Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Hochschule für Angewandte Wissenschaften Hamburg

Fakultät Life Sciences

Softwareentwickulng zur Erfassung, Verarbeitung und datenbankbasierter Archivierung von Daten eines Ultraschallanemometers

Software development for gathering, processing and data bank storage of data taken from a sonic Anemometer

Bachelorarbeit
im Verfahrenstechnik

vorgelegt von
Teodoro Abdo Duran
Matrikelnummer: 2042042
Hamburg
am 30. September 2014

Gutachter: Prof. Dr.	**Constantin Canavas (HAW Hamburg)**
Gutachter: Dipl.-Ing.	**Ilja Knippschild (HAW Hamburg)**

Abstract

This thesis describes the advantages of replacing a three cup anemometer and wind vane for a sonic anemometer and focuses on the software development needed to use the new sonic anemometer with the existing installations. As development progresses, more and more features are added to the program. The end result is a fully functioning program capable of reading the information, processing it and using it to calculate the wind speed and direction among six other measurements and save them in a MYSQL database with reliability. This thesis was written entirely in English but uses some sources written in German.

Erklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorarbeit mit dem im Ausgabenantrag formuleirten Thema ohne fremde Hilfe sebstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellensind unter Angabe der Quellen kenntlich gemacht.

Hamburg, den 30.09.2014 ------------------------------------

Teodoro Abdo Duran

Vorwort

Die vorliegende Arbeit wurde in der Zeit vom Juli 2014 bis zum September 2014 am Labor für Automatisierungstechnik der Hochschule für Angewandte Wissenschaften in Hamburg durchgeführt.

An dieser Stelle möchte ich mