **Overclocking capability** - The Pi can be overclocked if there are performance problems with the application used. Especially with a heat sink installed, the risk of damaging the board is almost non-existing.
- **Software** - The Raspberry Pi primarily uses Linux kernel-based operating systems. That makes it even more attractive due to all the benefits that Linux-based OS has in the field of servers. Some of these are:
    - Stability – Linux systems are well known for their ability to run for years without failure.
    - Security - Linux is also innately more secure than Windows is, whether on the server, the desktop or in an embedded environment. That's due largely to the fact that Linux, which is based on Unix, was designed from the start to be a multiuser operating system. Only the administrator, or root user, has administrative privileges, and fewer users and applications have permission to access the kernel or each other. That keeps everything modular and protected
    - Total cost of Ownership – There is no beating Linux’s TCO, since software is generally free.
    - Freedom – With Linux, there is no commercial vendor trying to lock you into certain products or protocols. Instead, it allows the user to mix and match and choose what works best for each job.
- **Affordable** - Compared to other similar alternatives, the Pi (revision B) offers the best specs for the price. It is one of the few devices in its class that offers 512 MB of RAM.

WordPress Hosting

WordPress [34] is a free and open source blogging tool and a content management system (CMS) based on PHP and MySQL. Features include a plugin architecture and a template system. WordPress was used by more than 22.0% of the top 10 million websites as of August 2013. WordPress is the most popular blogging system in use on the Web, at more than 60 million websites.

It was first released on May 27, 2003, by its founders, Matt Mullenweg and Mike Little, as a fork of b2/cafelog. The license under which WordPress software is released is the GPLv2 (or later) from the Free Software Foundation. [Source: Wikipedia]

WordPress was chosen among its rivals because it encapsulates a variety of advantages. Some of these are the following.

- Ease of use – WordPress is very easy to use out of the box.
- It is free to use - In addition to the freedom from licensing costs and restrictions, with WordPress, you get full access to the source code. This means that any changes can be made without reliance on any one particular web developer.
- It is well supported – WordPress is well supported by its huge community. The support comes mainly from its own Support Page and StackOverflow.
- It is continually improved - WordPress is constantly being updated and maintained. WordPress has developers all over the world contributing to its continual improvement. It is already a great system and it’s getting even better every day
- Plethora of Plugins

Hosting and running WordPress is intense for the Raspberry PI due to its limited processing power. As soon as you start using WordPress its CPU usage instantly raises to 50% and more. Although, even if it is not running super-fast, it is enough for educational and experimental purposes, like this.

Dynamic DNS Update Client

The main problem when hosting a Home Web Server are Dynamic IPs. This means that the public IP of a broadband connection changes. How often it changes it depends from the ISP.

ISPS prefer Dynamic IP over Static IP [more on A.1] because addresses are much more difficult to manage. Even most "static" IP addresses are assigned dynamically through DHCP using static reservations, but with either a statically configured IP address or a DHCP static reservation, if the client connection device changes (hardware replacement, upgrade, etc), then someone needs to make a change either on that device or in the DHCP configuration. Dynamic assignment avoids this problem.

Using DHCP also simplifies the network design and allows for easier changes later. If the ISP chooses to make changes to the way they are using their IP addresses, they can do so without making manual changes on client connection devices. Examples of such changes could include any of the following (or others):

- Increasing the size of the network (moving from a /24 network to a /23)
- Decreasing the size of the network (moving from a /24 to a /25)
- Changing the network in use entirely (moving from 10.0.0.0/24 to 192.168.0.0/24)
- Changing the gateway IP address
- Changing the DNS server addresses

DHCP also allows for the ISP to provide additional configuration to some devices. For instance, if they want to provide images/configuration on a TFTP server, this can be provided to the device through DHCP.

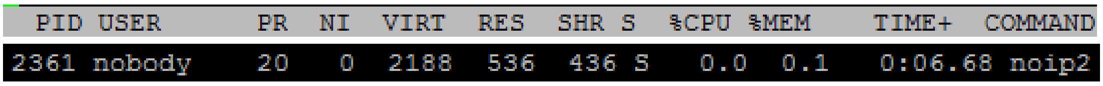
These changes on IPs present a problem if the customer wants to provide a service to other users on the Internet, such as a web service. As the IP address may change frequently, corresponding domain names must be quickly re-mapped in the DNS, to maintain accessibility using a well-known URL. That problem Dynamic DNS service providers comes to solve.

Dynamic DNS (DDNS) is a method of automatically updating a name server in the Domain Name System (DNS), often in real time, with the active DNS configuration of its configured hostnames, addresses or other information.

The term is used to describe two different concepts. At the administration levels of the Internet, "dynamic DNS updating" refers to systems that are used to update traditional DNS records without manual editing. These mechanisms are explained in RFC 2136, and use the TSIG mechanism to provide security.

Another kind of dynamic DNS permits lightweight and immediate updates to its local database, often using a web-based mechanism. It is used to resolve a well-known domain name to an IP address that may change frequently. It provides a persistent addressing method for devices that change their location or configuration.

For the needs of the project No – IP service provider is used. No – IP offers for free a client that continually checks for IP address changes in the background and automatically updates the DNS at No – IP whenever it changes. The client is a very lightweight written in C.



```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2361 nobody    20   0  2188  536  436 S  0.0  0.1   0:06.68 noip2
```

Figure 2 - DDNS client

Time Lapse Photography

Raspberry PI also handles the creation of the daily Time Lapse video.

Time-lapse photography is a technique whereby the frequency at which film frames are captured (the frame rate) is much lower than that used to view the sequence. When played at normal speed, time appears to be moving faster and thus lapsing. For example, an image of a scene may be captured once every second, then played back at 30 frames per second; the result is an apparent 30 × (24 [cinema] / 25 [PAL] / 30 [NTSC] frames per second) times speed increase. Time-lapse photography can be considered the opposite of high speed photography or slow motion.

Processes that would normally appear subtle to the human eye, e.g. the motion of the sun and stars in the sky, become very pronounced.

This procedure occupies the PI around the clock, since it must capture pictures, through the USB camera, every two minutes and every day at midnight it 'merges' the pictures into a video. More on this will be analyzed on chapter 4.2.3.

4.1.2. Arduino

The Arduino and the Ethernet Shield can be imagined as the 'brain' of the weather station. It provides all the core functionality from sensor manipulation up to data logging. The Arduino Mega and the Arduino Ethernet Shield are introduced below

4.1.2.1. Arduino Mega 2560 R3

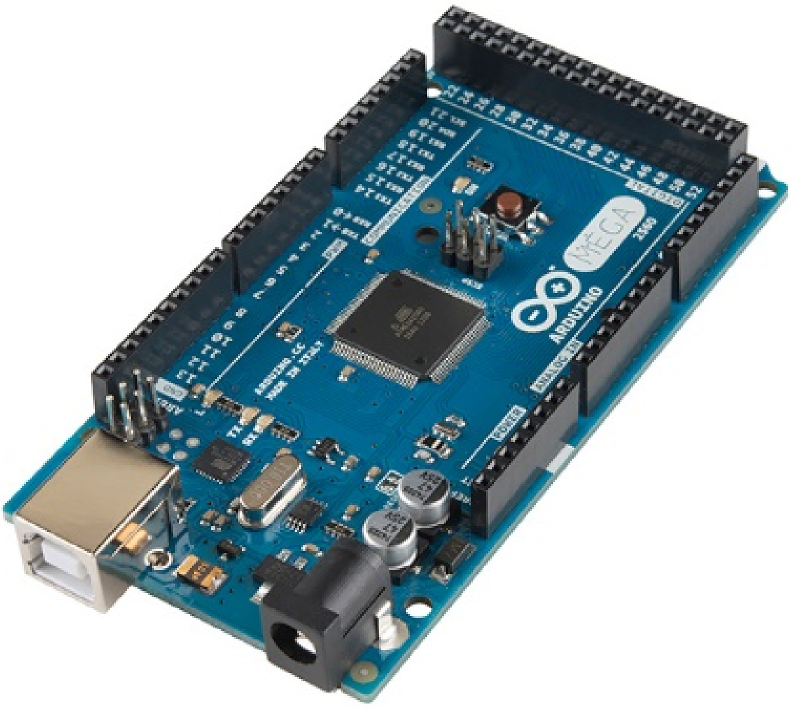


Figure 3 - The Arduino Mega 2560 R3 Microcontroller Board

Arduino Mega 2560 [11] is the board that is powered by ATmega2560. Its main advantages compared to other Arduino boards are the clock speed of 16MHz, the flash memory of 256Kbytes of the ATmega2560 chip and the 8Kbytes SRAM. The large flash and SRAM memories offer the Arduino vast of space for more interactive components to be added in future improvements.

The Arduino Mega provides 54 digital pins. They can be used as input or output and are operating at 5V. Each pin can provide or receive a maximum of 40mA and has an internal pull-up resistor of 20 – 50 kΩ. The pull-up resistor is disconnected by default but they can be accessed from software by setting the **pinMode()** as **INPUT_PULLUP**. This effectively inverts the behavior of the INPUT mode, where HIGH means the sensor is off, and LOW means the sensor is on. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).**
  Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).**
  These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.

- **PWM: 2 to 13 and 44 to 46.**
  Provide 8-bit PWM output with the analogWrite() function.

- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).**
  These pins support SPI communication using the SPI library.

- **LED: 13.**
  There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

- **TWI: 20 (SDA) and 21 (SCL).**
  Support TWI communication using the Wire library.
  (BMP085 sensor uses the Wire protocol to communicate with Arduino)

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 2^{10} different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

4.1.2.2. Arduino Ethernet Shield

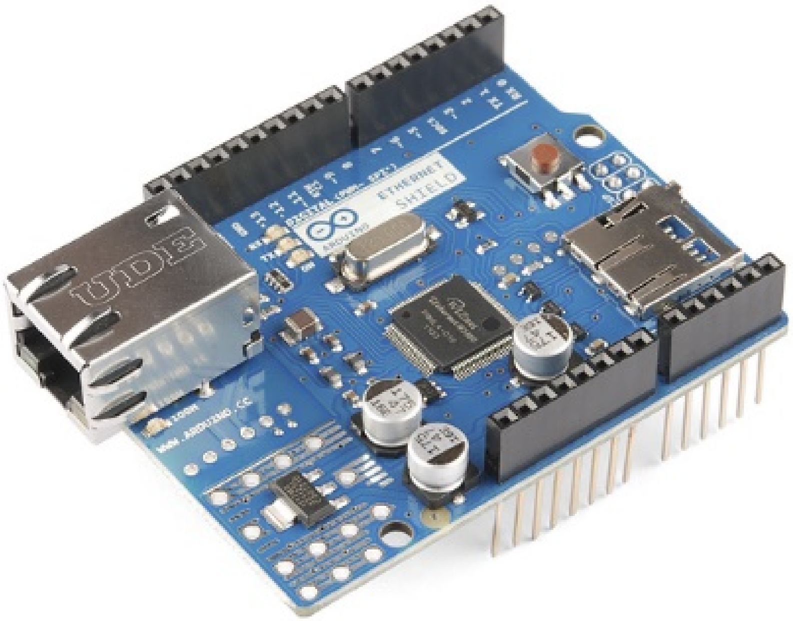


Figure 4 - The Arduino Ethernet Shield

The Ethernet shield [12] is powered by a Wiznet W5100 Ethernet controller and also includes a micro SD card slot for storage purposes. It is made exclusively for Arduino boards and connects over the SPI port.

The Wiznet W5100 provides a network (IP) stack capable of both TCP and UDP. It supports up to four simultaneous socket connections. The Ethernet shield connects to an Arduino board using long wire-wrap headers which extend through the shield. This keeps the pin layout intact and allows another shield to be stacked on top.
The Ethernet Shield has a standard RJ-45 connection, with an integrated line transformer and Power over Ethernet enabled.

It worth mentioning that pins SPI pins cannot be used as digital in or output any more when using the Ethernet Shield as they are being used by the SPI bus. Pins 4, 10 are used for the SD card and cannot be used as well. All the other pins are free to be used.

The Ethernet Shield is used in the project for data logging and as server for controlling the pan-tilt camera mount.

To integrate functionalities of the Ethernet Shield into the whole system, Ethernet [13] and SD [14] libraries are used.

4.1.3. Sensors

In this section the functionality and the characteristics of each sensors used by the prototype will be analyzed. In addition, connection and communication examples are given in this section.

4.1.3.1. Relative Humidity and Air Temperature (DHT22 or RHT03)

The RHT03 sensor from is Maxdetect made of two parts, a capacitive humidity sensor and a thermistor. There is also a basic chip inside that does some analog to digital conversion and offers a digital signal with the temperature and humidity.

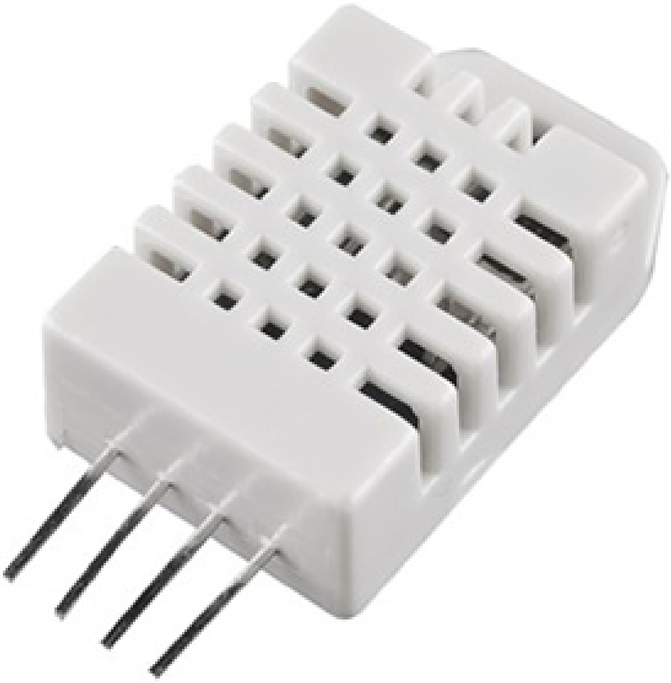


Figure 5 - RHT03

The RHT03 was chosen because of its high Resolution and Accuracy. Technical Specifications are shown in the table below [15].

Model	**DHT 22 also known as RHT03**
Power Supply	3.3 – 6V DC
Output Signal	Digital Signal via MaxDetect 1-wire bus
Sensing Element	Polymer Humidity Capacitor
Operating Range	Humidity: 0 – 100% RH Temperature: - 40 ~ 80 °C
Accuracy	Humidity: +- 2% RH (Max +-5% RH)

	Temperature: +- 0.5 °C
Resolution or Sensitivity	Humidity: 0.1% RH Temperature: 0.1 °C
Repeatability	Humidity: +- 1% RH Temperature: +- 0.2 °C

Table 1 - RHT03 Technical Specifications

Pin	Function
1	Power Supply
2	Data – Signal
3	NULL
4	Ground

Table 2 - RHT03 Pin Sequence Number (1, 2, 3, 4 from left to right)

Communication and signal

Wire Bus is used for communication between the MCU and RHTY03.

Illustration of MaxDetect's wire bus:

Data is comprised of integral and decimal part, the following is the formula for data.

- 8bit integral Relative Humidity data
- 8bit decimal Relative Humidity data
- 8bit integral Temperature data
- 8bit decimal Temperature data
- 8but checksum

When MCU send start signal, RHT03 change from standby-status to running-status. When MCU finishes sending the start signal, RHT03 will send response signal of 40-bit data that reflect the relative humidity and temperature to MCU. Without start signal from MCU, RHT03 will not give response signal to MCU. One start signal for one response data from RHT03 that reflect the relative humidity and temperature. RHT03 will change to standby status when data collecting finished if it don't receive start signal from MCU again.

Below one can find the overall communication process. The interval of whole process must beyond two seconds.

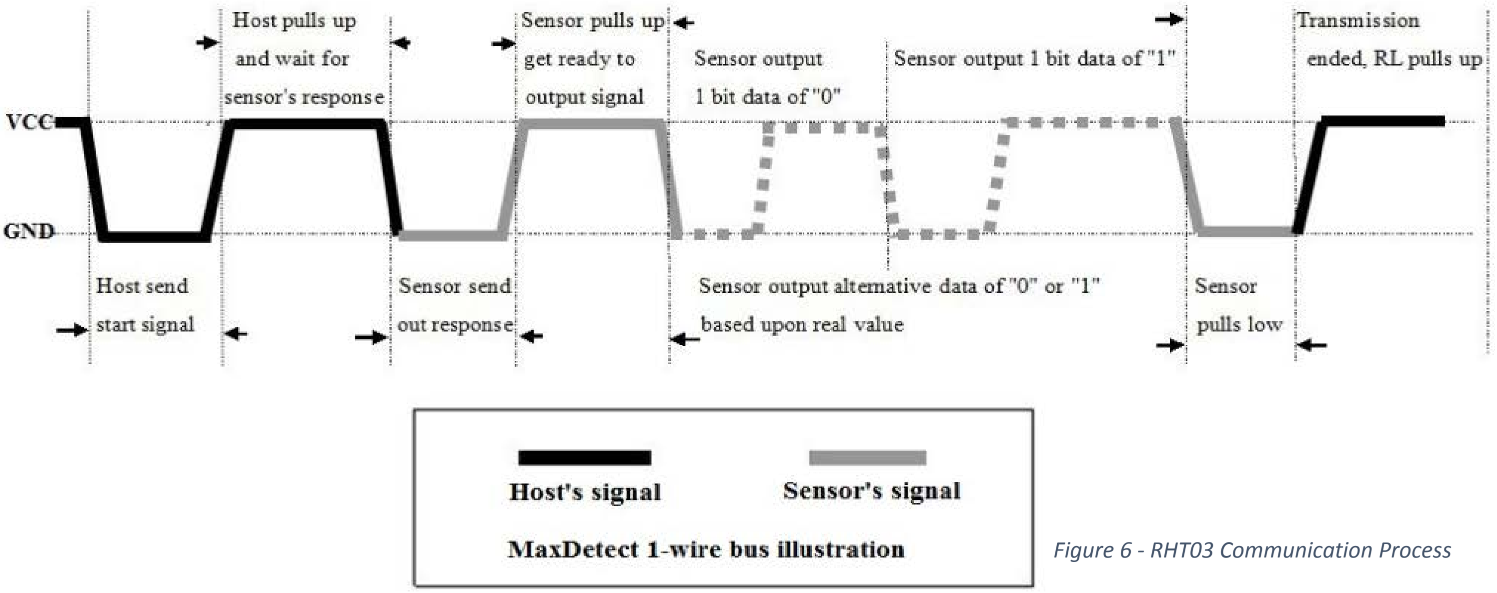

Host pulls up and wait for sensor's response
Sensor pulls up get ready to output signal
Sensor output 1 bit data of "0"
Sensor output 1 bit data of "1"
Transmission ended, RL pulls up
VCC
GND
Host send start signal
Sensor send out response
Sensor output alternative data of "0" or "1" based upon real value
Sensor pulls low
Host's signal
Sensor's signal
MaxDetect 1-wire bus illustration

Figure 6 - RHT03 Communication Process

4.1.3.2. Air Pressure and Temperature (BMP085)

For air pressure readings and temperature the digital sensor BMP085 from Bosch Sensortec is used [16]. It was chosen for the prototype because of its good accuracy through individual precise calibration and its long term stability feature. The main characteristics are summarized in the table below.

Parameter	Condition	Min	Typical	Max	Unit
Resolution	Typical	0.04	0.01	0.01	hPA
Accuracy	Typical		+- 1.0		hPA
Response Time	Mode	4.5		25.5	Ms
Operation Range		700		1100	hPA
Supply Voltage		1.8	2.5	3.6	Volts

Table 3 - Main Properties of the BMP085

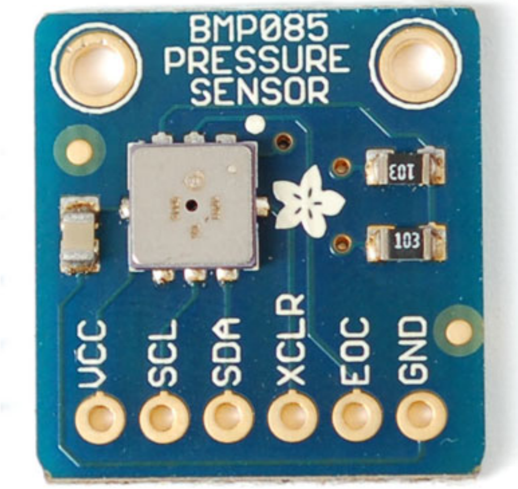


Figure 6 - BMP085

It is based on piezo - resistive Technology for EMC robustness, high accuracy and linearity as well as long term stability. It is designed to be connected directly to a MCU via the I^{2}C bus. The pressure and temperature data has to be compensated by the calibration data of the E^{2}PROM of the BMP085.

The sensor consists of a piezo – resistive sensor, an analog to digital converter and a control unit with

E²PROM and a serial I²C interface. It delivers the uncompensated value of pressure and temperature. The E²PROM has stored 176 bit of individual calibration data. This is used to compensate offset, temperature dependence and other parameters of the sensor.

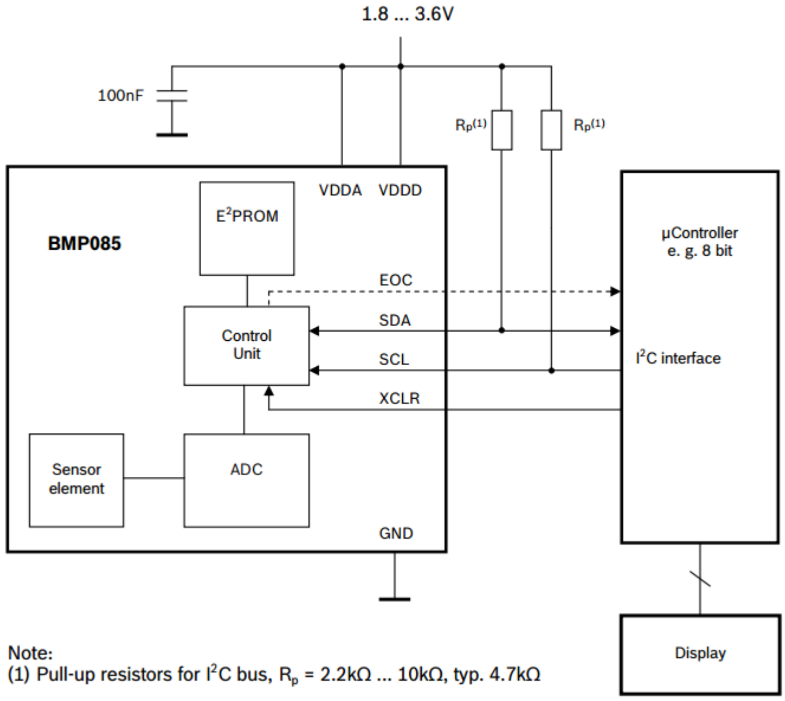


Figure 7 - BMP085 Application Circuit

The I²C interface and a read-only-memory register (E²PROM) are part of the control unit of the BMP085. In the E²PROM eleven 16bit calibration coefficients are stored and used to compensate offset of temperature and pressure readings. As temperature is a factor for air pressure calculation, it has to be known to calculate the true pressure. Reading of the temperature can be done with the piezo-resistive sensor as well.

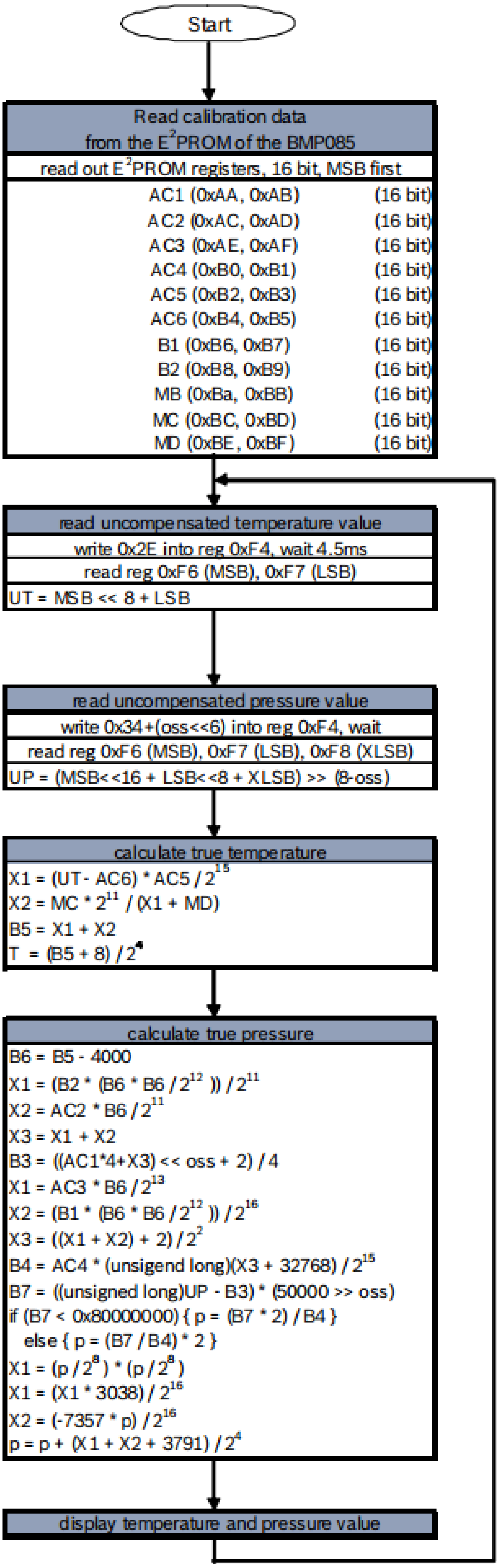

Start
Read calibration data
from the E^2PROM of the BMP085
read out E^2PROM registers, 16 bit, MSB first
AC1 (0xAA, 0xAB) (16 bit)
AC2 (0xAC, 0xAD) (16 bit)
AC3 (0xAE, 0xAF) (16 bit)
AC4 (0xB0, 0xB1) (16 bit)
AC5 (0xB2, 0xB3) (16 bit)
AC6 (0xB4, 0xB5) (16 bit)
B1 (0xB6, 0xB7) (16 bit)
B2 (0xB8, 0xB9) (16 bit)
MB (0xBa, 0xBB) (16 bit)
MC (0xBC, 0xBD) (16 bit)
MD (0xBE, 0xBF) (16 bit)
read uncompensated temperature value
write 0x2E into reg 0xF4, wait 4.5ms
read reg 0xF6 (MSB), 0xF7 (LSB)
UT = MSB << 8 + LSB
read uncompensated pressure value
write 0x34+(oss<<6) into reg 0xF4, wait
read reg 0xF6 (MSB), 0xF7 (LSB), 0xF8 (XLSB)
UP = (MSB<<16 + LSB<<8 + XLSB) >> (8-oss)
calculate true temperature
X1 = (UT - AC6) * AC5 / 2^{15}
X2 = MC * 2^{11} / (X1 + MD)
B5 = X1 + X2
T = (B5 + 8) / 2^4
calculate true pressure
B6 = B5 - 4000
X1 = (B2 * (B6 * B6 / 2^{12})) / 2^{11}
X2 = AC2 * B6 / 2^{11}
X3 = X1 + X2
B3 = ((AC1*4+X3) << oss + 2) / 4
X1 = AC3 * B6 / 2^{13}
X2 = (B1 * (B6 * B6 / 2^{12})) / 2^{16}
X3 = ((X1 + X2) + 2) / 2^2
B4 = AC4 * (unsigend long)(X3 + 32768) / 2^{15}
B7 = ((unsigned long)UP - B3) * (50000 >> oss)
if (B7 < 0x80000000) { p = (B7 * 2) / B4 }
else { p = (B7 / B4) * 2 }
X1 = (p / 2^8) * (p / 2^8)
X1 = (X1 * 3038) / 2^{16}
X2 = (-7357 * p) / 2^{16}
p = p + (X1 + X2 + 3791) / 2^4
display temperature and pressure value

Figure 8 - BMP085 Temperature and Pressure Reading Process

A measurement sequence with the complete algorithm is shown in Figure 9. It has to be implemented to the MCU according to this order.

After the MC has sent a start sequence, the calibration data is requested from the E²PROM registers. The 16 constants need to be read out only once and are stored on the MCU. Then the MCU has to wait for uncompensated temperature (UT) and pressure (UP) readings from the sensors. The stored calibration data is now used to calculate temperature in °C and pressure in Pa. After applying the algorithms, the sensor waits for the next measurement command.

As additional factor for true pressure calculation, an oversampling mode can be chosen to set the internal sampling of the sensor for one measurement. Changing the mode to a higher resolution will increase the accuracy of a measurement but also has an impact on energy consumption, reaction time and the long term stability [17].

Table 4 gives an overview about the four oversampling setting modes. As the weather station is designed for long and continuous measurements, standard mode will be sufficient.

Mode	**Oversampling Setting**	**Internal Samples**	**Conversion Time (ms)**	**Noise (hPA)**
Ultra Low Power	0	1	4.5	0.0.6
Standard	1	2	7.5	0.05
High Resolution	2	4	13.5	0.0.4
Ultra High Resolution	3	8	25.5	0.03

Table 4 - Measurement modes of the BMP085 and resolution effects

As mentioned before, the BMP085 sensor is using the I^{2}C protocol to communicate with the MCU and has to be connected to SDA and SCL pins of the Arduino Mega as shown below.

BMP085	Arduino Mega Pin
SDA	20
SCL	21
XCLR	Not Connected
EOC	Not Connected
GND	GND
VCC	3.3V

Table 5 - BMP085 Pin Connections to Arduino Mega

The altitude of the instrument has to be taken into account as well, because air pressure decreases with increasing height. For getting comparable values it has to be calculated to sea level pressure [6].

Working with the BMP085 should be done carefully, because it could be damaged by shocks or when getting in contact with water. It should be handled as Electrostatic Sensitive Device (ESD) [18] too. Therefore, it should be installed in a dry environment with constant temperature.

4.1.3.3. Precipitation

Rainfall is measured by a Tipping Bucket Rain Gauge included in the Weather Meters Kit [19] available from Sparkfun. Each time the self – emptying tipping bucket is emptied, a reed switch closes once. This corresponds to 0.2794 mm of rain fall [20]. The state change is caught by the MCU using Arduino's Interrupt input.

A problem that frequently appears at trying to capture the state change of a switch is called bouncing problem. State change means that the input at a digital pin on the Arduino changes from high to low (equals from 5V to 0V) or vice versa. The problem is that each switching contains interferences in the signal before it reaches the level of 5V or 0V. This can lead to multiple counts from only one state switch. A visualization of the problem is shown in Figure 11.

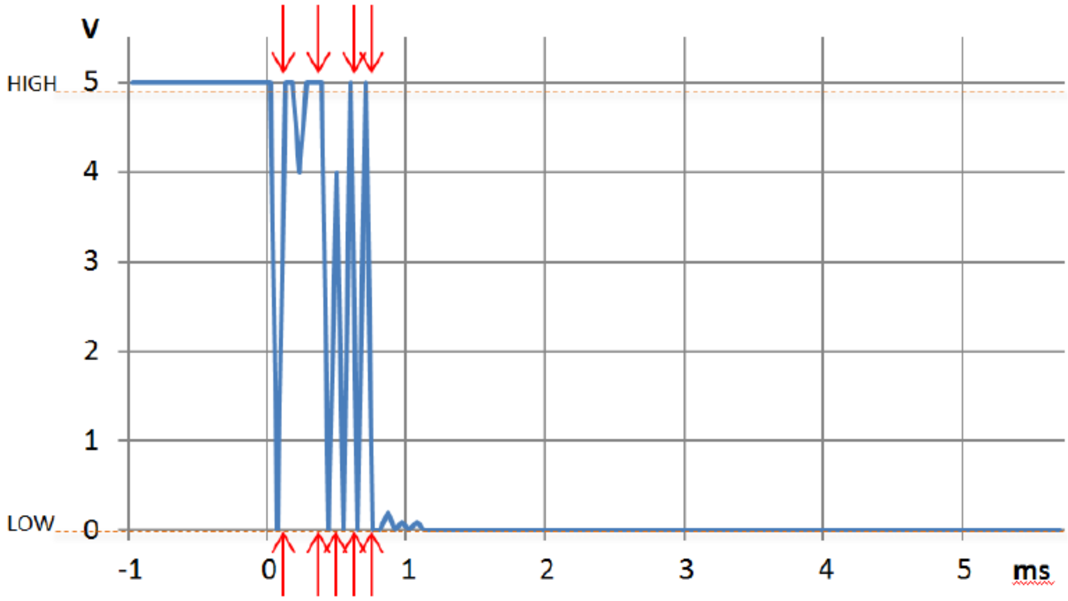
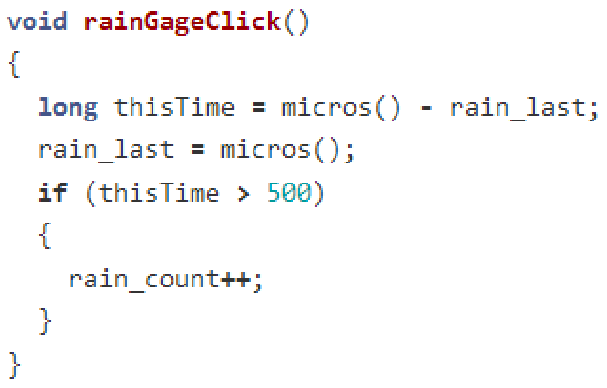


Figure 10 - Bouncing problem at a digital input. MCU will read nine state changes instead of one

To overcome this problem there are two solutions. The first is to integrate a capacitor to the circuit and the second is to define a software method with a debounce interval of some microseconds (µs), which is called very time a state change is recognized.

For the project the second method is used. The implementation is show below.



```
void rainGageClick()
{
  long thisTime = micros() - rain_last;
  rain_last = micros();
  if (thisTime > 500)
  {
    rain_count++;
  }
}
```

Figure 11 – Rain Reading And Switch Debouncing Implementation

The code above will ignore any interrupt that occurs within 500µs (0.0005s) of the previous one. This technique will limit the max rain this configuration can measure to a very high value that do not corresponds to an earthly value.

Wiring the Rain Gauge

The two wires coming from the rain gauge, through the RJ11 cable, have to be connected to the Arduino Mega as shown in the figure below.

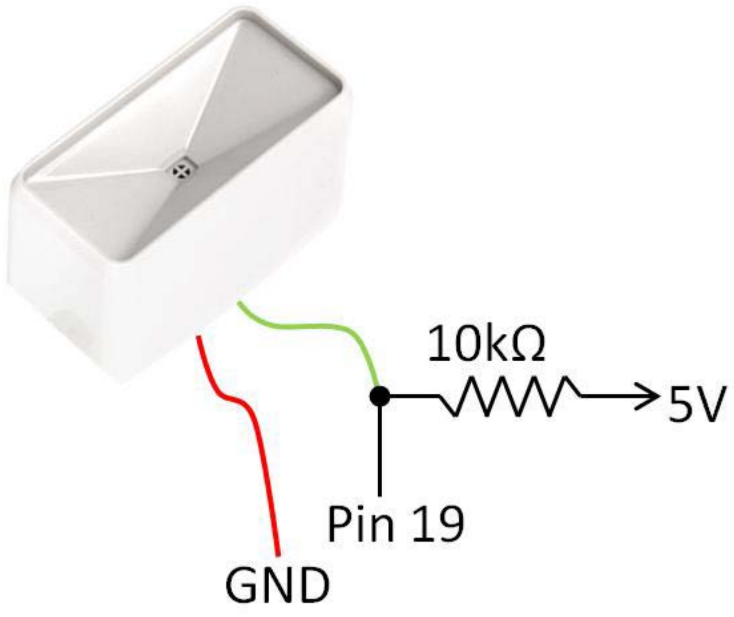


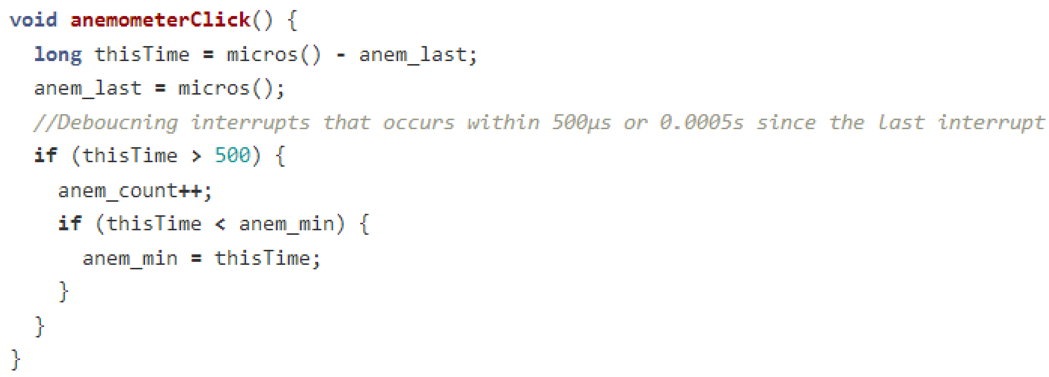
Figure 12 - Connection of the Rain Gauge to the Arduino Mega

The red line must be connected to Ground while the green line is pulled up with a 10kΩ resistor to 5V. The output of the sensor has to be connected to digital pin 19 on the Arduino Mega.

A disadvantage of this method is that only rain fall can be measured. To include snow fall in the precipitation measurement a self-heated device would be needed. Temperatures below 0° Celsius could also lock up the anemometer. This should be taken into account when analyzing the data of the rain gauge.

4.1.3.4. *Wind Speed*

The wind speed is measured with a cup – type anemometer included in Sparkfun’s Weather Meters Kit [19]. In principle, it works exactly like the rain gauge. At every turn, it closes a contact as a magnet moves past a switch, which is connected to a digital counter. 0.67 meter per second or 1.29651 knots causes the switch to close once per second [20].



```
void anemometerClick() {
  long thisTime = micros() - anem_last;
  anem_last = micros();
  //Deboucning interrupts that occurs within 500µs or 0.0005s since the last interrupt
  if (thisTime > 500) {
    anem_count++;
    if (thisTime < anem_min) {
      anem_min = thisTime;
    }
  }
}
```

Figure 13 - Anemometer Reading And Switch Debouncing Implementation

Switch debouncing is implemented for the anemometer exactly the same way as the rain gauge. The max wind speed this configuration can measure is limited to about 2,700 knots that does not affect the system since that kind of values are not earthly.

Wiring the Anemometer

The anemometer is connected with a two wired cable to the wind vane. The two center wires (yellow and red) of the cable coming from the wind vane are used by the anemometer. Again, the red cable needs a pull-up resistor before connecting it to 5V. The yellow cable is connected to GND.

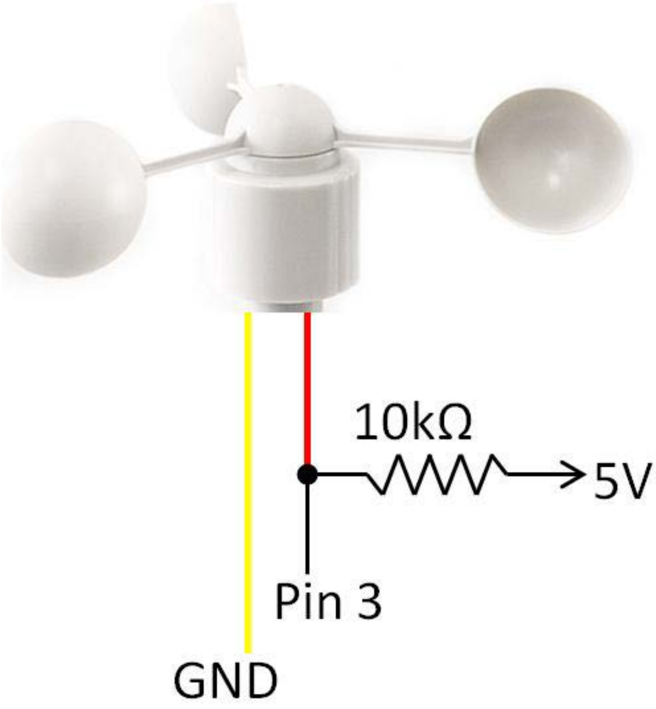


Figure 14 - Connection of the Anemometer to the Arduino Mega

4.1.3.5. Wind Direction

The wind vane of the Weather Sensor Assembly [19] contains eight switches, each of them connected to a resistor with different resistance values. A magnet switch can close two switches at once which allows to indicate 16 different positions. The voltage of the output cable can be measured at an analog input on the Arduino Mega. Directions for corresponding voltage output can be found in table 6.

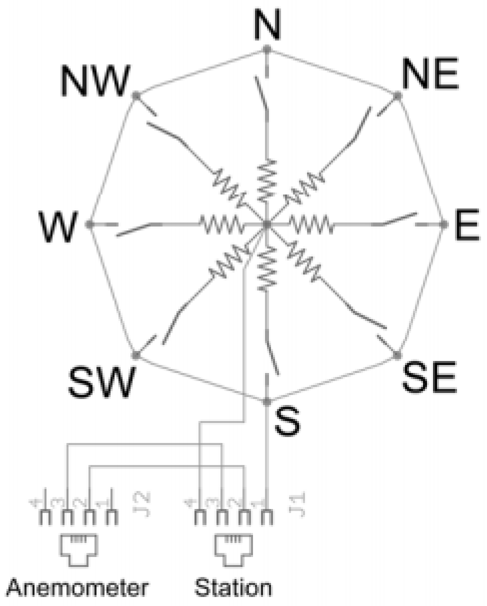


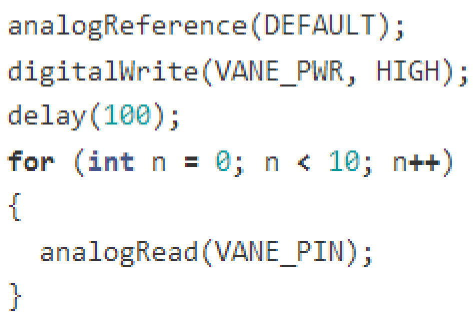
Figure 15 - Wind Vane resistor arrangement

Direction (Compass)	Direction (Degrees)	Voltage Output
N	0 / 360	3.84
N - NE	22.5	1.98
NE	45	2.25
E - NE	67.5	0.41
E	90	0.45
E – SE	112.5	0.32
SE	135	0.90
S – SE	157.5	0.62
S	180	1.40
S – SW	202.5	1.19
SW	225	3.08
W – SW	247.5	2.93
W	270	4.62
W – NW	292.5	4.04
NE	315	4.78 (4.33)
N – NW	337.5	3.43

Table 6 - Analog Voltage output with corresponding direction

Testing of the wind vane showed that voltage values at 315^{o} is not 4.78V as mentioned at the datasheet. A 10kΩ resistor in series with a 64.9kΩ resistor, the voltage drop across the vane will be 4.33V.

For better power management, the wind vane is powered by the digital pin VANE_PWR right before it is used. A 100ms delay is inserted after the power pin is switched to HIGH just to let things settle down.



```
analogReference(DEFAULT);
digitalWrite(VANE_PWR, HIGH);
delay(100);
for (int n = 0; n < 10; n++)
{
  analogRead(VANE_PIN);
}
```

Figure 16 - Wind Vane Powering Up

Wiring the Wind Vane

The green and black outer wires of the cable from the wind vane have to be connected to the Arduino Mega as shown in Figure 18.

The wind vane and the anemometer shares the same RJ11 cable.

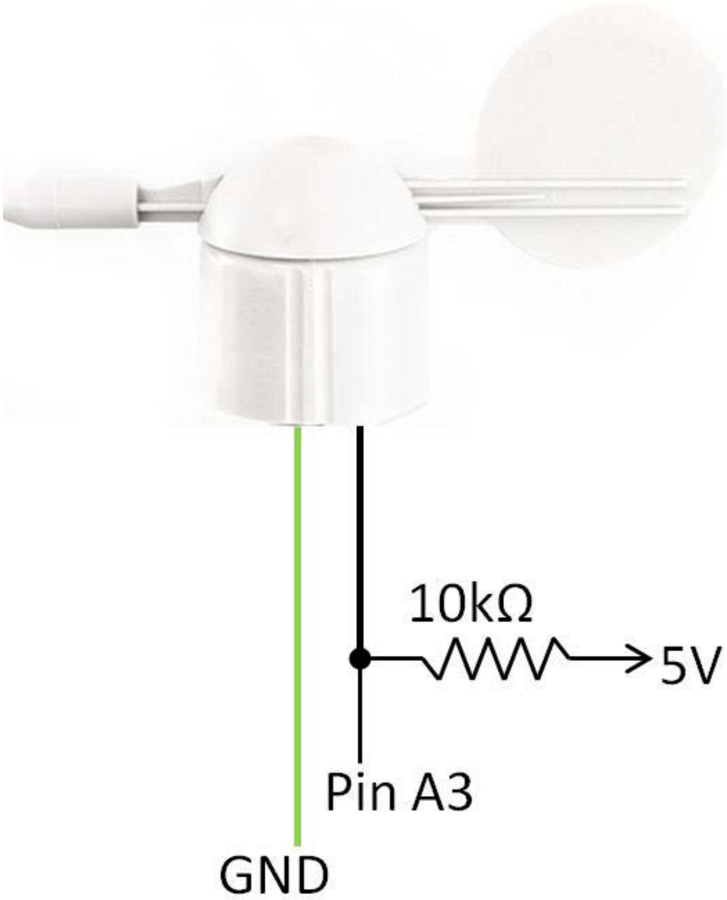


Figure 17 - Connection of the Wind Vane to the Arduino Mega

As mentioned before, the incoming voltage can be measured with an analog pin on the Arduino Mega. The values, which are read out at the analog input, have to be converted from 10 bit (0-1023) to a scale between 0 and 5V (0 to 5000). The resulting voltage can then be used to grab the corresponding value from a list according to Table 6.

4.1.4. Prototype Architecture

A schematic overview, created on Fritzing [21], of the wiring is show in Figure 19.

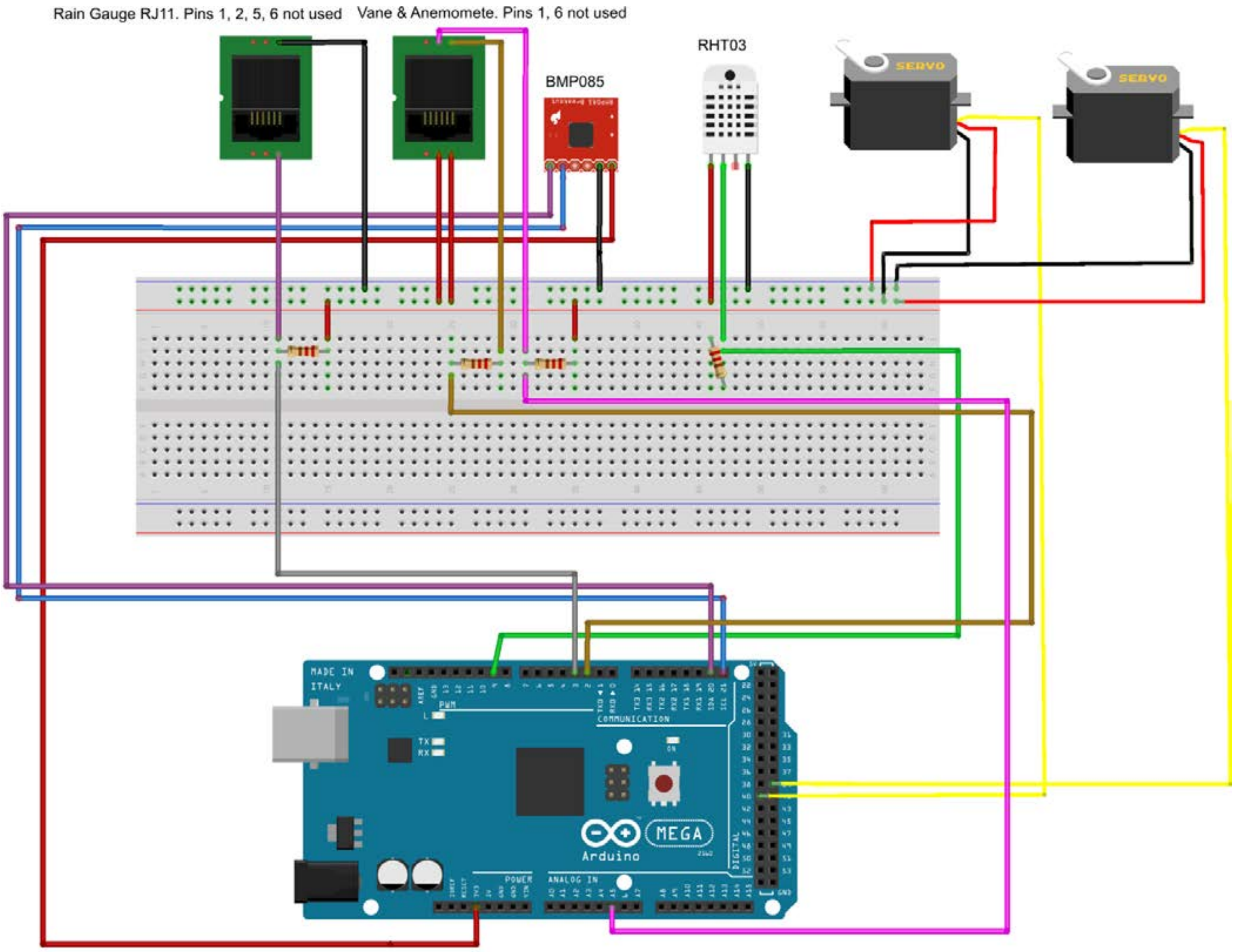

Rain Gauge RJ11. Pins 1, 2, 5, 6 not used
Vane & Anemomete. Pins 1, 6 not used
BMP085
RHT03
SERVO
SERVO
MEGA
Arduino

Figure 18 - Schematic wiring of all components

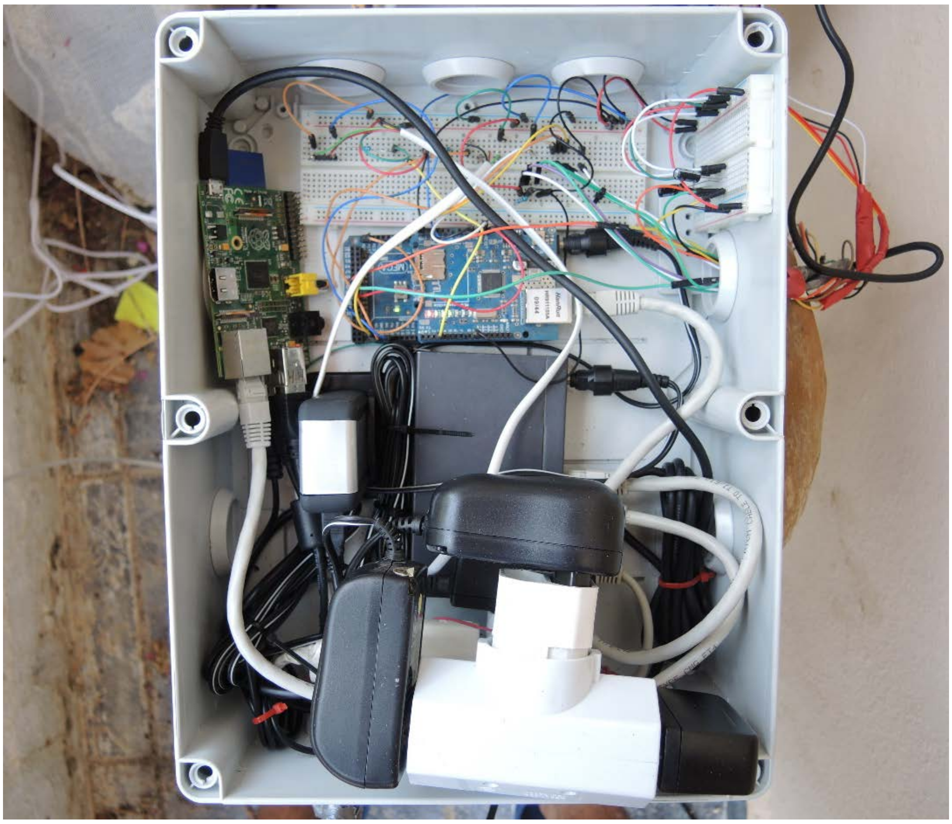


Figure 19 - The Interior of the PVC Waterproof box

The waterproof PVC box hosts all the hardware used for the project. The parts that reside inside the box are:

1. Arduino Mega 2560 R3
2. Raspberry PI
3. Two breadboards
4. External Powered USB Hub
5. Ethernet Switch
6. Surge
7. Power Adapters

Normally, in order for the anemometer to be accurate, any surrounding objects must be 4 time as far as they are higher than the anemometer. For example, if the anemometer is 2 meters off the ground, and my house is 8 meters tall, the anemometer would need to be 26.6 meters from the house to be able to measure the wind speed accurately. In my case, nothing close to this leeway is possible, but I still setup the anemometer just for the educational experience.

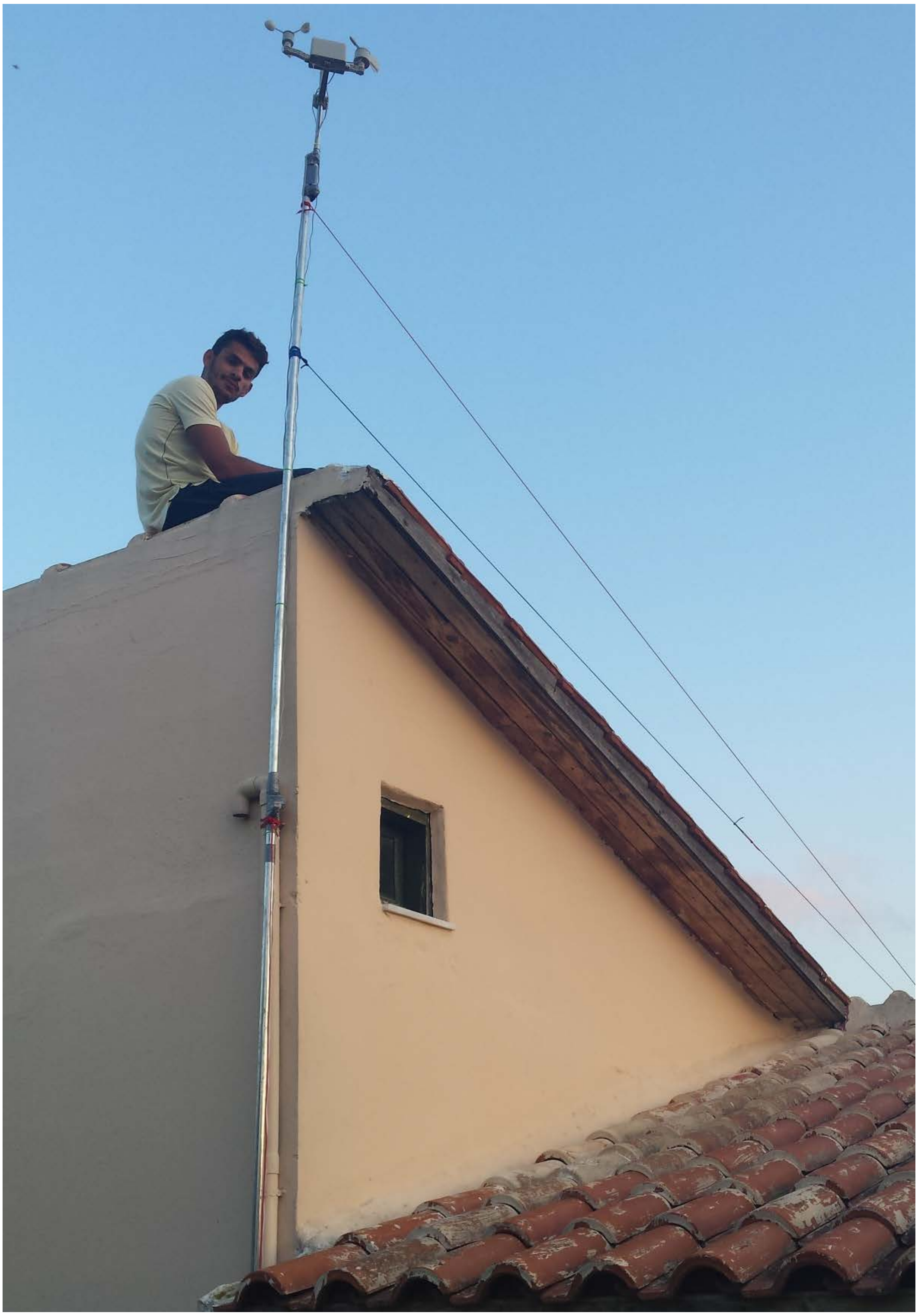
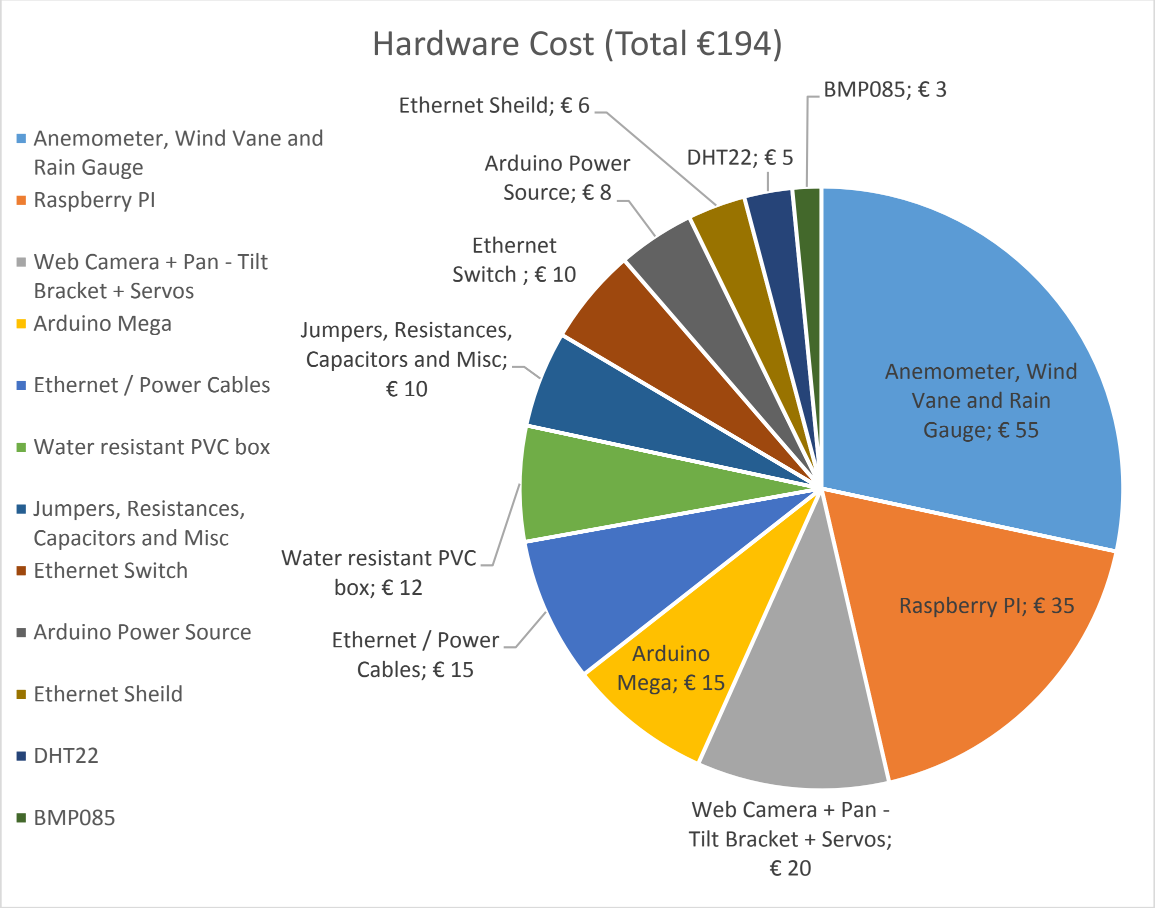


Figure 20 - The column in which the Weather Meters are mounted. My best friend Giorgos helped me to set up the column.

Hardware Cost



4.2. In Depth Project Analysis

Following the Hardware components review, the Software analysis will dive deep into the main parts of the code that was used for the project to work.

This section will be split into four parts:

- Arduino Sketch
- Server – Side PHP scripts
- Android Application
- Time Lapse Video
- Web Site

4.2.1. Arduino Sketch [23]

The Arduino Sketch is the most basic part of the project. It is the beginning. If this does not work perfectly nothing else can be done.

The Sketch consists of three major parts.

- The sensors values readings
- The HTTP Requests
- The Pan – Tilt camera mount control

4.2.1.1. Sensors Values Readings

Each sensor has its own style for reading its value.

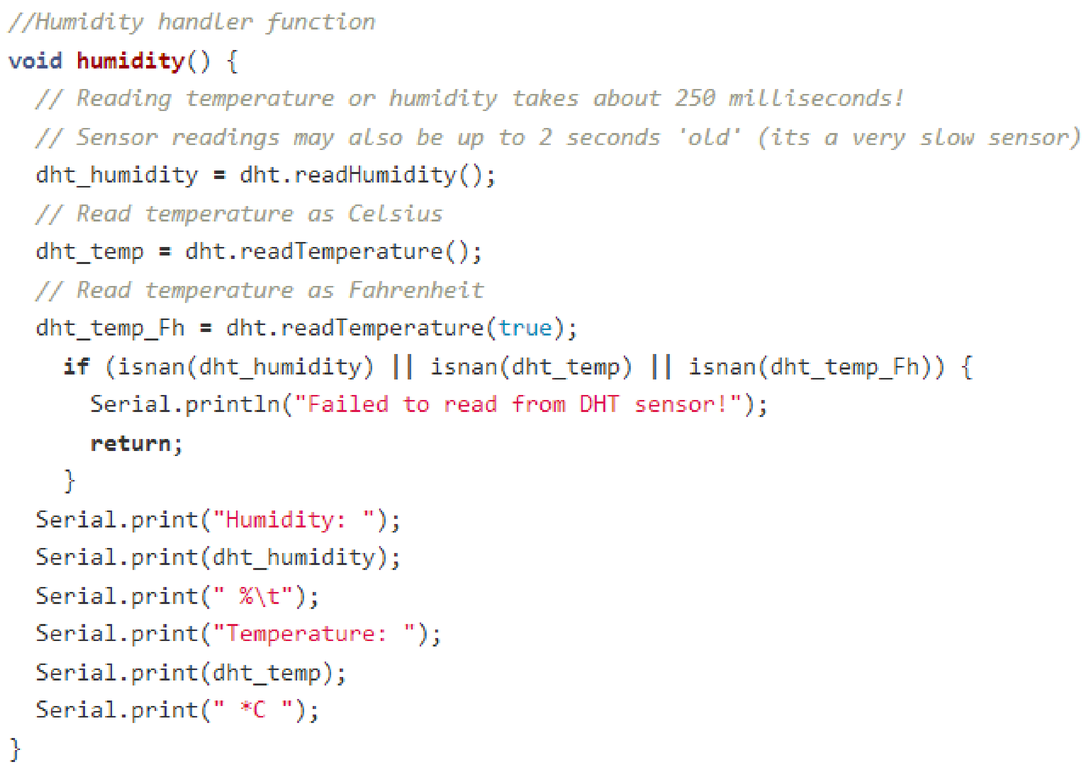
RHT03

To integrate RHT03's functionality to the project the library written by Adafruit Industries [22] is used.

First of all, the library must be imported and a DHT object is created.

```
#include <DHT.h>

DHT dht(DHTPIN, DHTTYPE);
```

To get RHT03's readings the library does the heavy job.



```
//Humidity handler function
void humidity() {
  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  dht_humidity = dht.readHumidity();
  // Read temperature as Celsius
  dht_temp = dht.readTemperature();
  // Read temperature as Fahrenheit
  dht_temp_Fh = dht.readTemperature(true);
    if (isnan(dht_humidity) || isnan(dht_temp) || isnan(dht_temp_Fh)) {
      Serial.println("Failed to read from DHT sensor!");
      return;
    }
  Serial.print("Humidity: ");
  Serial.print(dht_humidity);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(dht_temp);
  Serial.print(" *C ");
}
```

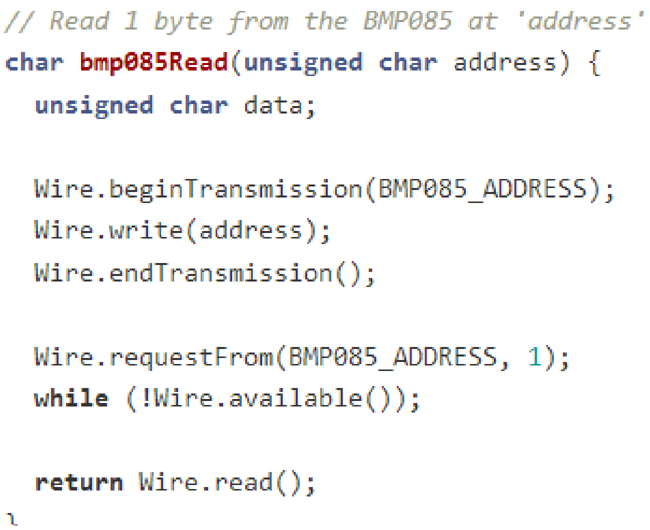
The code above is very simple, since DHT library undertakes to deal with all the heavy job. At the beginning, temperature and humidity values are stored at the global variables ***dht_humidity, dht_temp***. Then an if else statement checks if the values read are numbers by calling **isnan** function (Is Not A Number). This check is being done to ensure that everything works well in the sensor.

BMP035

Collecting BMP035's values was a challenging task, since the algorithm described in Figure 10 had to be implemented in the MCU.

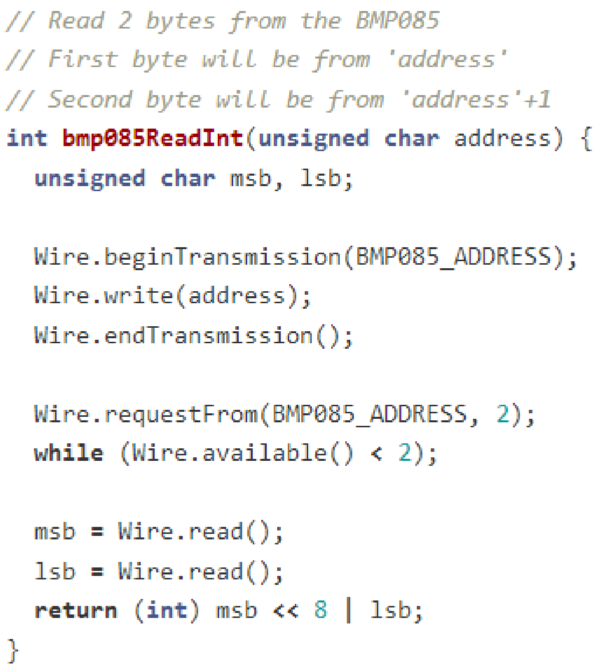
The first thing during the development of the sketch, was to make sure that I can actually communicate with the sensor. For many digital sensors, once the communication is established it is halfway done.

First of all, the functions used to read 8 and 16 bit values from the BMP085 were implemented.



```
// Read 1 byte from the BMP085 at 'address'
char bmp085Read(unsigned char address) {
  unsigned char data;

  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(BMP085_ADDRESS, 1);
  while (!Wire.available());

  return Wire.read();
}
```

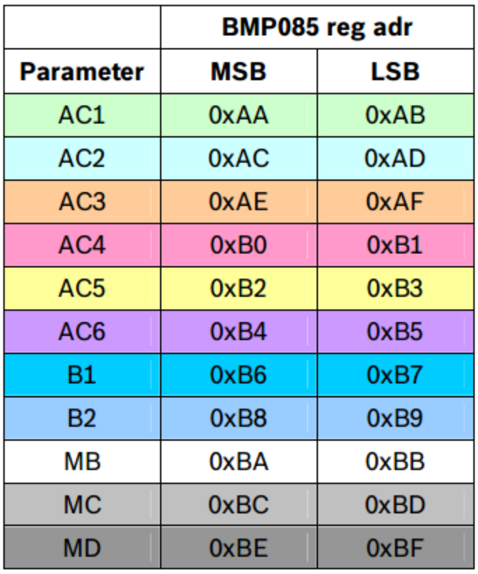
Figure 21 - Function to read 1 byte from the bmp085



```
// Read 2 bytes from the BMP085
// First byte will be from 'address'
// Second byte will be from 'address'+1
int bmp085ReadInt(unsigned char address) {
  unsigned char msb, lsb;

  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(BMP085_ADDRESS, 2);
  while (Wire.available() < 2);

  msb = Wire.read();
  lsb = Wire.read();
  return (int) msb << 8 | lsb;
}
```

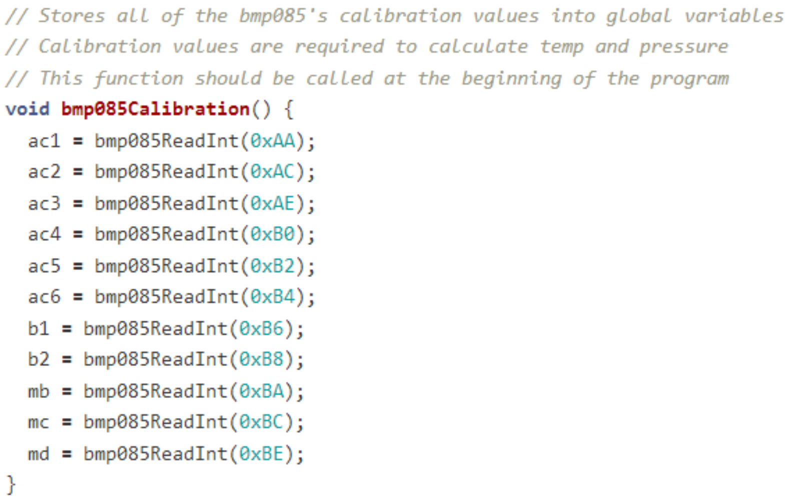
Figure 22 - Function to read 2 bytes from the bmp085

According to sensor's datasheet, the 176 bit E^{2}PROM memory is partitioned in eleven words of 16 bit each. These contain eleven calibration coefficients. Before the first calculation of temperature and pressure, the master reads out the E^{2}PROM data (The function at Figure 24 is responsible for this task. It is being called only once during the setup process). The data communication can be checked by checking that none of these words has the value of 0 or 0xFFFF. The parameter association with the Registers Addresses can be seen in the Table 7.



	BMP085 reg adr	
Parameter	**MSB**	**LSB**
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

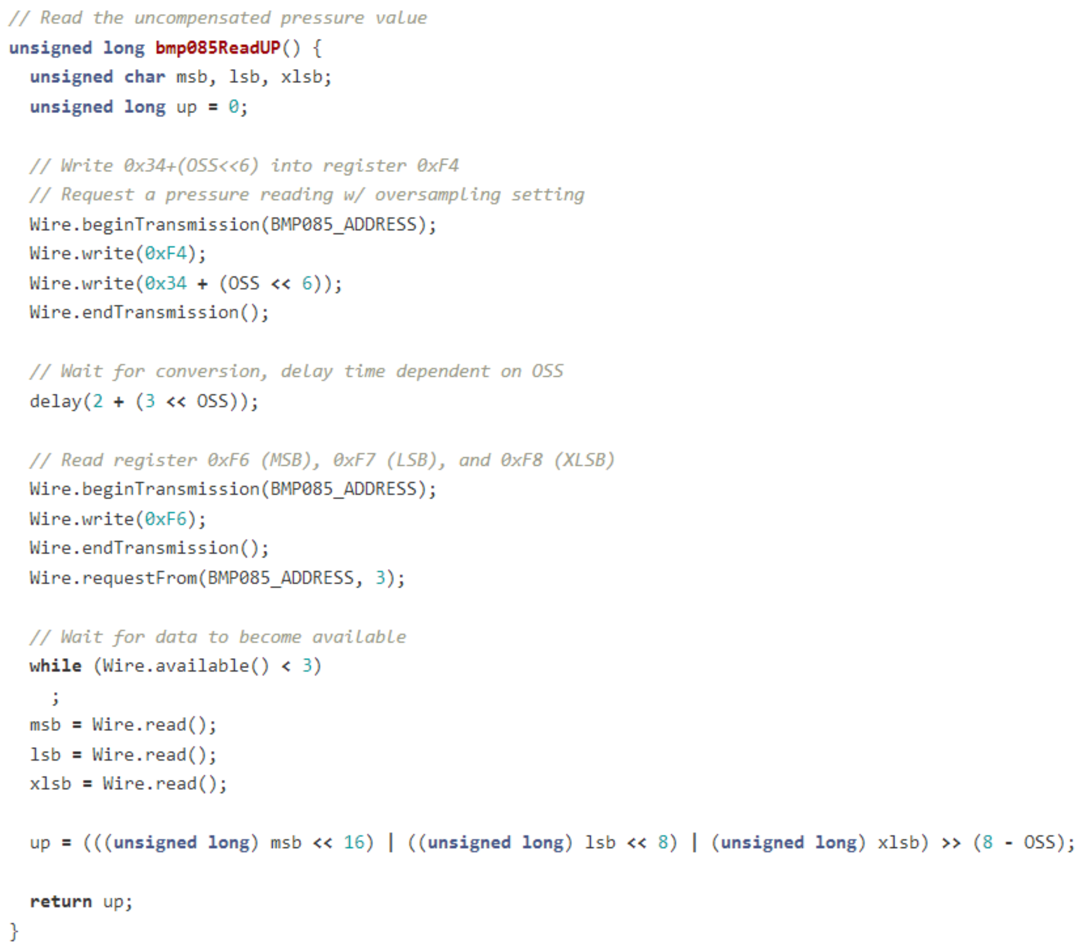
Table 7 - BMP085 Calibration Coefficients



```
// Stores all of the bmp085's calibration values into global variables
// Calibration values are required to calculate temp and pressure
// This function should be called at the beginning of the program
void bmp085Calibration() {
  ac1 = bmp085ReadInt(0xAA);
  ac2 = bmp085ReadInt(0xAC);
  ac3 = bmp085ReadInt(0xAE);
  ac4 = bmp085ReadInt(0xB0);
  ac5 = bmp085ReadInt(0xB2);
  ac6 = bmp085ReadInt(0xB4);
  b1 = bmp085ReadInt(0xB6);
  b2 = bmp085ReadInt(0xB8);
  mb = bmp085ReadInt(0xBA);
  mc = bmp085ReadInt(0xBC);
  md = bmp085ReadInt(0xBE);
}
```

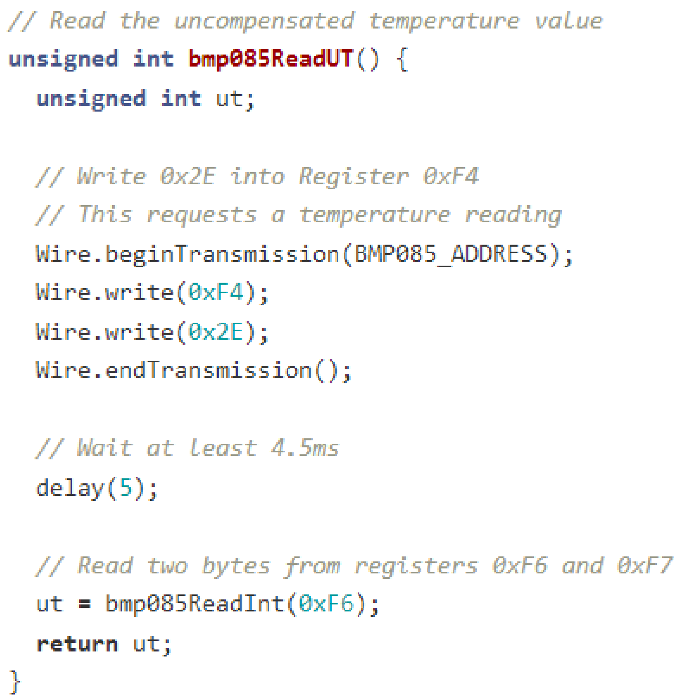
Figure 23 - BMP085 Calibration

Once we have read the calibration values, we just need two more variables in order to calculator temperature and pressure. 'UT' and 'UP' are the uncompensated temperature and pressure values. Every time the temperature or pressure is measured, first ‘UT’ and ‘UP’ values must be read. They are the starting point for calculating the actual temperature and pressure values. The uncompensated temperature is an unsigned 16-bit integer number while uncompensated pressure is an unsigned 32-bit long number. The two functions can be seen at Figure 22, Figure 23.



```
// Read the uncompensated pressure value
unsigned long bmp085ReadUP() {
  unsigned char msb, lsb, xlsb;
  unsigned long up = 0;

  // Write 0x34+(OSS<<6) into register 0xF4
  // Request a pressure reading w/ oversampling setting
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF4);
  Wire.write(0x34 + (OSS << 6));
  Wire.endTransmission();

  // Wait for conversion, delay time dependent on OSS
  delay(2 + (3 << OSS));

  // Read register 0xF6 (MSB), 0xF7 (LSB), and 0xF8 (XLSB)
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF6);
  Wire.endTransmission();
  Wire.requestFrom(BMP085_ADDRESS, 3);

  // Wait for data to become available
  while (Wire.available() < 3)
    ;
  msb = Wire.read();
  lsb = Wire.read();
  xlsb = Wire.read();

  up = (((unsigned long) msb << 16) | ((unsigned long) lsb << 8) | (unsigned long) xlsb) >> (8 - OSS);

  return up;
}
```

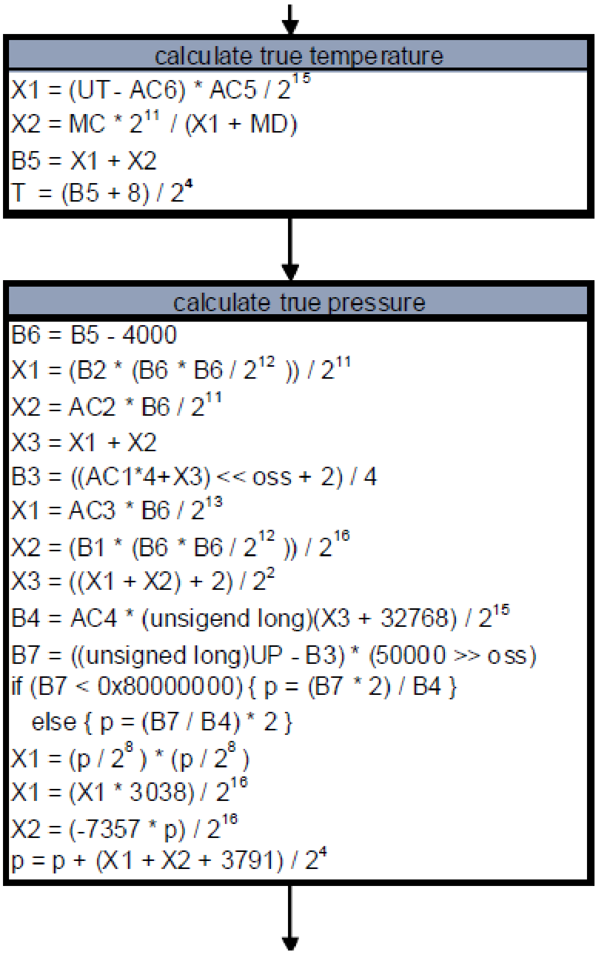
Figure 24 - BMP085 Read Uncompensated Pressure Function



```
// Read the uncompensated temperature value
unsigned int bmp085ReadUT() {
  unsigned int ut;

  // Write 0x2E into Register 0xF4
  // This requests a temperature reading
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF4);
  Wire.write(0x2E);
  Wire.endTransmission();

  // Wait at least 4.5ms
  delay(5);

  // Read two bytes from registers 0xF6 and 0xF7
  ut = bmp085ReadInt(0xF6);
  return ut;
}
```

Figure 25 - BMP085 Read Uncompensated Temperature Function

For last part of the calculation process, two functions are needed to take care of the following:



calculate true temperature

$X1 = (UT - AC6) * AC5 / 2^{15}$

$X2 = MC * 2^{11} / (X1 + MD)$

$B5 = X1 + X2$

$T = (B5 + 8) / 2^4$

calculate true pressure

$B6 = B5 - 4000$

$X1 = (B2 * (B6 * B6 / 2^{12})) / 2^{11}$

$X2 = AC2 * B6 / 2^{11}$

$X3 = X1 + X2$

B3 = ((AC1*4+X3) << oss + 2) / 4

$X1 = AC3 * B6 / 2^{13}$

$X2 = (B1 * (B6 * B6 / 2^{12})) / 2^{16}$

$X3 = ((X1 + X2) + 2) / 2^2$

B4 = AC4 * (unsigend long)(X3 + 32768) / 2^{15}

B7 = ((unsigned long)UP - B3) * (50000 >> oss)

if (B7 < 0x80000000) { p = (B7 * 2) / B4 }

  else { p = (B7 / B4) * 2 }

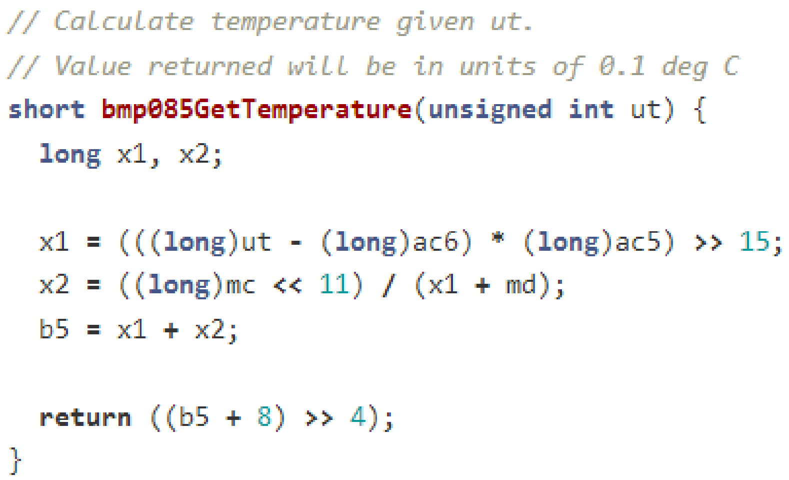
$X1 = (p / 2^8) * (p / 2^8)$

$X1 = (X1 * 3038) / 2^{16}$

$X2 = (-7357 * p) / 2^{16}$

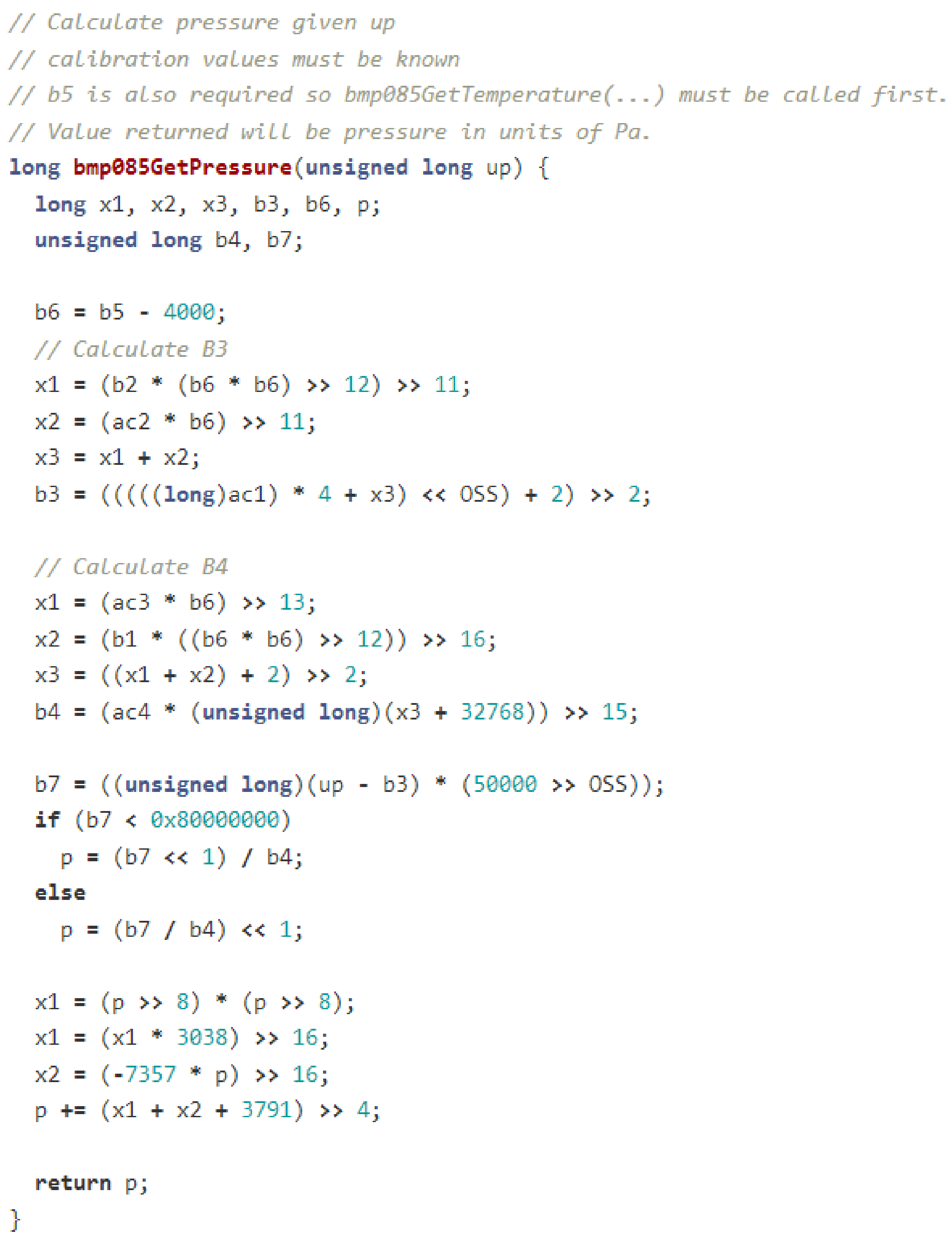
$p = p + (X1 + X2 + 3791) / 2^4$

The temperature calculation is rather a simple task, but the pressure ones are more complicated.



```
// Calculate temperature given ut.
// Value returned will be in units of 0.1 deg C
short bmp085GetTemperature(unsigned int ut) {
  long x1, x2;

  x1 = (((long)ut - (long)ac6) * (long)ac5) >> 15;
  x2 = ((long)mc << 11) / (x1 + md);
  b5 = x1 + x2;

  return ((b5 + 8) >> 4);
}
```

Figure 26 - BMP085 Final Temperature Calculations



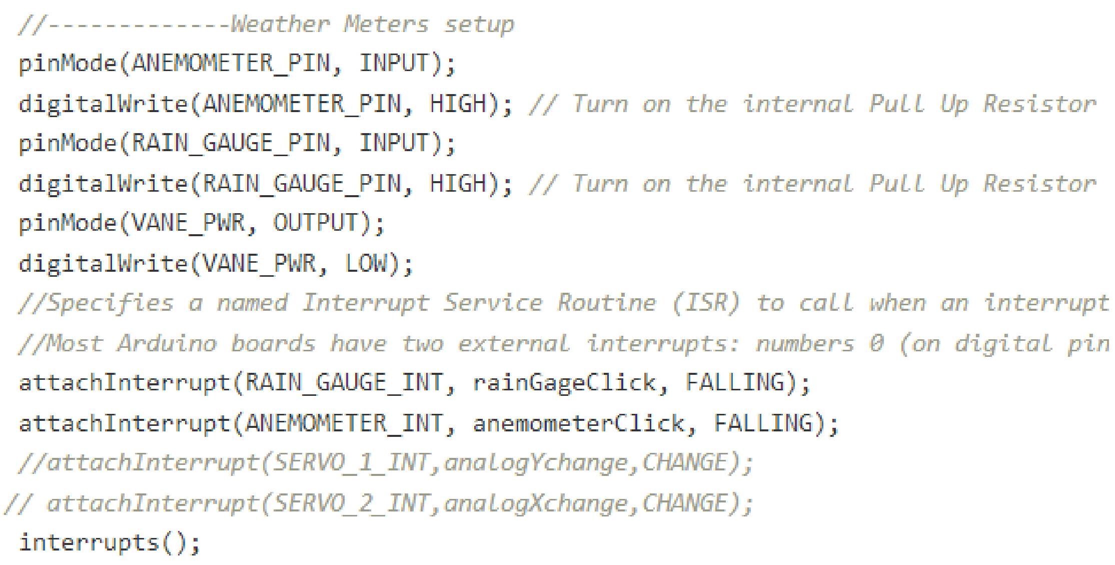
```
// Calculate pressure given up
// calibration values must be known
// b5 is also required so bmp085GetTemperature(...) must be called first.
// Value returned will be pressure in units of Pa.
long bmp085GetPressure(unsigned long up) {
  long x1, x2, x3, b3, b6, p;
  unsigned long b4, b7;

  b6 = b5 - 4000;
  // Calculate B3
  x1 = (b2 * (b6 * b6) >> 12) >> 11;
  x2 = (ac2 * b6) >> 11;
  x3 = x1 + x2;
  b3 = ((((long)ac1) * 4 + x3) << OSS) + 2) >> 2;

  // Calculate B4
  x1 = (ac3 * b6) >> 13;
  x2 = (b1 * ((b6 * b6) >> 12)) >> 16;
  x3 = ((x1 + x2) + 2) >> 2;
  b4 = (ac4 * (unsigned long)(x3 + 32768)) >> 15;

  b7 = ((unsigned long)(up - b3) * (50000 >> OSS));
  if (b7 < 0x80000000)
    p = (b7 << 1) / b4;
  else
    p = (b7 / b4) << 1;

  x1 = (p >> 8) * (p >> 8);
  x1 = (x1 * 3038) >> 16;
  x2 = (-7357 * p) >> 16;
  p += (x1 + x2 + 3791) >> 4;

  return p;
}
```

Figure 27 - BMP085 Final Pressure Calculations

Anemometer and Rain Gauge

Gathering the data from the anemometer and the rain gauge was an almost similar procedure, this is why they are analyzed together. The concept behind is the Interrupts mentioned at 4.1.2.1, 4.1.3.3, 4.1.3.4.

Firstly, pins and interrupts must be configured. Interrupts are set to FALLING mode, for when the pin goes from HIGH to LOW (Figure 29).

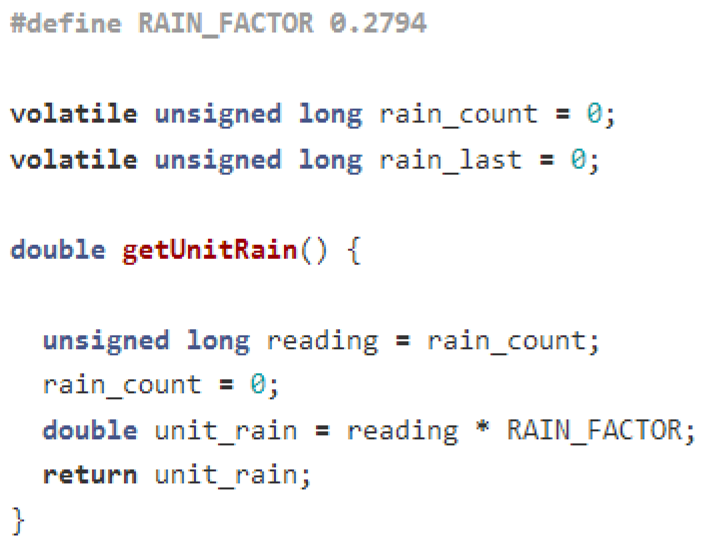


```
 //------------Weather Meters setup
 pinMode(ANEMOMETER_PIN, INPUT);
 digitalWrite(ANEMOMETER_PIN, HIGH); // Turn on the internal Pull Up Resistor
 pinMode(RAIN_GAUGE_PIN, INPUT);
 digitalWrite(RAIN_GAUGE_PIN, HIGH); // Turn on the internal Pull Up Resistor
 pinMode(VANE_PWR, OUTPUT);
 digitalWrite(VANE_PWR, LOW);
 //Specifies a named Interrupt Service Routine (ISR) to call when an interrupt
 //Most Arduino boards have two external interrupts: numbers 0 (on digital pin
 attachInterrupt(RAIN_GAUGE_INT, rainGageClick, FALLING);
 attachInterrupt(ANEMOMETER_INT, anemometerClick, FALLING);
 //attachInterrupt(SERVO_1_INT,analogYchange,CHANGE);
// attachInterrupt(SERVO_2_INT,analogXchange,CHANGE);
 interrupts();
```

Figure 28 - Weather Meters Configuration

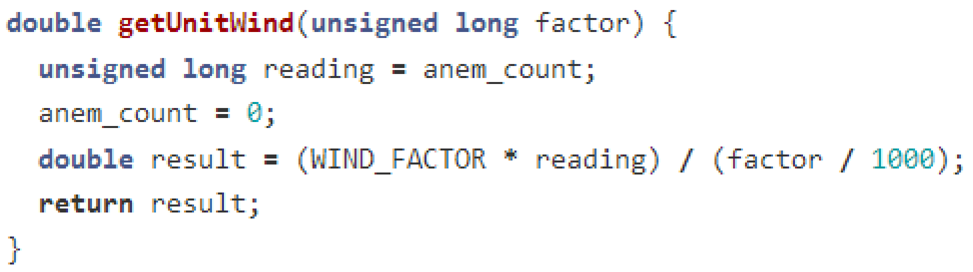
The next step is to implement rainGageClick (Figure 12) and anemometerClick (Figure 14) functions that are called whether an interrupt occurs. These functions are very simple. They just count how many times an interrupt occurred.

Finally, the counters values are used to calculate the final wind and rain depending on each sensor's factor. (i.e. For Rain each interrupt occurred corresponds to 0.2794mm of rain, instead for wind it is 2.4 miles per hour).



```
#define RAIN_FACTOR 0.2794

volatile unsigned long rain_count = 0;
volatile unsigned long rain_last = 0;

double getUnitRain() {

  unsigned long reading = rain_count;
  rain_count = 0;
  double unit_rain = reading * RAIN_FACTOR;
  return unit_rain;
}
```

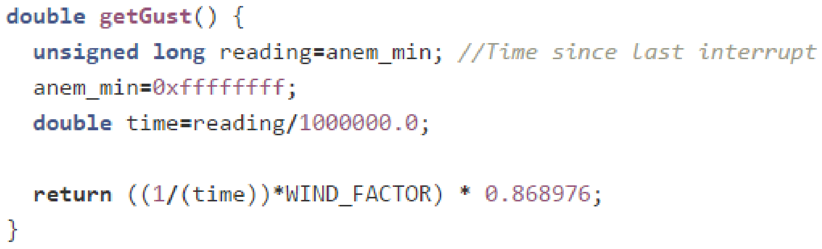
Figure 29 - Rain Final Calculation



```
double getUnitWind(unsigned long factor) {
  unsigned long reading = anem_count;
  anem_count = 0;
  double result = (WIND_FACTOR * reading) / (factor / 1000);
  return result;
}
```

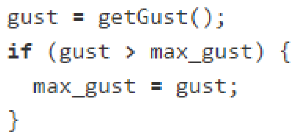
Figure 30 - Wind Final Calculation

Also, a mechanism for calculating the maximum gust has been implemented. This function runs continuously and measures the wind gust, which is a short burst of high speed wind. The idea behind the implementation is that the time between two interrupts is saved in a variable (Figure 14), it is converted from microseconds to seconds and then it is used to calculate the gust, according to the formula given in wind speed instrument's documentation [19].



```
double getGust() {
  unsigned long reading=anem_min; //Time since last interrupt
  anem_min=0xffffffff;
  double time=reading/1000000.0;

  return ((1/(time))*WIND_FACTOR) * 0.868976;
}
```

Figure 31 - Get Gust Function

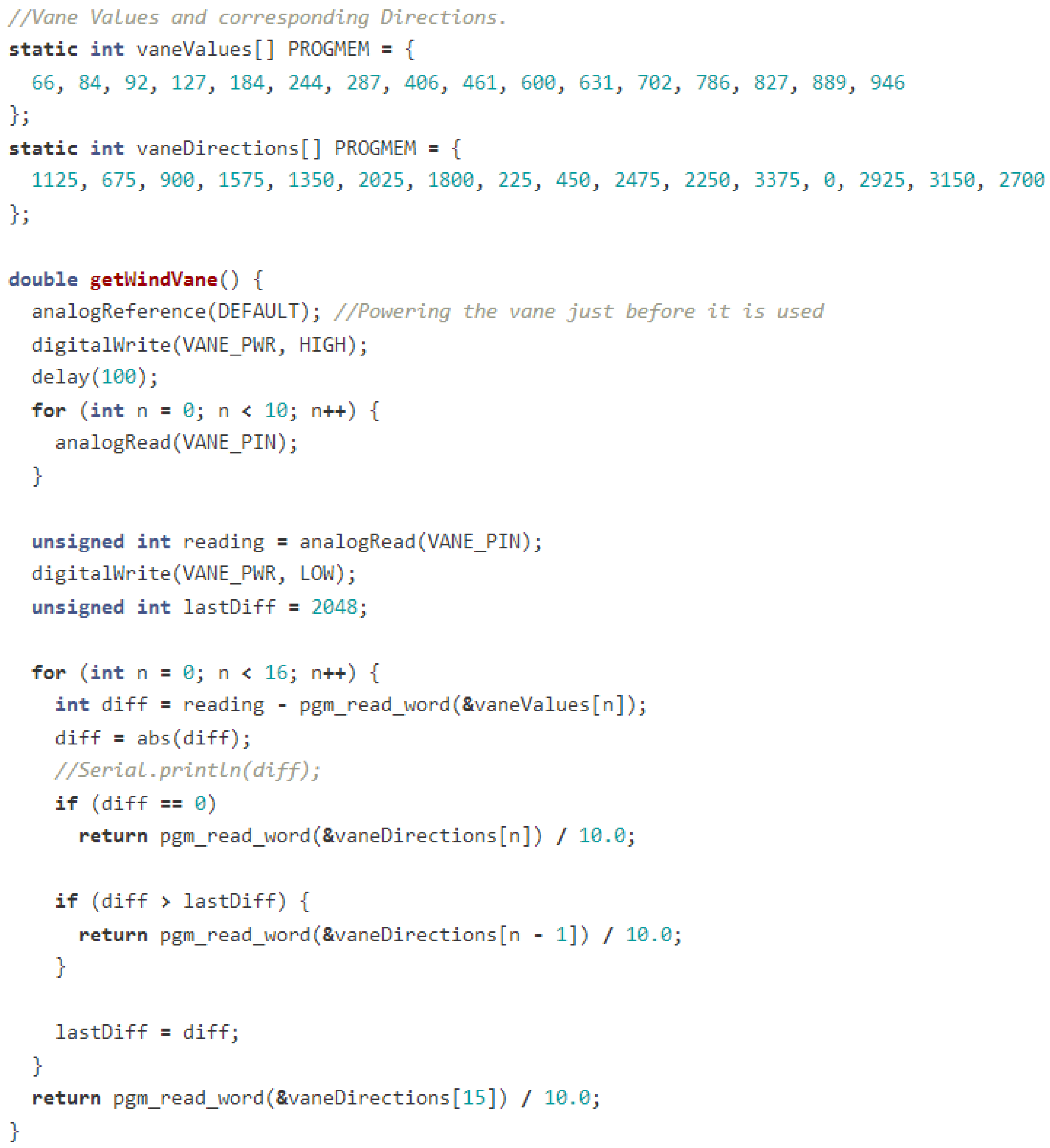


```
gust = getGust();
if (gust > max_gust) {
  max_gust = gust;
}
```

The maximum gust is stored to the database every two minutes.

Wind Direction

Wind Direction’s function compares the ADC value to the ideal values in vaneValues array and returns the corresponding wind direction in degrees.



```
//Vane Values and corresponding Directions.
static int vaneValues[] PROGMEM = {
  66, 84, 92, 127, 184, 244, 287, 406, 461, 600, 631, 702, 786, 827, 889, 946
};
static int vaneDirections[] PROGMEM = {
  1125, 675, 900, 1575, 1350, 2025, 1800, 225, 450, 2475, 2250, 3375, 0, 2925, 3150, 2700
};

double getWindVane() {
  analogReference(DEFAULT); //Powering the vane just before it is used
  digitalWrite(VANE_PWR, HIGH);
  delay(100);
  for (int n = 0; n < 10; n++) {
    analogRead(VANE_PIN);
  }

  unsigned int reading = analogRead(VANE_PIN);
  digitalWrite(VANE_PWR, LOW);
  unsigned int lastDiff = 2048;

  for (int n = 0; n < 16; n++) {
    int diff = reading - pgm_read_word(&vaneValues[n]);
    diff = abs(diff);
    //Serial.println(diff);
    if (diff == 0)
      return pgm_read_word(&vaneDirections[n]) / 10.0;

    if (diff > lastDiff) {
      return pgm_read_word(&vaneDirections[n - 1]) / 10.0;
    }

    lastDiff = diff;
  }
  return pgm_read_word(&vaneDirections[15]) / 10.0;
}
```

To save SRAM memory, PROGMEM [25] is used to store data in flash instead of SRAM. For the purpose of the project it is not really useful since memory is enough, but it is a programming practice worth mentioning, especially when referring to the domain of MCUs that memory is valuable.

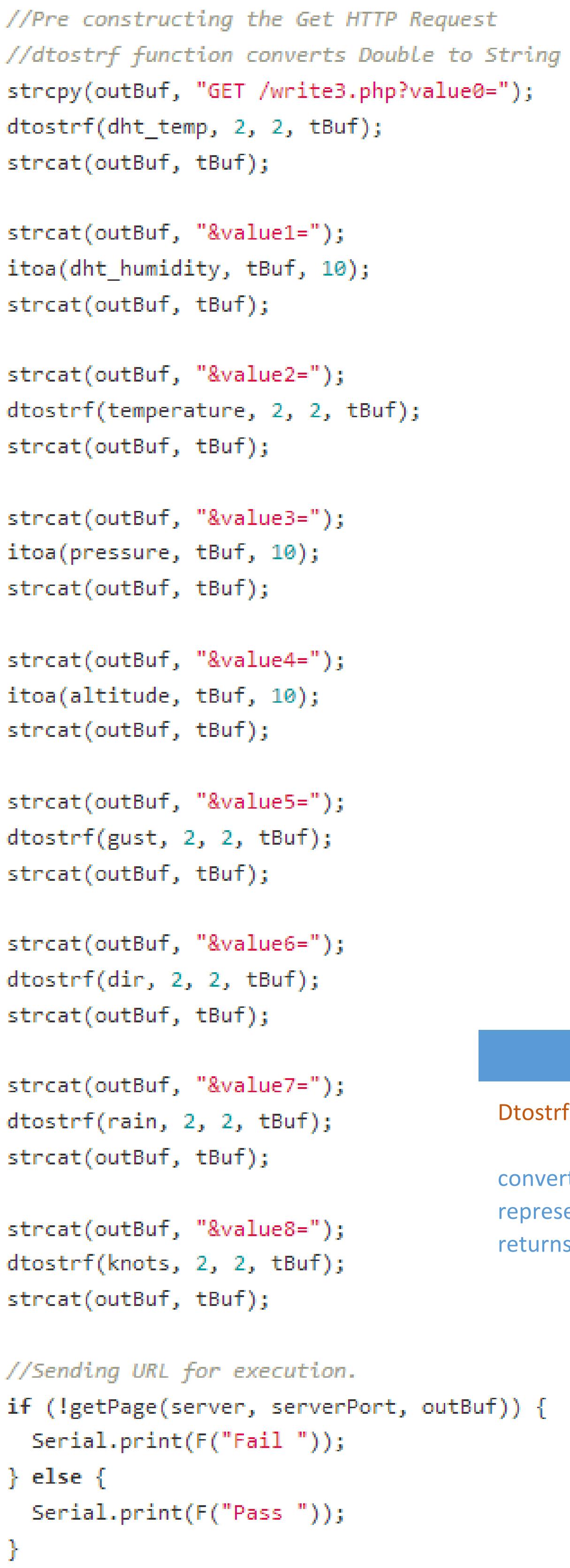
The pgm_read_word function [24] pulls the values out of FLASH and into SRAM when needed but only works with integers, so the actual wind directions are stored as 10x their value.

4.2.1.2. HTTP Request

The HTTP Request sends the sensors data to the MYSQL database. The request method used is GET [26]. GET can be used to send small amounts of data to the server through URL Encoding.

The construction of the URL (Figure 33) is being done before the HTTP request process, so Arduino can send the data in one packet using the write function of Ethernet library.

When the URL assembly process is done, it is passed to getPage( ) (Figure 34) function.



```
//Pre constructing the Get HTTP Request
//dtostrf function converts Double to String
strcpy(outBuf, "GET /write3.php?value0=");
dtostrf(dht_temp, 2, 2, tBuf);
strcat(outBuf, tBuf);

strcat(outBuf, "&value1=");
itoa(dht_humidity, tBuf, 10);
strcat(outBuf, tBuf);

strcat(outBuf, "&value2=");
dtostrf(temperature, 2, 2, tBuf);
strcat(outBuf, tBuf);

strcat(outBuf, "&value3=");
itoa(pressure, tBuf, 10);
strcat(outBuf, tBuf);

strcat(outBuf, "&value4=");
itoa(altitude, tBuf, 10);
strcat(outBuf, tBuf);

strcat(outBuf, "&value5=");
dtostrf(gust, 2, 2, tBuf);
strcat(outBuf, tBuf);

strcat(outBuf, "&value6=");
dtostrf(dir, 2, 2, tBuf);
strcat(outBuf, tBuf);

strcat(outBuf, "&value7=");
dtostrf(rain, 2, 2, tBuf);
strcat(outBuf, tBuf);

strcat(outBuf, "&value8=");
dtostrf(knots, 2, 2, tBuf);
strcat(outBuf, tBuf);

//Sending URL for execution.
if (!getPage(server, serverPort, outBuf)) {
  Serial.print(F("Fail "));
} else {
  Serial.print(F("Pass "));
}
```

Figure 32 - URL contruction

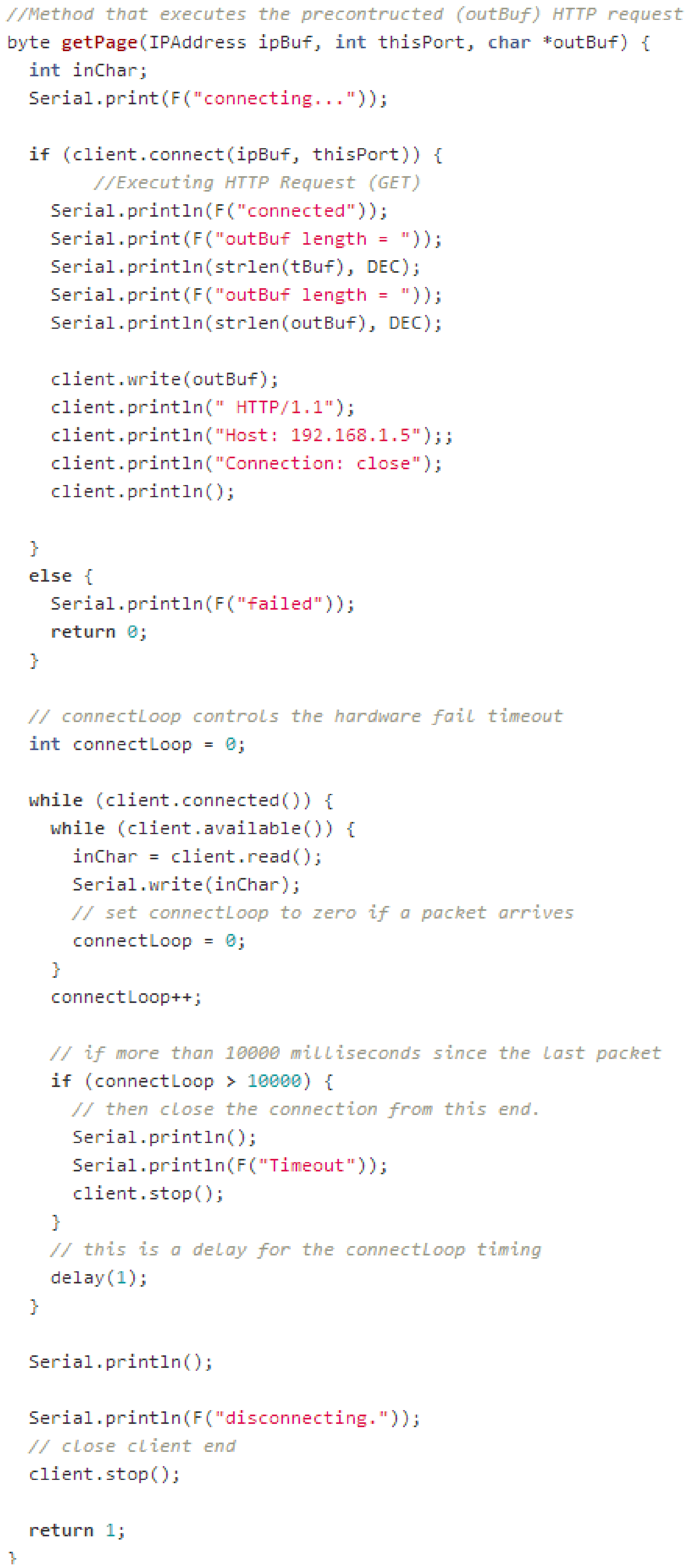
This function connects to the server and executes the HTTP Request with the data included within the URI.

The getPage function also has a connection status control. If the connection fails or the server stalls, the Arduino code shows a “Timeout” message after ten seconds and continue to attempt the connection every a time threshold we set.

Dtostrf function explained

Dtostrf (double val, signed char width, unsigned char prec, char* s)

converts the double values passes into an ASCII representation that will be stored under s. The function returns the pointer to the converter string s.

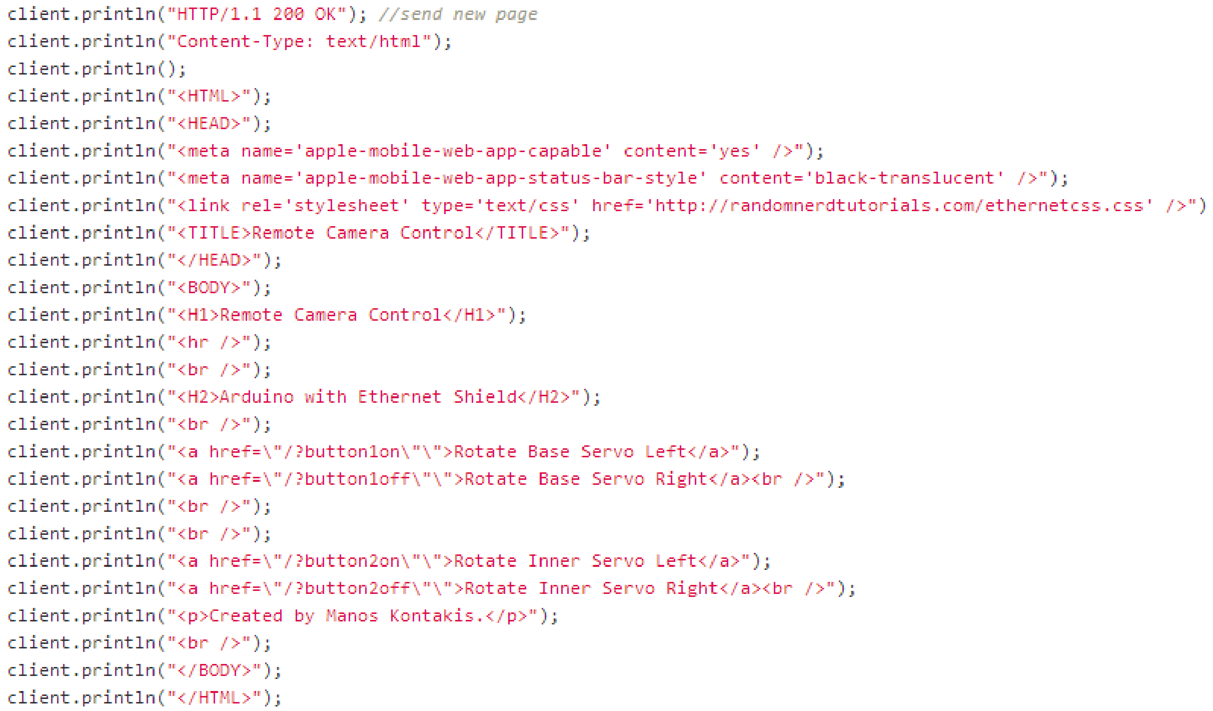


```
//Method that executes the precontructed (outBuf) HTTP request
byte getPage(IPAddress ipBuf, int thisPort, char *outBuf) {
  int inChar;
  Serial.print(F("connecting..."));

  if (client.connect(ipBuf, thisPort)) {
        //Executing HTTP Request (GET)
    Serial.println(F("connected"));
    Serial.print(F("outBuf length = "));
    Serial.println(strlen(tBuf), DEC);
    Serial.print(F("outBuf length = "));
    Serial.println(strlen(outBuf), DEC);

    client.write(outBuf);
    client.println(" HTTP/1.1");
    client.println("Host: 192.168.1.5");;
    client.println("Connection: close");
    client.println();

  }
  else {
    Serial.println(F("failed"));
    return 0;
  }

  // connectLoop controls the hardware fail timeout
  int connectLoop = 0;

  while (client.connected()) {
    while (client.available()) {
      inChar = client.read();
      Serial.write(inChar);
      // set connectLoop to zero if a packet arrives
      connectLoop = 0;
    }
    connectLoop++;

    // if more than 10000 milliseconds since the last packet
    if (connectLoop > 10000) {
      // then close the connection from this end.
      Serial.println();
      Serial.println(F("Timeout"));
      client.stop();
    }
    // this is a delay for the connectLoop timing
    delay(1);
  }

  Serial.println();

  Serial.println(F("disconnecting."));
  // close client end
  client.stop();

  return 1;
}
```

Figure 33 - HTTP Request Execution

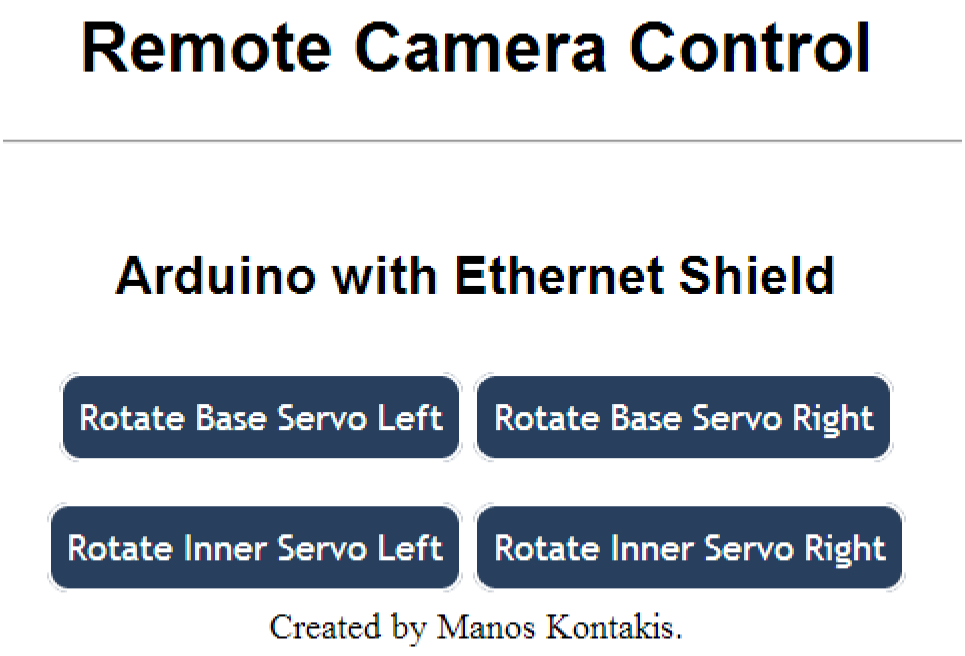
4.2.1.3. Pan Tilt Camera Mount Control

The Remote Camera Control part of the Arduino sketch, controls the Pan Tilt [27] Bracket remotely. Arduino and Ethernet shield serves up a web page that allows the control of the servos that rotate the bracket. When a client connects to the Arduino server, it sends back to the client's browser the HTML page (Figure 35, Figure 36).



```
client.println("HTTP/1.1 200 OK"); //send new page
client.println("Content-Type: text/html");
client.println();
client.println("<HTML>");
client.println("<HEAD>");
client.println("<meta name='apple-mobile-web-app-capable' content='yes' />");
client.println("<meta name='apple-mobile-web-app-status-bar-style' content='black-translucent' />");
client.println("<link rel='stylesheet' type='text/css' href='http://randomnerdtutorials.com/ethernetcss.css' />")
client.println("<TITLE>Remote Camera Control</TITLE>");
client.println("</HEAD>");
client.println("<BODY>");
client.println("<H1>Remote Camera Control</H1>");
client.println("<hr />");
client.println("<br />");
client.println("<H2>Arduino with Ethernet Shield</H2>");
client.println("<br />");
client.println("<a href=\"/?button1on\"\">Rotate Base Servo Left</a>");
client.println("<a href=\"/?button1off\"\">Rotate Base Servo Right</a><br />");
client.println("<br />");
client.println("<br />");
client.println("<a href=\"/?button2on\"\">Rotate Inner Servo Left</a>");
client.println("<a href=\"/?button2off\"\">Rotate Inner Servo Right</a><br />");
client.println("<p>Created by Manos Kontakis.</p>");
client.println("<br />");
client.println("</BODY>");
client.println("</HTML>");
```

Figure 34 - HTTP Response



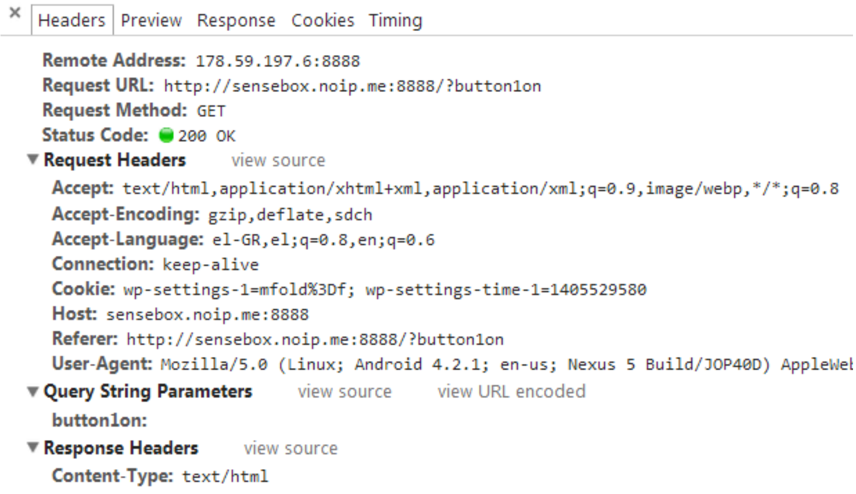
Remote Camera Control

Arduino with Ethernet Shield

Created by Manos Kontakis.

Figure 35 - Remote Camera Control

When the user clicks on a button, the browser generates a HTTP Request with the button name attached to the URL. Arduino reads the HTTP Request and checks which button triggered the request.

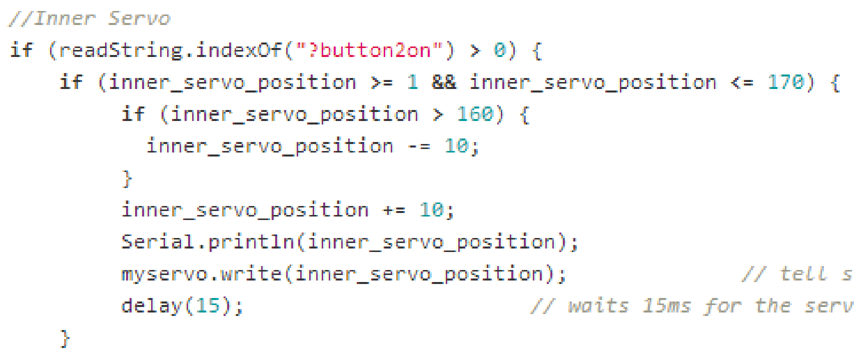


```
Remote Address: 178.59.197.6:8888
Request URL: http://sensebox.noip.me:8888/?button1on
Request Method: GET
Status Code: ● 200 OK
▼ Request Headers
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
  Accept-Encoding: gzip,deflate,sdch
  Accept-Language: el-GR,el;q=0.8,en;q=0.6
  Connection: keep-alive
  Cookie: wp-settings-1=mfold%3Df; wp-settings-time-1=1405529580
  Host: sensebox.noip.me:8888
  Referer: http://sensebox.noip.me:8888/?button1on
  User-Agent: Mozilla/5.0 (Linux; Android 4.2.1; en-us; Nexus 5 Build/JOP40D) AppleWet
▼ Query String Parameters
  button1on:
▼ Response Headers
  Content-Type: text/html
```

Figure 36 - Generated HTTP Request on button click

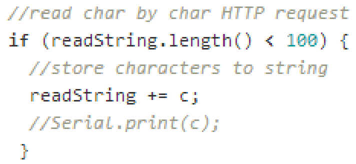
indexOf locates a character or String within another String. Returns the index of the desired String within the String or -1 if not found.

The rest of the code rotates the servo.



```
//Inner Servo
if (readString.indexOf("?button2on") > 0) {
    if (inner_servo_position >= 1 && inner_servo_position <= 170) {
         if (inner_servo_position > 160) {
           inner_servo_position -= 10;
         }
         inner_servo_position += 10;
         Serial.println(inner_servo_position);
         myservo.write(inner_servo_position);              // tell s
         delay(15);                       // waits 15ms for the serv
    }
```

Figure 37 - Servo Control Example



```
//read char by char HTTP request
if (readString.length() < 100) {
  //store characters to string
  readString += c;
  //Serial.print(c);
 }
```

Figure 38

4.2.1.4. Watchdog Timer

A watchdog timer (WDT; sometimes called a computer operating properly or COP timer, or simply a watchdog) is an electronic timer that is used to detect and recover from computer malfunctions. During normal operation, the computer regularly restarts the watchdog timer to prevent it from elapsing, or "timing out". If, due to a hardware fault or program error, the computer fails to restart the watchdog, the timer will elapse and generate a timeout signal. The timeout signal is used to initiate corrective action or actions. The corrective actions typically include placing the computer system in a safe state and restoring normal system operation.

The ATmega328P that powers the Arduino Mega provides a programmable Watchdog Timer with a separator on-chip 128kHz Oscillator. The WDT gives an interrupt and / or a system reset when the counter reaches a given time-out value. In normal operation mode, it is required that the system uses the WDR - Watchdog Timer Reset - instruction to restart the counter before the time-out value is reached. If the system Does not restart the counter, an interrupt or system reset will be issued.

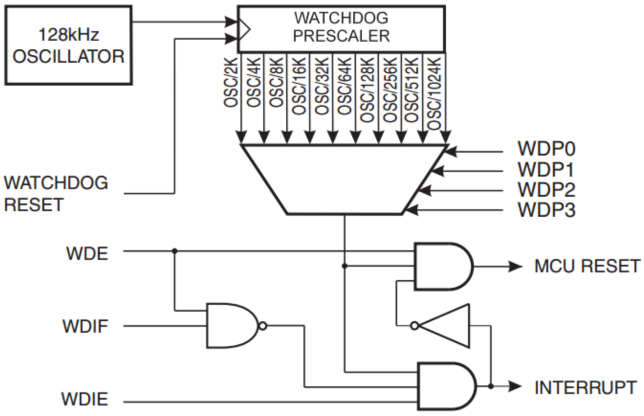


Figure 39 - Watchdog Timer Schematic

In Interrupt mode (WDIE), the WDT triggers an interrupt when the timer expires. This interrupt can be used to wake the device from sleep-modes, and also as a general system timer. **In System Reset mode**, the WDT gives a reset when the timer expires. This is typically used to prevent system hang-up in case of runaway code. **The third mode**, Interrupt and System Reset mode, combines the other two modes by first giving an interrupt and then switch to System Reset mode. This mode will for instance allow a safe shutdown by saving critical parameters before a system reset. This is the mode used in the project.

To configuration of the WDT is done through one of the registers known as WDTCSR. Every feature will have a register allocated to it. The microcontroller knows where to go in its memory to get these registers. The watchdog is no different. The watchdog register, composed of one byte, thus has 8 bits in on/off settings. A '1' indicates 'ON' and a '0' indicates 'OFF'. By default everything is set to 'OFF' so we only need to set the relevant bits to 'ON'.

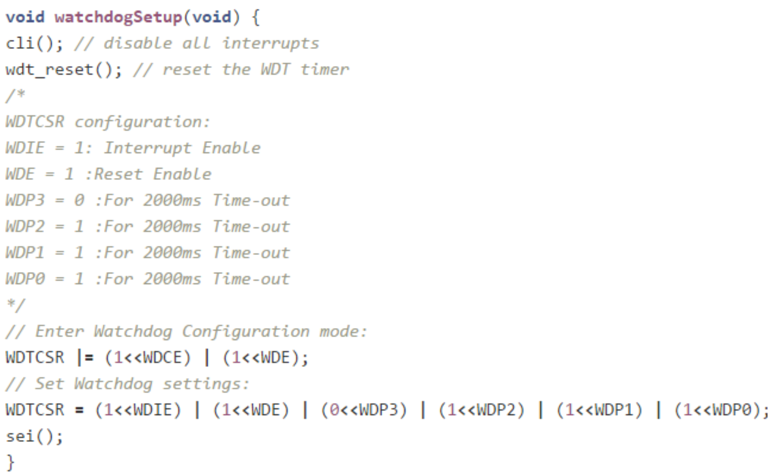
Bit	Name
7	WDIF
6	WDIE
5	WDP3
4	WDCE
3	WDE
2	WDP2
1	WDP1
0	WDP0

Table 8 - Watchdog Timer Register Bits

The bits of the WDTCSR register are explained in detail below.

- **Bit 7 – WDIF: Watchdog Interupt Flag**
  This bit is set when a time-out occurs in the Watchdog Timer and the Watchdog Timer is configured for interrupt. It is automatically flagged high and low by the system.
- **Bit 6 – WDIE: Watchdog Interrupt Enable**
  When this bit is written to one and the I-bit in the Status Register is set, the Watchdog Interrupt is enabled. If WDE is cleared in combination with this setting, the Watchdog Timer is in Interrupt Mode, and the corresponding interrupt is executed if time-out in the Watchdog Timer occurs. If WDE is set, the Watchdog Timer is in Interrupt and System Reset Mode. The first time-out in the Watchdog Timer will set WDIF. Executing the corresponding interrupt vector will clear WDIE and WDIF automatically by hardware (the Watchdog goes to System Reset Mode). This is useful for keeping the Watchdog Timer security while using the interrupt. To stay in Interrupt and System Reset Mode, WDIE must be set after each interrupt. This should however not be done within the interrupt service routine itself, as this might compromise the safety-function of the Watchdog System Reset mode. If the interrupt is not executed before the next time-out, a System Reset will be applied.
- **Bit 4 – WDCE: Watchdog Change Enable**
  This bit is used in timed sequences for changing WDE and prescaler bits. To clear the WDE bit, and/or change the prescaler bits, WDCE must be set. Once written to one, hardware will clear WDCE after four clock cycles.
- **Bit 3 – WDE: Watchdog System Reset Enable**
  WDE is overridden by WDRF in MCUSR. This means that WDE is always set when WDRF is set. To clear WDE, WDRF must be cleared first. This feature ensures multiple resets during conditions causing failure, and a safe start-up after the failure.
- **Bit 5, 2:0 - WDP[3:0]: Watchdog Timer Prescaler 3, 2, 1 and 0**
  The WDP[3:0] bits determine the Watchdog Timer prescaling when the Watchdog Timer is running.
  Source: [35]

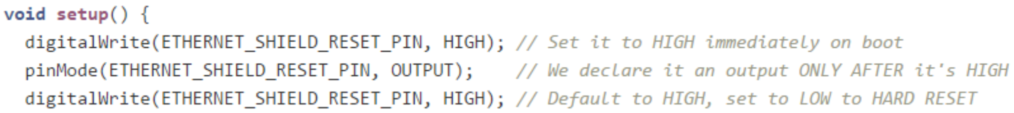
In the arduino sketch the WDT is implemented in a function called during the set up proccess. This function sets up the WDT to include interrupts, a reset and a time – out after 2 seconds.



```
void watchdogSetup(void) {
cli(); // disable all interrupts
wdt_reset(); // reset the WDT timer
/*
WDTCSR configuration:
WDIE = 1: Interrupt Enable
WDE = 1 :Reset Enable
WDP3 = 0 :For 2000ms Time-out
WDP2 = 1 :For 2000ms Time-out
WDP1 = 1 :For 2000ms Time-out
WDP0 = 1 :For 2000ms Time-out
*/
// Enter Watchdog Configuration mode:
WDTCSR |= (1<<WDCE) | (1<<WDE);
// Set Watchdog settings:
WDTCSR = (1<<WDIE) | (1<<WDE) | (0<<WDP3) | (1<<WDP2) | (1<<WDP1) | (1<<WDP0);
sei();
}
```

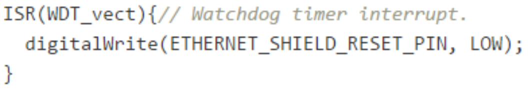
Figure 40 - Watchdog Timer Setup Function

The need to include an interrupt before Arduino resets comes from a bug that sometimes the board resets and the Ethernet Shield doesn't. To overcome this drawback a small hack has been applied to the Arduino by cross-wiring a hardware reset input to a digital output on the board itself. The ETHERNET_SHIELD_RESET_PIN is set to HIGH immidiatelty on boot and then we declare it to avoid any unexpected behavior. When the reset pin is set to LOW the Ethernet Shield hard resets. This happens on the interrupt that the WDT triggers just before the soft reset Figure 43 - Watchdog Timer Interrupt Routine.



```
void setup() {
  digitalWrite(ETHERNET_SHIELD_RESET_PIN, HIGH); // Set it to HIGH immediately on boot
  pinMode(ETHERNET_SHIELD_RESET_PIN, OUTPUT);    // We declare it an output ONLY AFTER it's HIGH
  digitalWrite(ETHERNET_SHIELD_RESET_PIN, HIGH); // Default to HIGH, set to LOW to HARD RESET
```

Figure 41 - Watchdog Timer Preliminaries



```
ISR(WDT_vect){// Watchdog timer interrupt.
  digitalWrite(ETHERNET_SHIELD_RESET_PIN, LOW);
}
```

Figure 42 - Watchdog Timer Interrupt Routine

4.2.2. Android Application [28]

The Android Application processes the weather data and presents them in different formats:

- Charts
- Current Weather Conditions Report
- Min / Max Report
- Data presentation in Table Layout

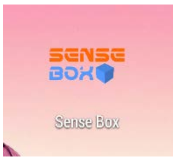


Figure 43 - Android App Icon

Figure 45 - Splash Screen

The Charts are generated using the GraphView [28] library. GraphView is a library for Android that programmatically creates flexible and elegant diagrams. The reason why I picked this library is because it is easy to understand, to integrate and to customize. The library supports two different types of diagrams. Line and Bar Charts.

As soon as the app is launched, the Splash Screen appears. During the Splash Screen, the weather data is fetched, in JSON format, and parsed to generate the 24 Hours Charts (First Screen). If the device is connected to the internet with a decent connection, the Splash Screen appears for around 5 seconds. The slowest the connection is the longest the Splash Screen lasts, because the device has to connect to the server and fetch the Weather Data. This is the most time – consuming part. Parsing the fetched data and drawing the Charts takes only some milliseconds.

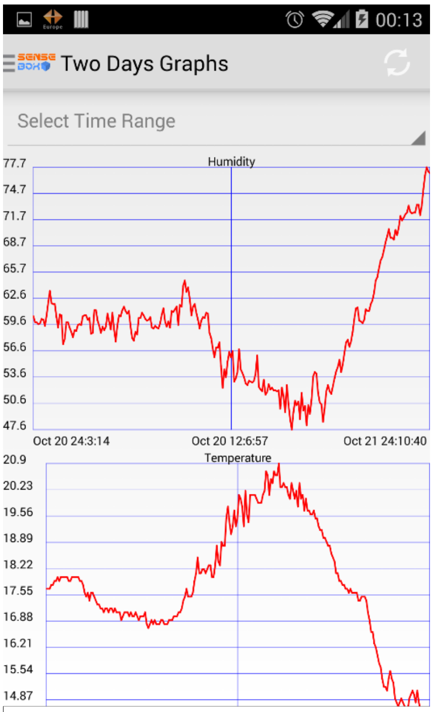


Figure 44 - Start Screen

Figure 45 - Start Screen shows the start screen of the application. Seven charts (only two are shown in the figure) are generated. Swiping from the left edge of the screen or by touching the application icon on the action bar will display the menu, implemented using Navigation Drawer [29] element.

The navigation drawer [**Error! Reference source not found.**] is a panel that transitions in from the left edge of the screen and displays the app's main navigation options. It is a fancy element for navigating between the views of the application, although it is not a general replacement for top – level navigation via spinners (more on Spinners later). As Google suggests in its Developer Documents the navigation drawer is recommended when there are more than 3 top – level views. In the application there are four so its use is officially recommended.

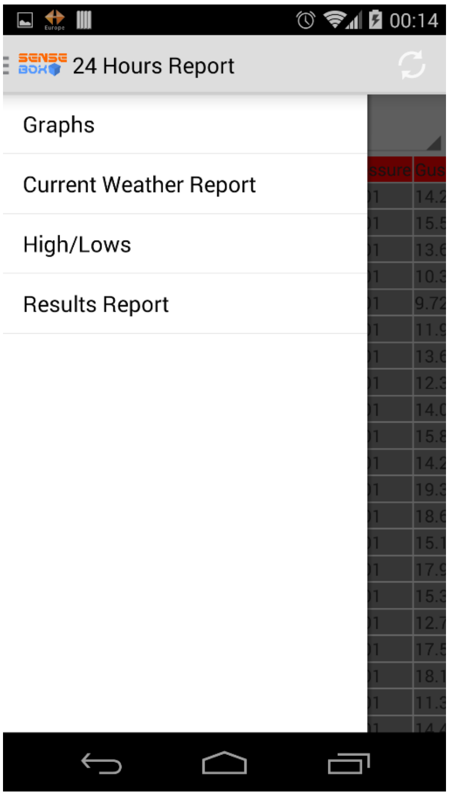
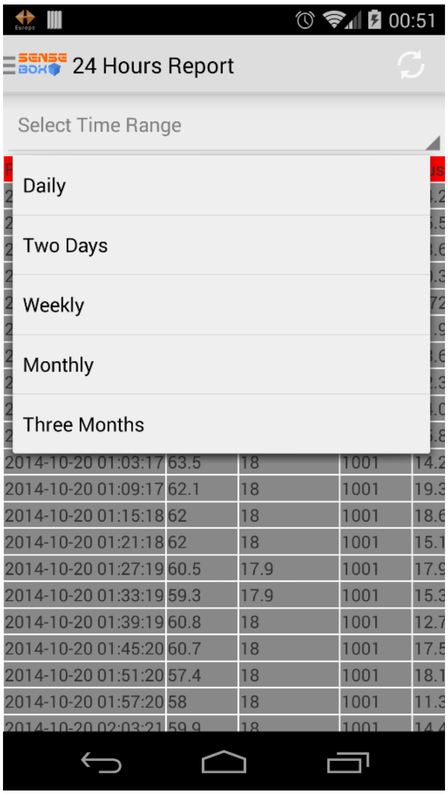


Figure 47 - Navigation Drawer

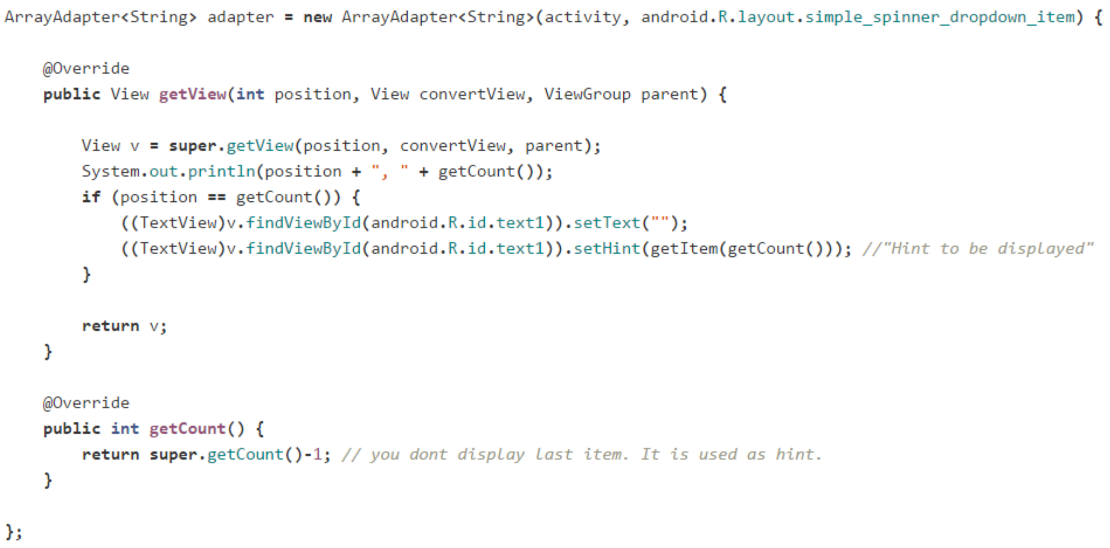


2014-10-20 01:03:17	63.5	18	1001	14.2
2014-10-20 01:09:17	62.1	18	1001	19.3
2014-10-20 01:15:18	62	18	1001	18.6
2014-10-20 01:21:18	62	18	1001	15.1
2014-10-20 01:27:19	60.5	17.9	1001	17.9
2014-10-20 01:33:19	59.3	17.9	1001	15.3
2014-10-20 01:39:19	60.8	18	1001	12.7
2014-10-20 01:45:20	60.7	18	1001	17.5
2014-10-20 01:51:20	57.4	18	1001	18.1
2014-10-20 01:57:20	58	18	1001	11.3
2014-10-20 02:03:21	59.9	18	1001	14.4

Figure 46 - Spinner

The navigation drawer navigates the user into the four main parts of the application: Graphs, Current Weather Report, High/Lows, Results Report. The default time range for the views is Daily (since midnight), expect for CWR that time range is fixed. As soon as the view has been loaded, the user can select the preferred time range for the retrieval of the weather data through the Spinner [40] under the Action Bar.

Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. However, it is better to provide a hint to the user for what to expect by clicking the spinner, so a small modification has been done to the Spinner to implement it.



```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(activity, android.R.layout.simple_spinner_dropdown_item) {

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        View v = super.getView(position, convertView, parent);
        System.out.println(position + ", " + getCount());
        if (position == getCount()) {
            ((TextView)v.findViewById(android.R.id.text1)).setText("");
            ((TextView)v.findViewById(android.R.id.text1)).setHint(getItem(getCount())); //"Hint to be displayed"
        }

        return v;
    }

    @Override
    public int getCount() {
        return super.getCount()-1; // you dont display last item. It is used as hint.
    }

};
```

Figure 48 - Spinner 'Hint' Modification Code

Charts Creation Procedure

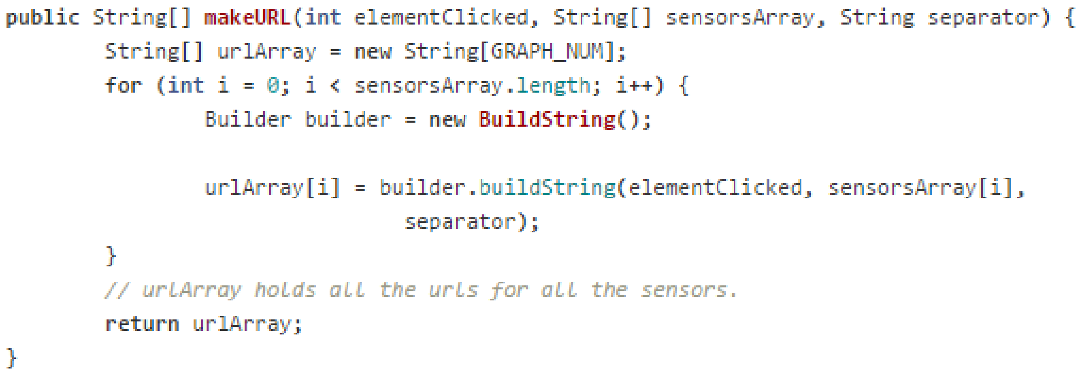
To generate the Charts the following steps are needed:

- Construct the URLS
- Execute HTTP Requests and fetch Weather Data from Database in JSON format
- Parse JSON Objects
- Draw the Charts

URLs Construction

The makeURL method constructs the URLs using the Builder Class (BuildString) and saves them in a String Array. One URL corresponds to one chart.

A URL built with that function can be found in Figure 73 - URL example



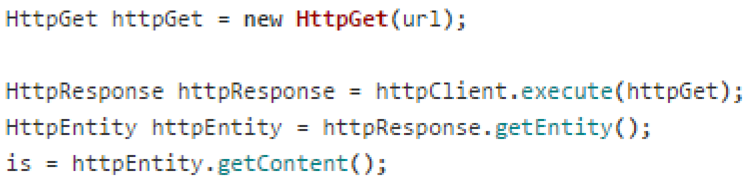
```
public String[] makeURL(int elementClicked, String[] sensorsArray, String separator) {
        String[] urlArray = new String[GRAPH_NUM];
        for (int i = 0; i < sensorsArray.length; i++) {
                Builder builder = new BuildString();

                urlArray[i] = builder.buildString(elementClicked, sensorsArray[i],
                                separator);
        }
        // urlArray holds all the urls for all the sensors.
        return urlArray;
}
```

Figure 49 - *Method to construct URLs*

As soon as this procedure is done, the application is ready to proceed to the next step and execute the HTTP requests.

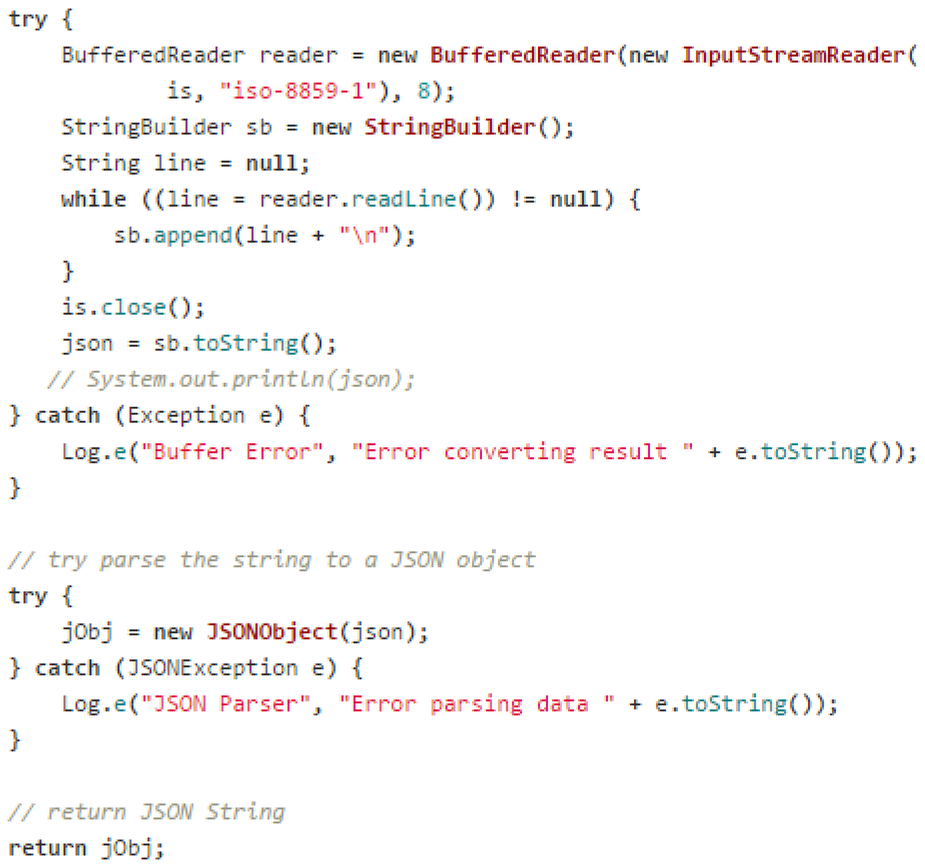
Execute HTTP Requests

The URLs one by one are executed and the content of the response is handled in **Figure 52**. There the BufferedReader reads the response and appends the characters of the response one by one in a StringBuilder object and finally when it is done it converts it to a **JSONobject**.



```
HttpGet httpGet = new HttpGet(url);

HttpResponse httpResponse = httpClient.execute(httpGet);
HttpEntity httpEntity = httpResponse.getEntity();
is = httpEntity.getContent();
```

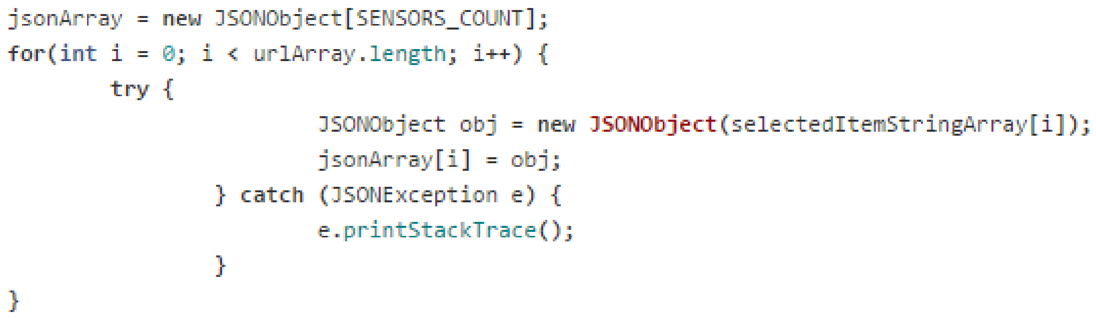
Figure 50 - Android HTTP Request



```
try {
    BufferedReader reader = new BufferedReader(new InputStreamReader(
            is, "iso-8859-1"), 8);
    StringBuilder sb = new StringBuilder();
    String line = null;
    while ((line = reader.readLine()) != null) {
        sb.append(line + "\n");
    }
    is.close();
    json = sb.toString();
   // System.out.println(json);
} catch (Exception e) {
    Log.e("Buffer Error", "Error converting result " + e.toString());
}

// try parse the string to a JSON object
try {
    jObj = new JSONObject(json);
} catch (JSONException e) {
    Log.e("JSON Parser", "Error parsing data " + e.toString());
}

// return JSON String
return jObj;
```

Figure 51 - HTTP Response Handling

At Figure 53, a JSONObject array is being filled with JSON data. Each index holds data for each sensor.



```
jsonArray = new JSONObject[SENSORS_COUNT];
for(int i = 0; i < urlArray.length; i++) {
        try {
                        JSONObject obj = new JSONObject(selectedItemStringArray[i]);
                        jsonArray[i] = obj;
                } catch (JSONException e) {
                        e.printStackTrace();
                }
}
```

So at this point of time, that all the data from all the sensors are stored into the jsonArray (JSONObject[]) we can proceed on parsing all these data and finally generate the charts.

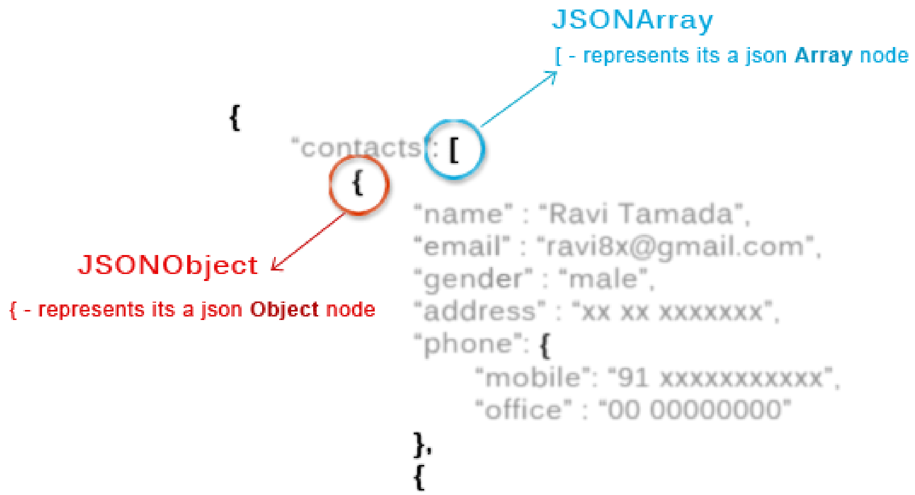
Figure 52 - Filling JSONobject Array, one index per sensor

Parse / Resolve JSON Objects

JSON or **JavaScript Object Notation**, is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML.

Parsing the JSON Objects and extracting the weather information from them is a little complex procedure. In the cases of monthly and quarter charts/data presentation these functions are intense for the device.

In general all the JSON nodes will start with a square bracket or with a curly bracket. The difference between [and { is, the square bracket ([) represents starting of an **JSONArray** node whereas curly bracket ({) represents **JSONObject**. So while accessing these nodes we need to call appropriate method to access the data.

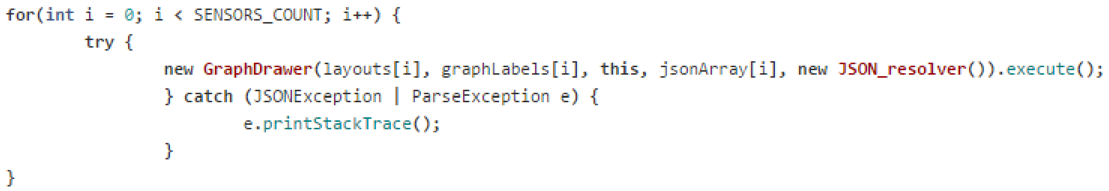


```
{
    "contacts": [
        {
            "name" : "Ravi Tamada",
            "email" : "ravi8x@gmail.com",
            "gender" : "male",
            "address" : "xx xx xxxxxxx",
            "phone": {
                "mobile": "91 xxxxxxxxxx",
                "office" : "00 00000000"
            },
        {
```

JSONArray
[- represents its a json Array node
JSONObject
{ - represents its a json Object node

Figure 53 - Example of JSON Elements

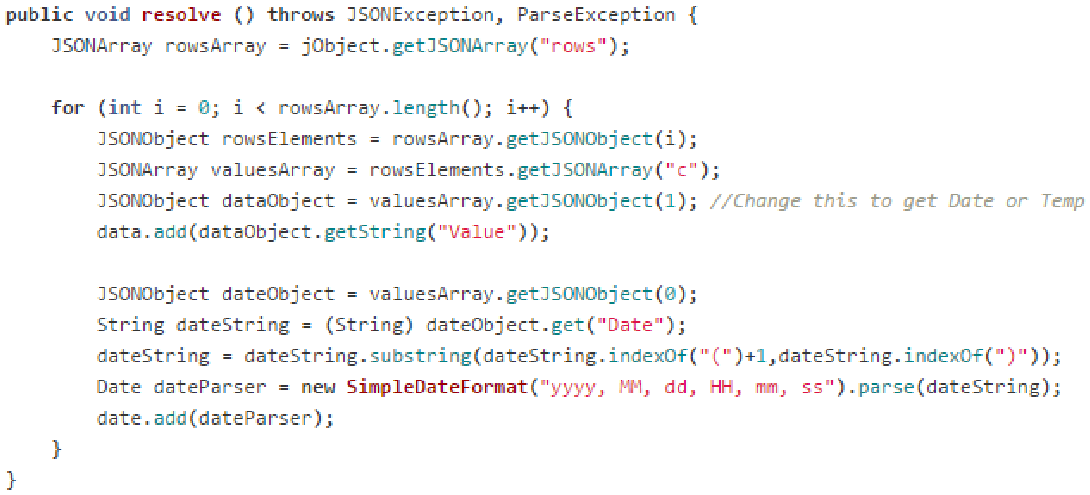
At the for loop below, a new instance for each sensor of the class that builds and draws the charts is created. One of the parameters that this method gets as input is one index, per loop, of the jsonArray and also a new JSON_resolver object in order to fulfil the parsing of the object. More details on GraphDrawer class can be found in the next section.



```
for(int i = 0; i < SENSORS_COUNT; i++) {
        try {
                new GraphDrawer(layouts[i], graphLabels[i], this, jsonArray[i], new JSON_resolver()).execute();
                } catch (JSONException | ParseException e) {
                        e.printStackTrace();
                }
}
```

The resolve method in Figure 55 extracts the data from the JSONObject passed as parameter in the constructor of GraphDrawer class. The resolve procedure is being done according to the Figure 54.

The method stores the Dates and the Data in two separate Array Lists data structures so they can be easily used to generate the charts.



```
public void resolve () throws JSONException, ParseException {
    JSONArray rowsArray = jObject.getJSONArray("rows");

    for (int i = 0; i < rowsArray.length(); i++) {
        JSONObject rowsElements = rowsArray.getJSONObject(i);
        JSONArray valuesArray = rowsElements.getJSONArray("c");
        JSONObject dataObject = valuesArray.getJSONObject(1); //Change this to get Date or Temp
        data.add(dataObject.getString("Value"));

        JSONObject dateObject = valuesArray.getJSONObject(0);
        String dateString = (String) dateObject.get("Date");
        dateString = dateString.substring(dateString.indexOf("(")+1,dateString.indexOf(")"));
        Date dateParser = new SimpleDateFormat("yyyy, MM, dd, HH, mm, ss").parse(dateString);
        date.add(dateParser);
    }
}
```

Figure 54 - JSON resolver method

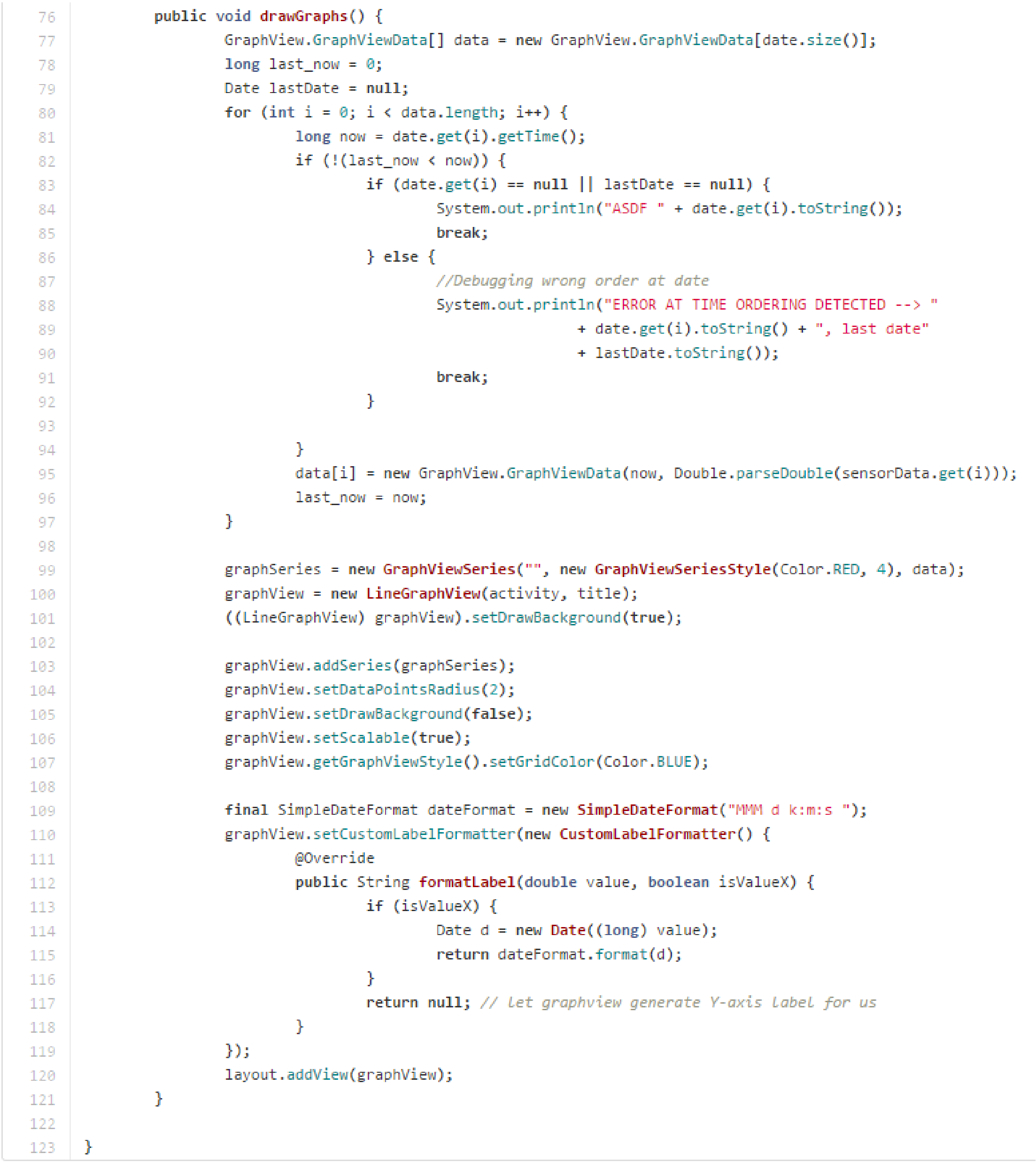
Drawing the Charts

As soon as the Data and the Dates is extracted from each JSONObject the charts are very easy to generate. The procedure is a straightforward one thanks to the intuitive library Android Graph – View [29].

GraphDrawer class is responsible to create and draw the charts to UI. It extends AsyncTask so each chart creation process is executed in a separate thread. AsyncTask fits just perfect in our situation since it performs HTTP Requests and JSON parsing to background and publish the results on the UI thread as soon as each chart is ready.

First of all, a GraphViewData Array is created, to keep all the samples for the charts. Its constructor takes two parameters, the X value (Date) and the Y value (Sensor Value) as shown in line 95. To add dates to X axis they must be converted in Unix Time format. This is what getTime() method does in line 81. This method returns how many milliseconds have passed since January 1, 1970, 00:00:00 GMT.

Then, a GraphViewSeries class is instantiated that holds the data, description and styles. The code at line 103 adds the series of data to the chart.

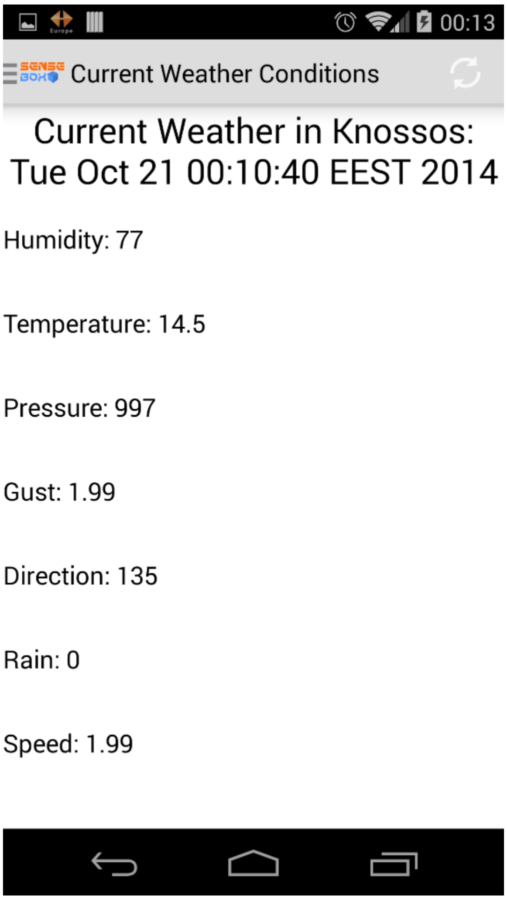


```
        public void drawGraphs() {
                GraphView.GraphViewData[] data = new GraphView.GraphViewData[date.size()];
                long last_now = 0;
                Date lastDate = null;
                for (int i = 0; i < data.length; i++) {
                        long now = date.get(i).getTime();
                        if (!(last_now < now)) {
                                if (date.get(i) == null || lastDate == null) {
                                        System.out.println("ASDF " + date.get(i).toString());
                                        break;
                                } else {
                                        //Debugging wrong order at date
                                        System.out.println("ERROR AT TIME ORDERING DETECTED --> "
                                                        + date.get(i).toString() + ", last date"
                                                        + lastDate.toString());
                                        break;
                                }

                        }
                        data[i] = new GraphView.GraphViewData(now, Double.parseDouble(sensorData.get(i)));
                        last_now = now;
                }

                graphSeries = new GraphViewSeries("", new GraphViewSeriesStyle(Color.RED, 4), data);
                graphView = new LineGraphView(activity, title);
                ((LineGraphView) graphView).setDrawBackground(true);

                graphView.addSeries(graphSeries);
                graphView.setDataPointsRadius(2);
                graphView.setDrawBackground(false);
                graphView.setScalable(true);
                graphView.getGraphViewStyle().setGridColor(Color.BLUE);

                final SimpleDateFormat dateFormat = new SimpleDateFormat("MMM d k:m:s ");
                graphView.setCustomLabelFormatter(new CustomLabelFormatter() {
                        @Override
                        public String formatLabel(double value, boolean isValueX) {
                                if (isValueX) {
                                        Date d = new Date((long) value);
                                        return dateFormat.format(d);
                                }
                                return null; // let graphview generate Y-axis label for us
                        }
                });
                layout.addView(graphView);
        }

}
```

Figure 55 - Method to handle graph drawing

Current Weather Conditions Report

Current weather conditions report is a very simple unique top – level view that displays the last reading from the sensors. As shown in Figure 57, the time and date of the last reading is displayed on the top of the screen. Arduino reads the sensors every six minutes. The last reading of this screenshot was at 00:10 and current time was 00:13 (top right corner) so this is confirmation that when the screenshot was taken everything worked smoothly.



Current Weather in Knossos:
Tue Oct 21 00:10:40 EEST 2014

Humidity: 77

Temperature: 14.5

Pressure: 997

Gust: 1.99

Direction: 135

Rain: 0

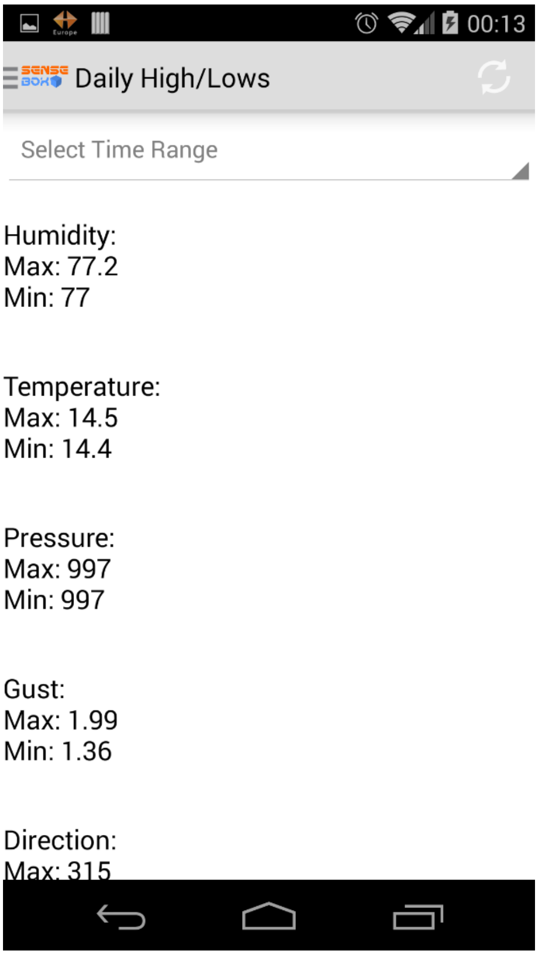
Speed: 1.99

Figure 56 - Current Weather Report

For the creation of this view a very similar process with the creation of the charts is followed. URL construction (with different flag as show in the example of Figure 72 - Dynamic SQL Query Construction), execution of HTTP Request, JSON resolving and finally instead of generating charts it displays the data in the TextViews.

Min / Max Report

This is another unique top – level view for displaying the minimum and the maximum reading of each sensor in daily, weekly, monthly and quarter basis.



Humidity:
Max: 77.2
Min: 77

Temperature:
Max: 14.5
Min: 14.4

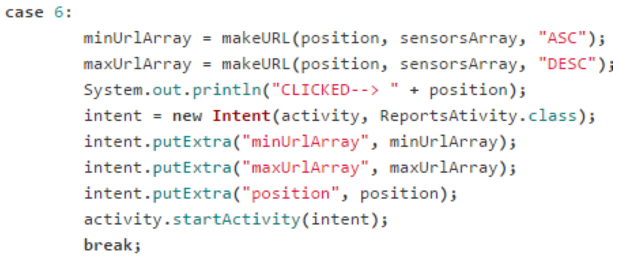
Pressure:
Max: 997
Min: 997

Gust:
Max: 1.99
Min: 1.36

Direction:
Max: 315

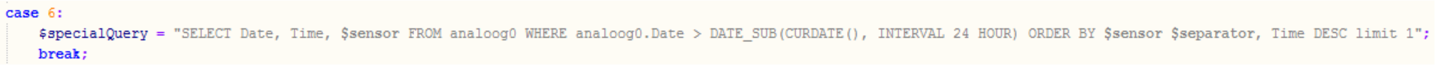
Figure 57 - High / Lows Report View

Like Current Weather Conditions Report, the creation process is similar with the creating of the charts. The difference, over the flag, that worth mentioning is that the URL construction and HTTP Requests execution processes happen twice. One for the Min and one for the Max. What separates these two is the **separator** variable that is attached to the URLs (Figure 73 - URL example). When the separator takes ASC value it refers to Min values and consequently DESC refers to Max values.



```
case 6:
        minUrlArray = makeURL(position, sensorsArray, "ASC");
        maxUrlArray = makeURL(position, sensorsArray, "DESC");
        System.out.println("CLICKED--> " + position);
        intent = new Intent(activity, ReportsAtivity.class);
        intent.putExtra("minUrlArray", minUrlArray);
        intent.putExtra("maxUrlArray", maxUrlArray);
        intent.putExtra("position", position);
        activity.startActivity(intent);
        break;
```

Speaking in SQL what exactly the separator does is to order the returned table by the sensor value in ascending or descending order and take the first result each time. This technique returns the Max/Min value.

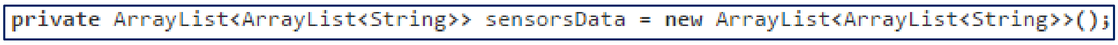


```
case 6:
    $specialQuery = "SELECT Date, Time, $sensor FROM analoog0 WHERE analoog0.Date > DATE_SUB(CURDATE(), INTERVAL 24 HOUR) ORDER BY $sensor $separator, Time DESC limit 1";
    break;
```

Data Presentation in Table Layout

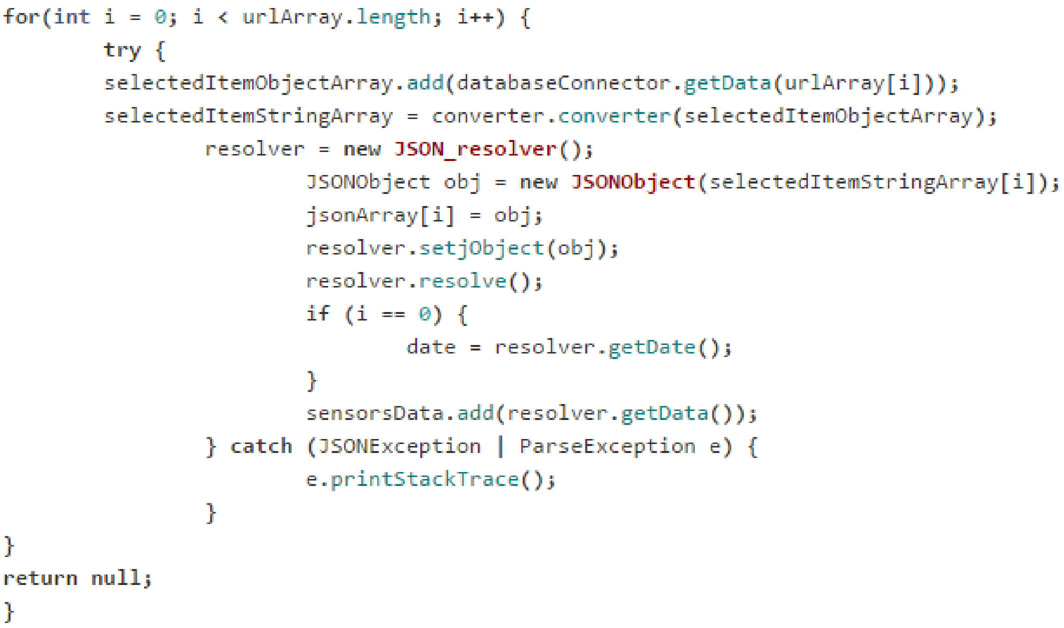
Table Layout is a complex and tricky layout that arranges its children into rows and columns. It consists of a number of TableRow objects, each defining a row.

Again, to generate the tables the first three steps are familiar. URLs construction with corresponding flag, HTTP Requests execution and JSON resolving. The next step is to add the Array Lists with the values of each sensor to an Array List of Array Lists (Figure 59 - ArrayList that holds ArrayLists).



```
private ArrayList<ArrayList<String>> sensorsData = new ArrayList<ArrayList<String>>();
```

Figure 58 - ArrayList that holds ArrayLists

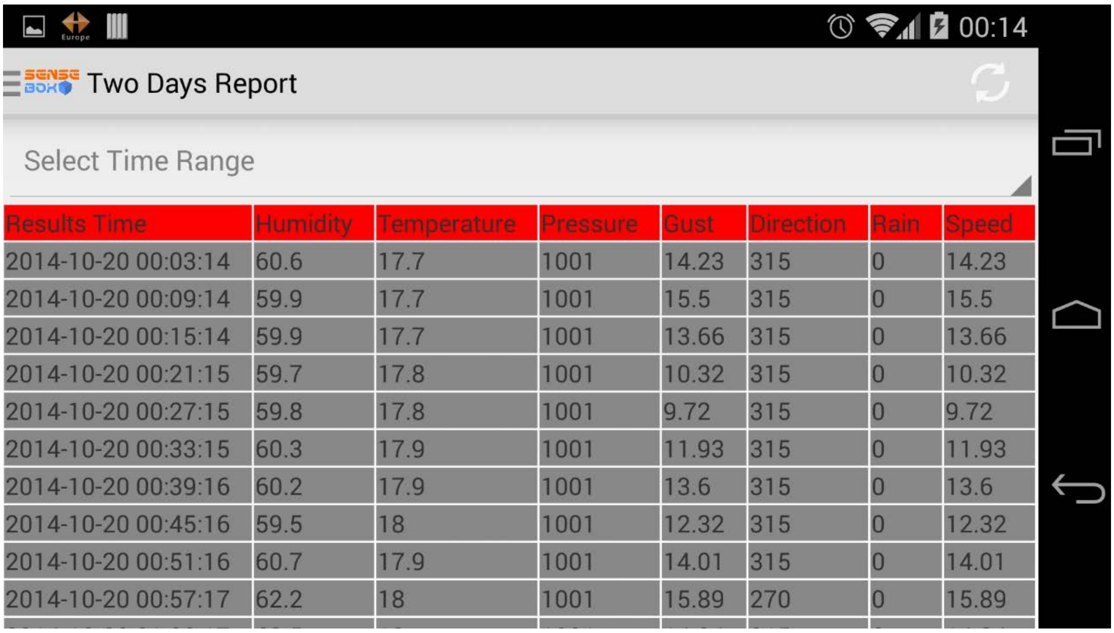


```
for(int i = 0; i < urlArray.length; i++) {
        try {
        selectedItemObjectArray.add(databaseConnector.getData(urlArray[i]));
        selectedItemStringArray = converter.converter(selectedItemObjectArray);
                resolver = new JSON_resolver();
                        JSONObject obj = new JSONObject(selectedItemStringArray[i]);
                        jsonArray[i] = obj;
                        resolver.setjObject(obj);
                        resolver.resolve();
                        if (i == 0) {
                                date = resolver.getDate();
                        }
                        sensorsData.add(resolver.getData());
                } catch (JSONException | ParseException e) {
                        e.printStackTrace();
                }
}
return null;
}
```

Figure 59 - Table Layout Creation

The last step of this procedure is to create the Table Rows object and dynamically add them to the Table Layout.

Figure 61 pictures the Table Layout. Below the Action Bar the range of the time results can be selected through the Spinner. Default is daily (24 Hours). 2 Days, Weekly, Monthly and Three Months can be selected. Also, there is a Refresh Button 🔄 to include the latest readings to the table.

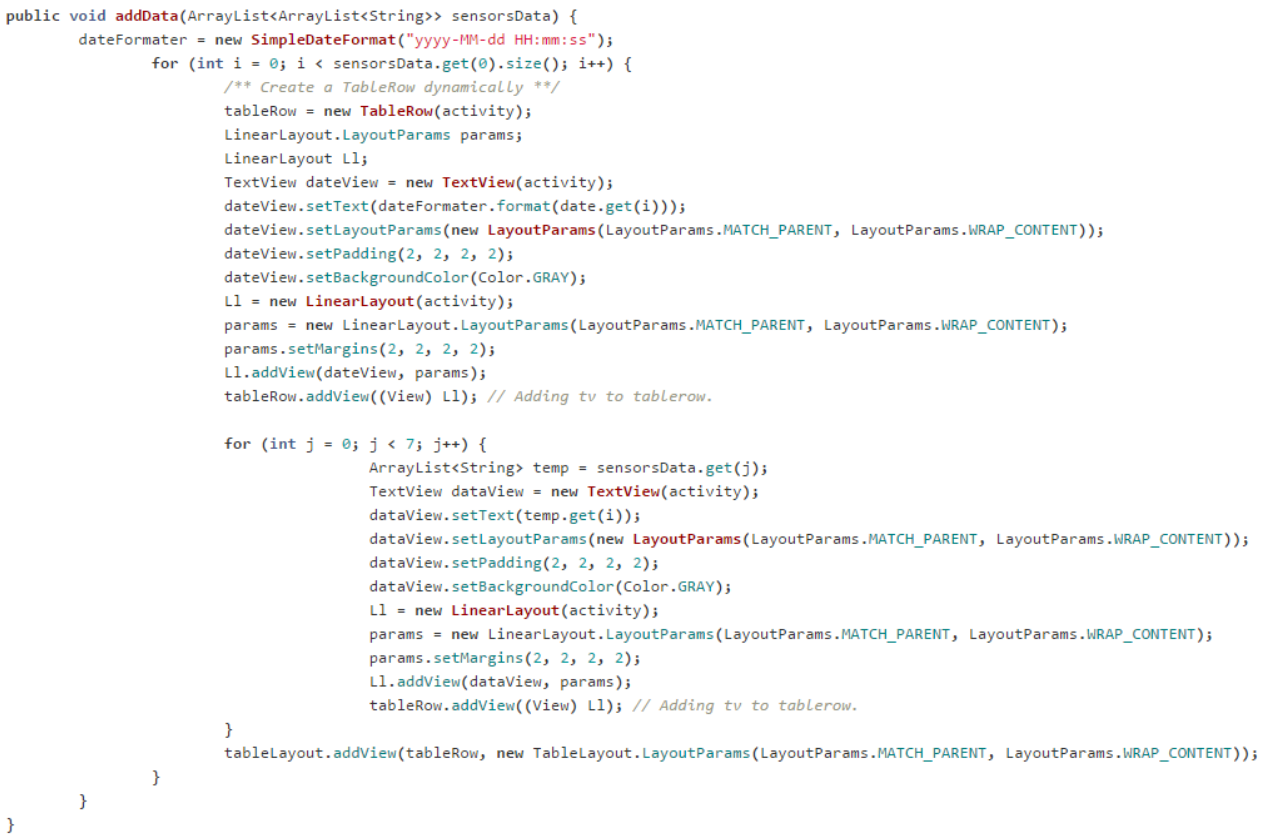


Results Time	Humidity	Temperature	Pressure	Gust	Direction	Rain	Speed
2014-10-20 00:03:14	60.6	17.7	1001	14.23	315	0	14.23
2014-10-20 00:09:14	59.9	17.7	1001	15.5	315	0	15.5
2014-10-20 00:15:14	59.9	17.7	1001	13.66	315	0	13.66
2014-10-20 00:21:15	59.7	17.8	1001	10.32	315	0	10.32
2014-10-20 00:27:15	59.8	17.8	1001	9.72	315	0	9.72
2014-10-20 00:33:15	60.3	17.9	1001	11.93	315	0	11.93
2014-10-20 00:39:16	60.2	17.9	1001	13.6	315	0	13.6
2014-10-20 00:45:16	59.5	18	1001	12.32	315	0	12.32
2014-10-20 00:51:16	60.7	17.9	1001	14.01	315	0	14.01
2014-10-20 00:57:17	62.2	18	1001	15.89	270	0	15.89

Figure 60 - Table Layout View

As shown in Figure 62 - Adding Data to Table Layout first a TableRow is instantiated. Then a TextView is instantiated and some parameters are passed to it. This TextView (in the outer for loop) is for the dates. In the inner for loop seven TextViews are created per one loop of the outer loop. These TextViews correspond to the seven sensors.

Generally, the number of the Text Views that are added to a Table Row corresponds to how many cells each row has.



```
public void addData(ArrayList<ArrayList<String>> sensorsData) {
        dateFormater = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                for (int i = 0; i < sensorsData.get(0).size(); i++) {
                        /** Create a TableRow dynamically **/
                        tableRow = new TableRow(activity);
                        LinearLayout.LayoutParams params;
                        LinearLayout Ll;
                        TextView dateView = new TextView(activity);
                        dateView.setText(dateFormater.format(date.get(i)));
                        dateView.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT));
                        dateView.setPadding(2, 2, 2, 2);
                        dateView.setBackgroundColor(Color.GRAY);
                        Ll = new LinearLayout(activity);
                        params = new LinearLayout.LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT);
                        params.setMargins(2, 2, 2, 2);
                        Ll.addView(dateView, params);
                        tableRow.addView((View) Ll); // Adding tv to tablerow.

                        for (int j = 0; j < 7; j++) {
                                        ArrayList<String> temp = sensorsData.get(j);
                                        TextView dataView = new TextView(activity);
                                        dataView.setText(temp.get(i));
                                        dataView.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT));
                                        dataView.setPadding(2, 2, 2, 2);
                                        dataView.setBackgroundColor(Color.GRAY);
                                        Ll = new LinearLayout(activity);
                                        params = new LinearLayout.LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT);
                                        params.setMargins(2, 2, 2, 2);
                                        Ll.addView(dataView, params);
                                        tableRow.addView((View) Ll); // Adding tv to tablerow.
                        }
                        tableLayout.addView(tableRow, new TableLayout.LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT));
                }
        }
}
```

Figure 61 - Adding Data to Table Layout

Finally, each TableRow that now includes the Date and the value of each sensor – eight in total - at that point of time, is added to the Table Layout with some Layout Parameters.

4.2.3. Time Lapse Video Making

The procedure of making the daily Time Lapse video is mainly consists of Bash Scripts and Cron Jobs [34]. The software utility Cron is a time – based job scheduler in UNIX – like computer OS.

The camera that capture the frames of the video is a Sony PlayStation Eye Toy. It was chosen because it is the cheapest camera (€ 6) compatible with the Raspberry PI on Raspbian. It works out of the box, directly plugged in to the PI without the need for external powered HUB with 640x480 resolution.

The first script is very simple. When executed it captures an image with the camera, using FSWEBCAM. It also names the image file according to the current date and time in a custom format. The script runs every two minutes

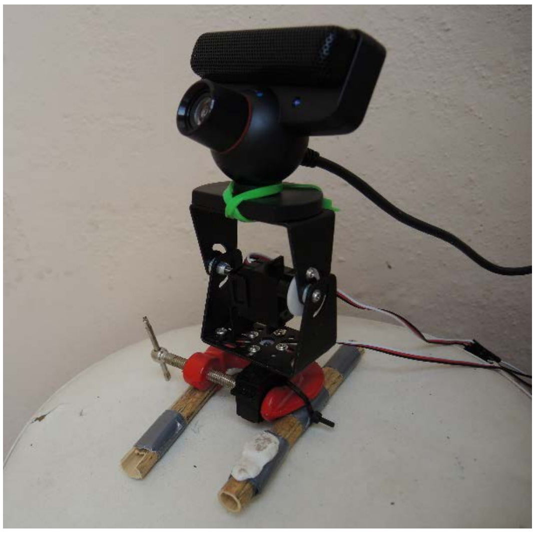
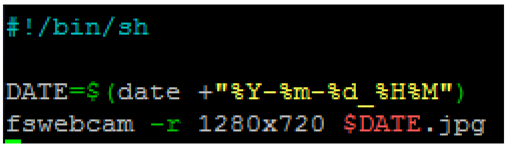


Figure 62 - Sony PlayStation Eye Toy mounted on the Pan – Tilt Bracket.



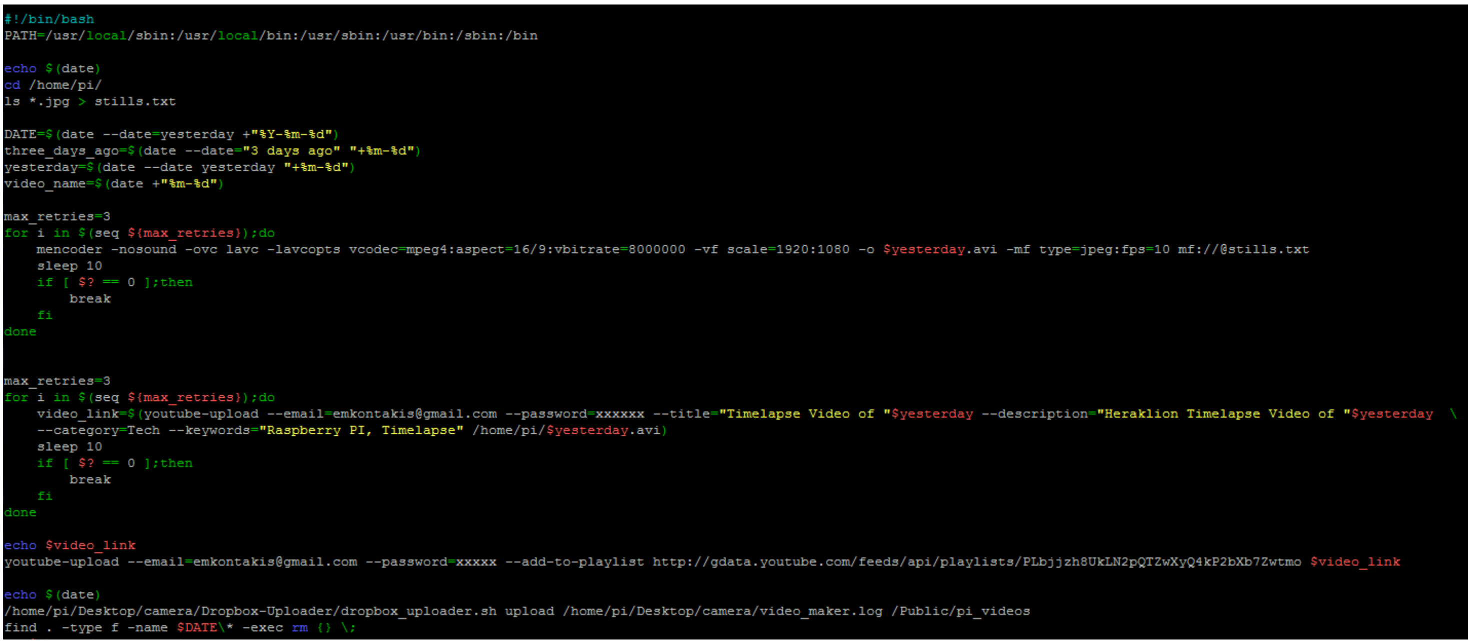
```
#!/bin/sh

DATE=$(date +"%Y-%m-%d_%H%M")
fswebcam -r 1280x720 $DATE.jpg
```

Figure 63 - Image Capture Script

Fswebcam [36] is a neat and simple webcam app. It captures images from a V4L1/V4L2 [A.2 Video4Linux] compatible device or file, averages them to reduce noise and draws a caption using the GD Graphics Library which also handles compressing the image to PNG or JPEG. The resulting image is saved to a file or sent to stdio where it can be piped to something like ncftpput or scp.

The second script is more complex. It is executed every day at midnight and is responsible for the following jobs:

- Encode the video
- Upload the video to youtube
- Upload the output of the script to dropbox



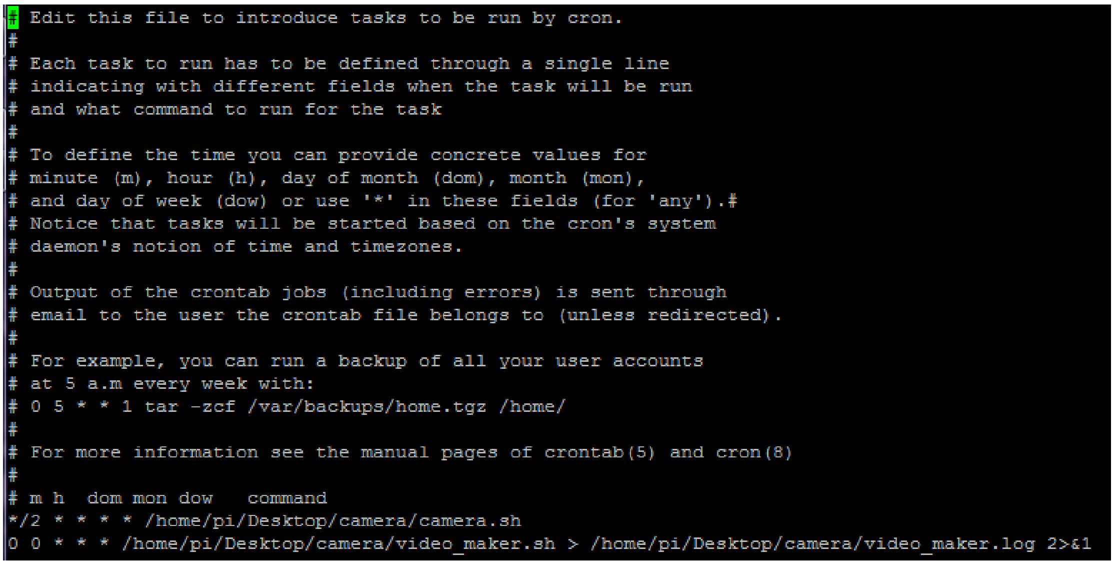
```
#!/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

echo $(date)
cd /home/pi/
ls *.jpg > stills.txt

DATE=$(date --date=yesterday +"%Y-%m-%d")
three_days_ago=$(date --date="3 days ago" "+%m-%d")
yesterday=$(date --date yesterday "+%m-%d")
video_name=$(date +"%m-%d")

max_retries=3
for i in $(seq ${max_retries});do
    mencoder -nosound -ovc lavc -lavcopts vcodec=mpeg4:aspect=16/9:vbitrate=8000000 -vf scale=1920:1080 -o $yesterday.avi -mf type=jpeg:fps=10 mf://@stills.txt
    sleep 10
    if [ $? == 0 ];then
        break
    fi
done


max_retries=3
for i in $(seq ${max_retries});do
    video_link=$(youtube-upload --email=emkontakis@gmail.com --password=xxxxxx --title="Timelapse Video of "$yesterday --description="Heraklion Timelapse Video of "$yesterday  \
    --category=Tech --keywords="Raspberry PI, Timelapse" /home/pi/$yesterday.avi)
    sleep 10
    if [ $? == 0 ];then
        break
    fi
done

echo $video_link
youtube-upload --email=emkontakis@gmail.com --password=xxxxx --add-to-playlist http://gdata.youtube.com/feeds/api/playlists/PLbjjzh8UkLN2pQTZwXyQ4kP2bXb7Zwtmo $video_link

echo $(date)
/home/pi/Desktop/camera/Dropbox-Uploader/dropbox_uploader.sh upload /home/pi/Desktop/camera/video_maker.log /Public/pi_videos
find . -type f -name $DATE\* -exec rm {} \;
```

Figure 64 - Video Maker Script

The encoding of the video is being done using the MEncoder tool. MEncoder is a free command line video decoding, encoding and filtering tool released under the GNU GPL. It is a sibling of MPlayer, and can convert all the formats that MPlayer understands into a variety of compressed and uncompressed formats using different codecs. MEncoder is included in the MPlayer distribution.

Uploading to Youtube is being handled by a command – line script written in python named Youtube-upload [38]. The script also adds the video to a playlist.

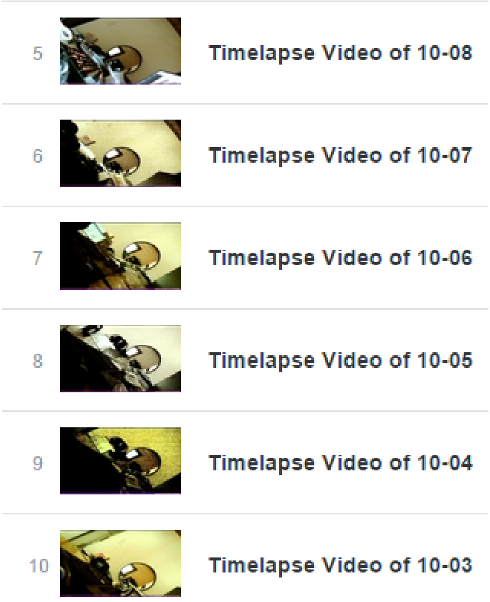
As shown in Figure 66 - Crontab for debugging purposes, the output of video_maker shell script is being redirected to video_maker.log file (2>&1 redirects stderr to stdout, meaning that both streams will be printed to the same file). This file is uploaded to the dropbox via a bash script named Dropbox Uploader [39] so the maintainer can easily keep track of the functionality of the station.



```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
*/2 * * * * /home/pi/Desktop/camera/camera.sh
0 0 * * * /home/pi/Desktop/camera/video_maker.sh > /home/pi/Desktop/camera/video_maker.log 2>&1
```

Figure 65 - Crontab

It is notable that the script uses a “retry if command fails” mechanism that re – runs the critical commands up to three times (MEncoder encoding, youtube uploading) if they return a non-zero Exit Status [A.3 Bash ‘Exit Status’].



5	Timelapse Video of 10-08
6	Timelapse Video of 10-07
7	Timelapse Video of 10-06
8	Timelapse Video of 10-05
9	Timelapse Video of 10-04
10	Timelapse Video of 10-03

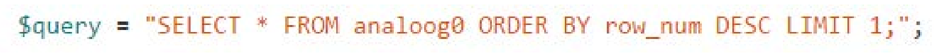
Figure 66 - Uploaded Timelapse Videos

4.2.4. Web Page

The Web Page has been built upon WordPress platform. It presents to the user the data gathered by Arduino, the images captured by the web camera and the time lapse video.

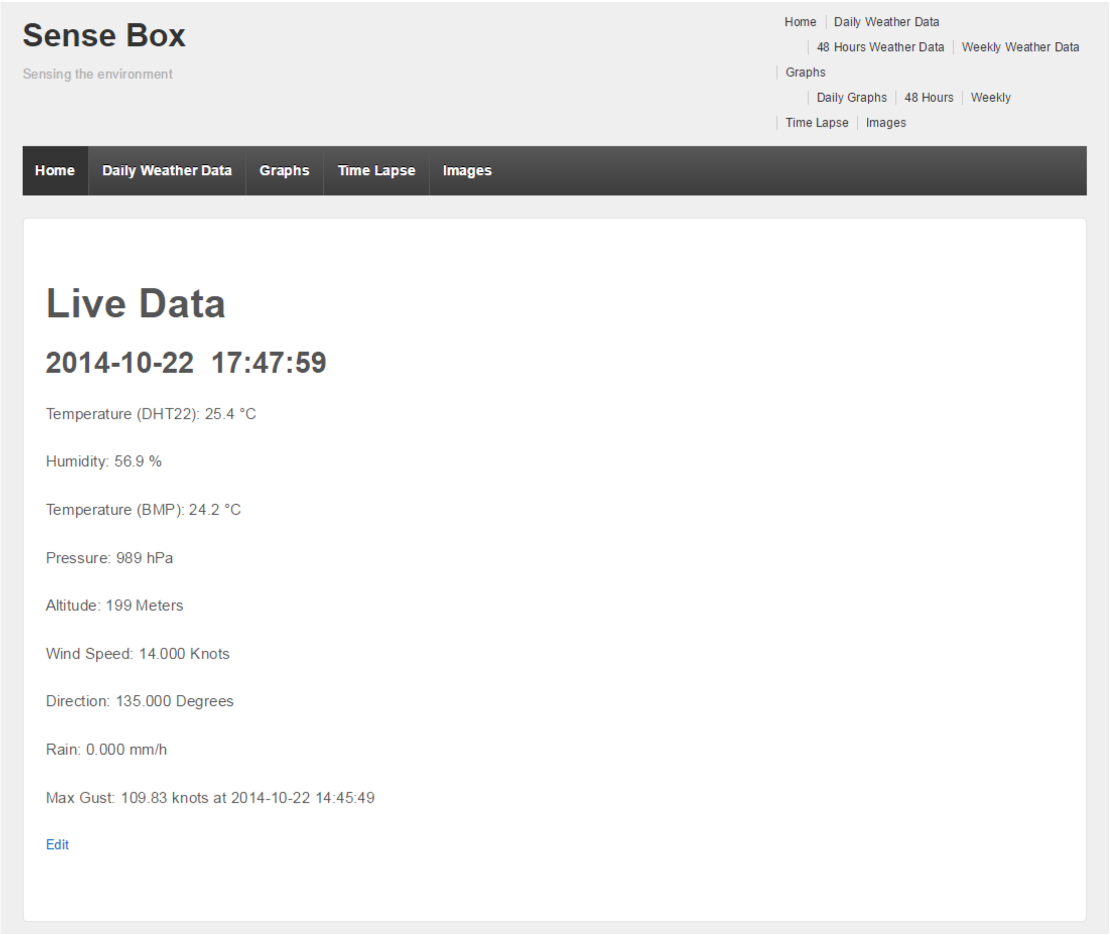
The theme used for the development of the page is Responsive [43]. As soon as the clients enters the page, the latest weather data are appeared. On the top there is the date and the time of the readings that follow. Latest results refresh every six minutes.

The procedure for this is very simple and consists of one PHP script. The script executes the SQL Query on Figure 68 and populates the returned row into the web page.



```
$query = "SELECT * FROM analoog0 ORDER BY row_num DESC LIMIT 1;";
```

Figure 67 - Latest Results Query



Live Data

2014-10-22 17:47:59

Temperature (DHT22): 25.4 °C

Humidity: 56.9 %

Temperature (BMP): 24.2 °C

Pressure: 989 hPa

Altitude: 199 Meters

Wind Speed: 14.000 Knots

Direction: 135.000 Degrees

Rain: 0.000 mm/h

Max Gust: 109.83 knots at 2014-10-22 14:45:49

Figure 68 - Home Page

Then the user can navigate to the web page through the site map on top right corner or from the main menu bar. The other pages of the site are the following:

- Weather data presentation inside a HTML table
- Graphs
- Time Lapse Video
- Images

4.2.5. Server – Side Scripts

Server-side scripting is a technique used in website design which involves embedding scripts in an HTML source code which results in a user's (client's) request to the server website being handled by a script running on the server-side before the server responds to the client's request. Scripts can be written in any of a number of server-side scripting languages that are available.

Server-side scripting is usually used to provide an interface for the client and to limit client access to proprietary databases or other data sources. These scripts may assemble client characteristics for use in customizing the response based on those characteristics, the user's requirements, access rights, etc. Server-side scripting also enables the website owner to reduce user access to the source code of server-side scripts which may be proprietary and valuable in itself.

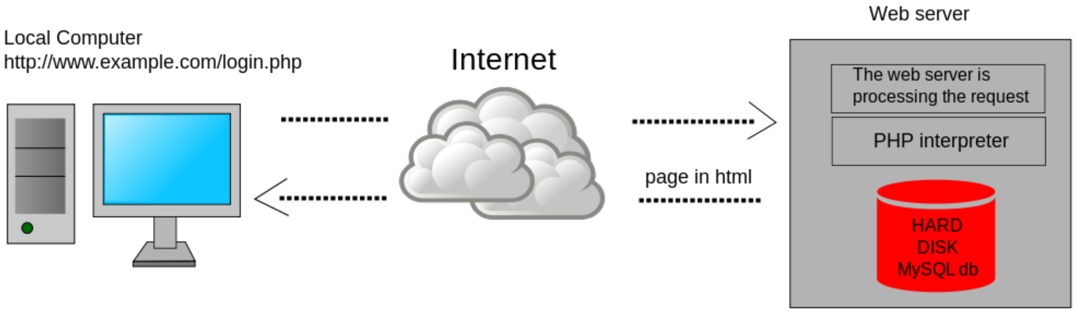


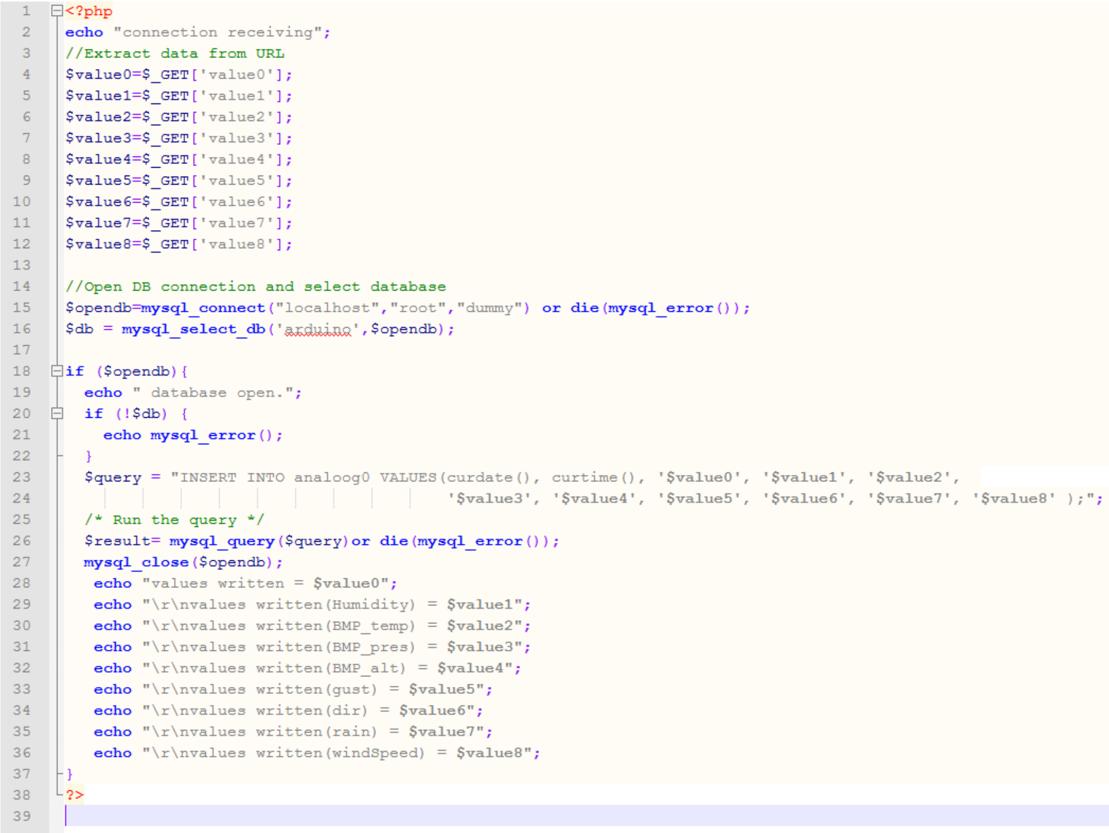
Figure 69 - Example of Server Side Scripting

4.2.5.1. PHP Scripts

PHP is a server – side scripting language designed mainly for web development but also as a general – purpose programming language. A PHP interpreter is usually responsible to process PHP, which is usually implemented as a web server's native module. After the PHP code is interpreted and executed, the web server sends resulting output to its client. This is exactly how PHP is used in the project.

Storing Data in **Database**

In *Figure 33 - URL contruction* the URL is constructed and data are URL encoded. The script in Figure 71, extracts the data from the URL, establishes a connection to the MySQL database and stores data. For debugging purposes, it returns to the client the data values stored.



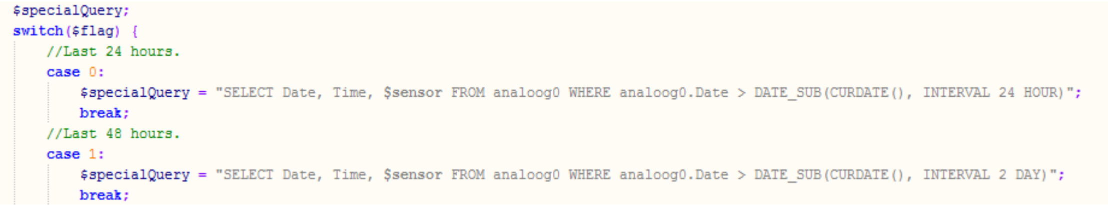
```
<?php
echo "connection receiving";
//Extract data from URL
$value0=$_GET['value0'];
$value1=$_GET['value1'];
$value2=$_GET['value2'];
$value3=$_GET['value3'];
$value4=$_GET['value4'];
$value5=$_GET['value5'];
$value6=$_GET['value6'];
$value7=$_GET['value7'];
$value8=$_GET['value8'];

//Open DB connection and select database
$opendb=mysql_connect("localhost","root","dummy") or die(mysql_error());
$db = mysql_select_db('arduino',$opendb);

if ($opendb){
  echo " database open.";
  if (!$db) {
    echo mysql_error();
  }
  $query = "INSERT INTO analoog0 VALUES(curdate(), curtime(), '$value0', '$value1', '$value2',
                                        '$value3', '$value4', '$value5', '$value6', '$value7', '$value8' );";
  /* Run the query */
  $result= mysql_query($query)or die(mysql_error());
  mysql_close($opendb);
   echo "values written = $value0";
   echo "\r\nvalues written(Humidity) = $value1";
   echo "\r\nvalues written(BMP_temp) = $value2";
   echo "\r\nvalues written(BMP_pres) = $value3";
   echo "\r\nvalues written(BMP_alt) = $value4";
   echo "\r\nvalues written(gust) = $value5";
   echo "\r\nvalues written(dir) = $value6";
   echo "\r\nvalues written(rain) = $value7";
   echo "\r\nvalues written(windSpeed) = $value8";
}
?>
```

Figure 70 - Script to store data to database

Android Application Data Fetching Script

This PHP script constructs the corresponding SQL Query depending on three variables attached on the URL, the Sensor, the Flag and the Separator.

The Sensor variable declares the sensors whom data will be fetched. The Flag informs the script which selection was clicked from the Navigation Drawer menu while the Separator variable separates Min and Max values.



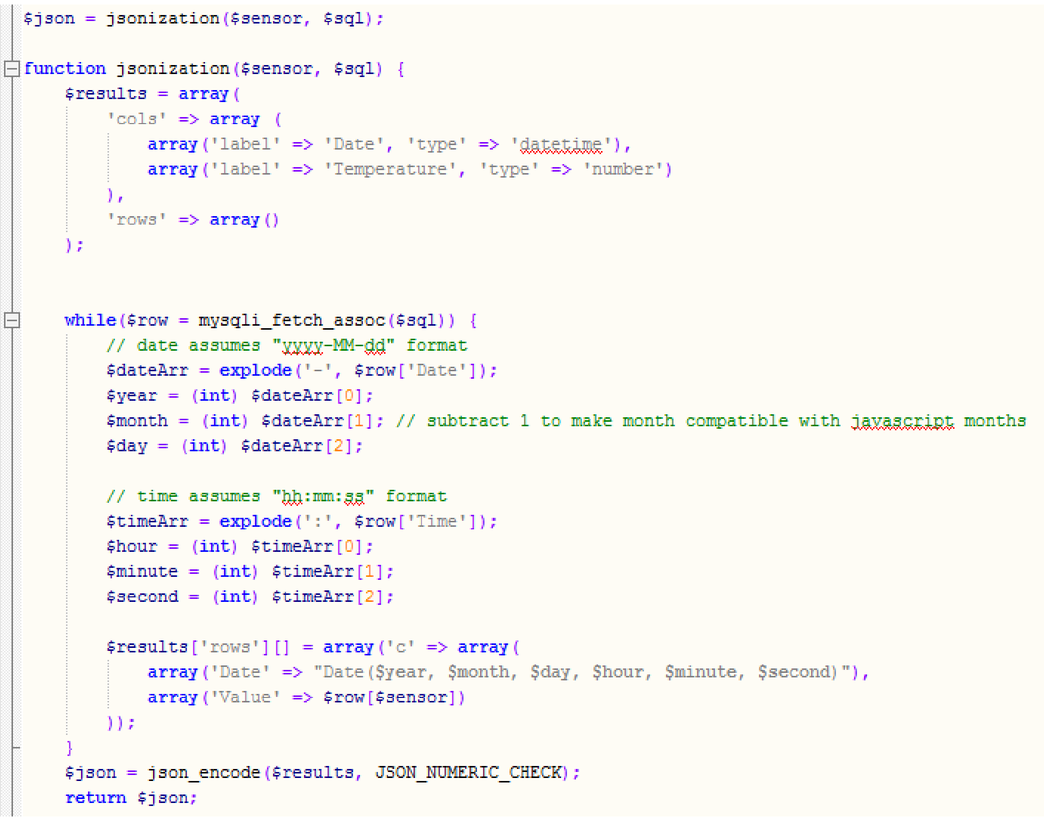
```
$specialQuery;
switch($flag) {
    //Last 24 hours.
    case 0:
        $specialQuery = "SELECT Date, Time, $sensor FROM analoog0 WHERE analoog0.Date > DATE_SUB(CURDATE(), INTERVAL 24 HOUR)";
        break;
    //Last 48 hours.
    case 1:
        $specialQuery = "SELECT Date, Time, $sensor FROM analoog0 WHERE analoog0.Date > DATE_SUB(CURDATE(), INTERVAL 2 DAY)";
        break;
```

Figure 71 - Dynamic SQL Query Construction

http://sensebox.noip.me/dynamicQueryExecutor.php?sensor=BMP_temp&flag=6&separator=ASC

Figure 72 - URL example

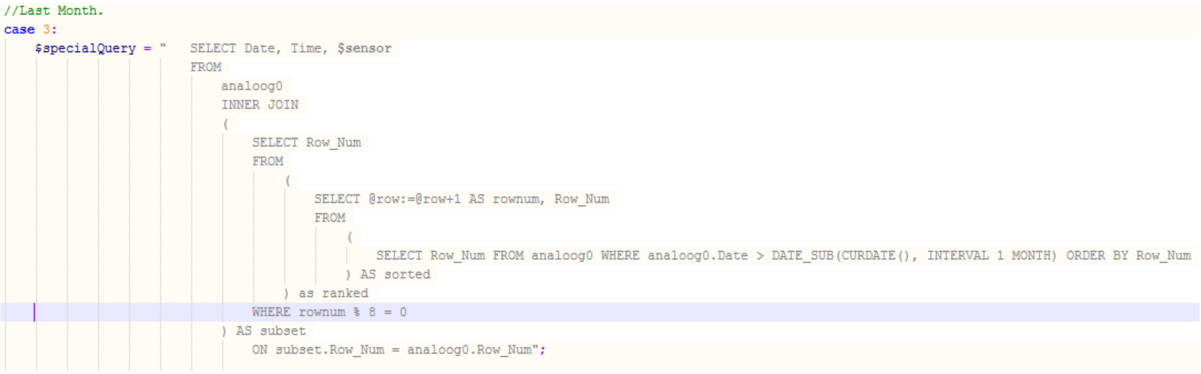
After the SQL Query is constructed it is executed. The script returns the weather data in JSON format to Android Application. The construction of the JSON is shown in Figure 74



```
$json = jsonization($sensor, $sql);

function jsonization($sensor, $sql) {
    $results = array(
        'cols' => array (
            array('label' => 'Date', 'type' => 'datetime'),
            array('label' => 'Temperature', 'type' => 'number')
        ),
        'rows' => array()
    );


    while($row = mysqli_fetch_assoc($sql)) {
        // date assumes "yyyy-MM-dd" format
        $dateArr = explode('-', $row['Date']);
        $year = (int) $dateArr[0];
        $month = (int) $dateArr[1]; // subtract 1 to make month compatible with javascript months
        $day = (int) $dateArr[2];

        // time assumes "hh:mm:ss" format
        $timeArr = explode(':', $row['Time']);
        $hour = (int) $timeArr[0];
        $minute = (int) $timeArr[1];
        $second = (int) $timeArr[2];

        $results['rows'][] = array('c' => array(
            array('Date' => "Date($year, $month, $day, $hour, $minute, $second)"),
            array('Value' => $row[$sensor])
        ));
    }
    $json = json_encode($results, JSON_NUMERIC_CHECK);
    return $json;
```

Figure 73 - SQL Table to JSON

Something that worth mentioning here are the cases of the Weekly, Monthly and Three Months Charts. In these three cases high resolution of the charts are of no importance. Also generating seven charts with so many samples (about 22000 for Three months, 7333 for one month per chart) makes the application too heavy for a mobile device. To fix this problem one row per two rows are selected from the table for the weekly graph, one per 8 for the monthly and one per 10 for the 3 months chart. The SQL Query for the Monthly chart is shown below.



```
//Last Month.
case 3:
    $specialQuery = "   SELECT Date, Time, $sensor
                        FROM
                            analoog0
                            INNER JOIN
                            (
                                SELECT Row_Num
                                FROM
                                    (
                                        SELECT @row:=@row+1 AS rownum, Row_Num
                                        FROM
                                            (
                                                SELECT Row_Num FROM analoog0 WHERE analoog0.Date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH) ORDER BY Row_Num
                                            ) AS sorted
                                    ) as ranked
                                WHERE rownum % 8 = 0
                            ) AS subset
                                ON subset.Row_Num = analoog0.Row_Num";
```

Figure 74 - SQL Query to select every n-th row

4.2.5.2. JavaScript Script

JavaScript is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed. It is also used in server-side network programming with frameworks such as Node.js, game development and the creation of desktop and mobile applications. Newer and faster JavaScript virtual machines (VMs) and platforms built upon them (notably Node.js) have also increased the popularity of JavaScript for server-side web applications.

Google Charts Script

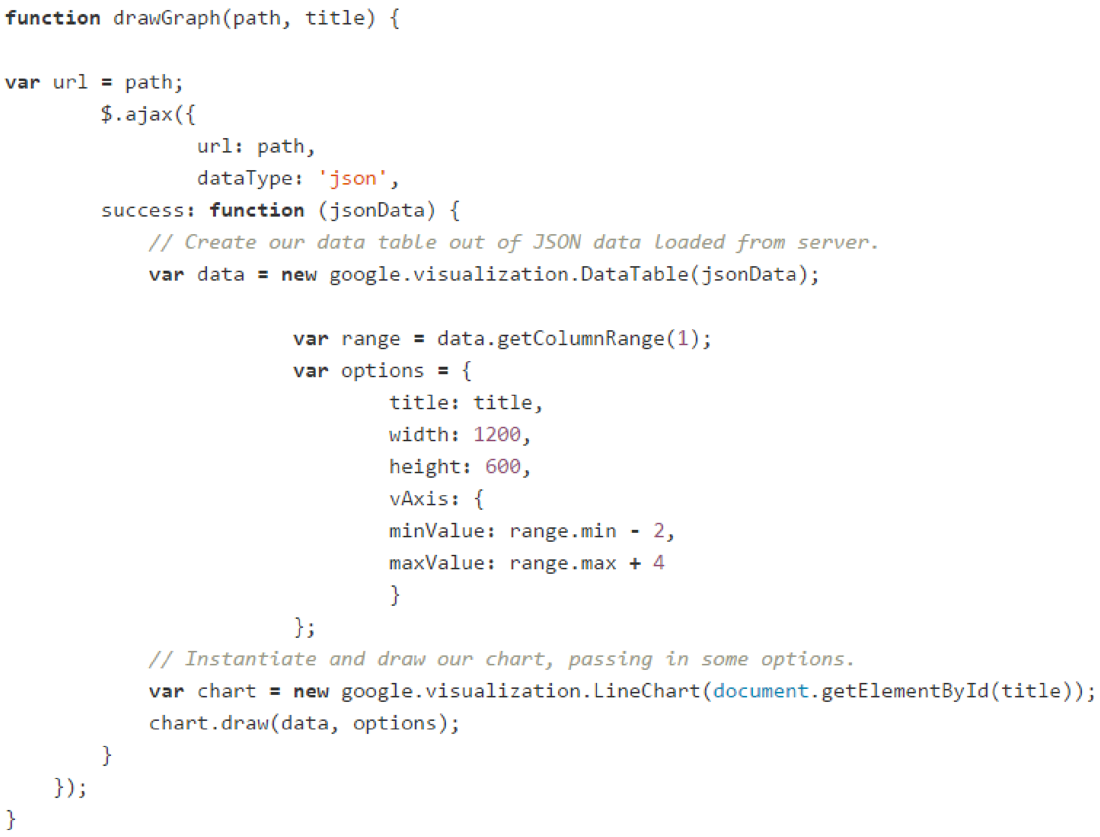
Google uses JavaScript to embed the Charts in the web page.

As soon as the Google Visualization API is loaded a function responsible for building the URLs is loaded. These URLs when executed returns the Weather Data using a slightly different version of Android Application Data Fetching Script. The logic is the same:

- Dynamic SQL Query Construction
- JSON Encode the returned Weather Data

```
var query = 'http://192.168.1.5/dynamicQueryExecutor[web].php?';
var urlArray = [];

function queryBuilder() {
        for (i = 0; i < sensorsArray.length; i++) {
                urlArray[i] = query + "sensor=" + sensorsArray[i] + "&flag=0" + "&title=" + titles[i];
        }
}
```

Then the function that handles the creation of the charts is loaded. This function calls the jQuery ajax() function [41] to send a query to a URL and get back a JSON string. The URL here is of the URLs passed to the function (one URL per sensor). The returned data is converted to a DataTable (var data). This DataTable is used to populate a line chart, which is then rendered on the page. Generally, all Charts are populated with data using the DataTable class, making it easy to switch between chart types. This function is called as many times as the number of the sensors (7). [42]



```
function drawGraph(path, title) {

var url = path;
        $.ajax({
                url: path,
                dataType: 'json',
        success: function (jsonData) {
            // Create our data table out of JSON data loaded from server.
            var data = new google.visualization.DataTable(jsonData);

                        var range = data.getColumnRange(1);
                        var options = {
                                title: title,
                                width: 1200,
                                height: 600,
                                vAxis: {
                                minValue: range.min - 2,
                                maxValue: range.max + 4
                                }
                        };
            // Instantiate and draw our chart, passing in some options.
            var chart = new google.visualization.LineChart(document.getElementById(title));
            chart.draw(data, options);
        }
    });
}
```

Figure 75 - Google Charts Chart Draw Function

References

[1] Open Source Initiative: The Open Source Definition.
http://opensource.org/docs/osd

[2] Open Source Hardware Association: OSHW Definition.
http://www.oshwa.org/definition/

[3] Definition of Free Cultural Works: *Open Source Hardware.*
http://freedomdefined.org/OSHW

[4] Arduino: Homepage
http://arduino.cc/en

[5] Arduino: *Reference.*
http://arduino.cc/en/Reference/HomePage

[6] World Meteorological Organization, 1992: *International Meteorological Vocabulary*. Second edition, WMO-No. 182,Geneva.

[7] Apache License, Version 2.0
http://www.apache.org/licenses/LICENSE-2.0.html

[8] GNU General Public License, Version 2.0
http://www.gnu.org/licenses/gpl-2.0.html

[9] System on a chip
http://en.wikipedia.org/wiki/System_on_a_chip

[10] Google Charts API
https://developers.google.com/chart/

[11] Arduino Mega 2560 R3 Web page.
http://arduino.cc/en/Main/ArduinoBoardMega2560

[12] Arduino Ethernet Shield Web page.
http://arduino.cc/en/Main/ArduinoEthernetShield

[13] Arduino Ethernet Library Web page.
http://arduino.cc/en/reference/ethernet

[14] Arduino SD Library Web page.
http://arduino.cc/en/Reference/SD

[15] RHT03 Official Datasheet.
http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Weather/RHT03.pdf

[16] Bosch Sensortec: Pressure Sensors.
http://goo.gl/SEpmgY

[17] BMP085 Datasheet.
http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Pressure/BST-BMP085-DS000-06.pdf

[18] Electrostatic Sensitive Device
http://en.wikipedia.org/wiki/Electrostatic-sensitive_device

[19] Sparkfun Electronics: Weather Meters Kit
https://www.sparkfun.com/products/8942

[20] Sparkfun Electronics: Weahter Meters Datasheet
https://www.sparkfun.com/datasheets/Sensors/Weather/Weather%20Sensor%20Assembly..pdf

[21] Fritzing
http://fritzing.org/home/

[22] Adafruit Industries
https://www.adafruit.com/

[23] Arduino Sketch Repository Link

https://github.com/mKontakis/Sensebox/blob/master/WeatherMeters/ver11/ver11.ino

[24] <AVR/pgmspace.h>: Program Space Utilities:
http://www.nongnu.org/avr-libc/user-manual/group__avr__pgmspace.html

[25] PROGMEM
http://arduino.cc/en/Reference/PROGMEM

[26] HTTP Request: GET
http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.3

[27] Pan – Tilt Bracket
https://www.sparkfun.com/products/10335

[28] Android Application Repository
https://github.com/mKontakis/Sensebox/blob/master/SenseBox.Project

[29] Android GraphView
http://android-graphview.org/

[30] Android Navigation Drawer
https://developer.android.com/design/patterns/navigation-drawer.html+

[31] Raspberry PI
ww.raspberrypi.org

[32] Apache HTTP Server
http://httpd.apache.org/

[33] WordPress
https://wordpress.org/

[34] Cron
http://en.wikipedia.org/wiki/Cron

[35] ATmega328P Data sheet
http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf

[36] Fswebcam
http://www.firestorm.cx/fswebcam/

[37] MPlayer – Mencoder
http://www.mplayerhq.hu/

[38] YouTube – Upload Script
https://code.google.com/p/youtube-upload/

[39] Dropbox Uploader
https://github.com/andreafabrizi/Dropbox-Uploader

[40] Android Spinners
http://developer.android.com/guide/topics/ui/controls/spinner.html

[41] jQuery ajax function
http://api.jquery.com/jQuery.ajax/

[42] Google Charts DataTables
https://developers.google.com/chart/interactive/docs/datatables_dataviews

[43] Responsive WordPress Theme
https://wordpress.org/themes/responsive

[44] Arduino Community Thread
http://forum.arduino.cc/index.php?topic=227643.0

Appendix

A.1 Why do ISPs prefer dynamic IP

When ISPs were first starting, everyone connected to the Internet over a modem. And most people used the Internet for a few minutes to a few hours per week. Assigning a static IP to every subscriber would have been very expensive, for something that most people used just a few minutes a week.

As broadband connections have become more common, the practical reasons for not assigning a static IP have become much less noticeable, as now the majority of connections are "always-on"--even when nobody is (actively) using the Internet.

So there's a bit of a historical reason not to use static IPs--customers are already accustomed to using dynamic IPs.

When modern ISPs enforce dynamic IPs these days, it may be in part to distinguish between "consumer" and "professional" services--by reserving static IPs for customers who pay more, it gives customers who need that feature an incentive to upgrade their service level.

It can also serve as a deterrent for people abusing their consumer-grade service. Many ISPs, for instance, explicitly prohibit running "servers" on a home Internet connection. If every home user had a static IP, they'd be more inclined to abuse such terms of service.

It's also less of a management problem to assign customers dynamic IPs. If you move across town (but within the same ISP's service area), there's no need to re-assign how your static IP is routed; you'll just get a dynamic IP that exists in the new neighborhood.

A.2 Video4Linux

Video4Linux or V4L is a video capture and output device API and driver framework for the Linux kernel, supporting many USB webcams, TV tuners, and other devices. Video4Linux is closely integrated with the Linux kernel. Video4Linux was named after Video for Windows (which is sometimes abbreviated "V4W"), but is not technically related to it.

- V4L had been introduced late into the 2.1.X development cycle of the Linux kernel. V4L1 support was dropped in kernel 2.6.38.
- V4L2 is the second version of V4L. Video4Linux2 fixed some design bugs and started appearing in the 2.5.X kernels. Video4Linux2 drivers include a compatibility mode for Video4Linux1 applications, though the support can be incomplete and it is recommended to use Video4Linux1 devices in V4L2 mode. The project DVB-Wiki is now hosted on Linux TV web site.

A.3 Bash 'Exit Status'

The exit status of an executed command is the value returned by the waitpid system call or equivalent function. Exit statuses fall between 0 and 255, though, as explained below, the shell may use values above 125 specially. Exit statuses from shell builtins and compound commands are also limited to this range. Under certain circumstances, the shell will use special values to indicate specific failure modes.

For the shell's purposes, a command which exits with a zero exit status has succeeded. A non-zero exit status indicates failure. This seemingly counter-intuitive scheme is used so there is one well-defined way to indicate success and a variety of ways to indicate various failure modes. When a command terminates on a fatal signal whose number is N, Bash uses the value 128+N as the exit status.

If a command is not found, the child process created to execute it returns a status of 127. If a command is found but is not executable, the return status is 126.

If a command fails because of an error during expansion or redirection, the exit status is greater than zero.

All of the Bash builtins return an exit status of zero if they succeed and a non-zero status on failure, so they may be used by the conditional and list constructs.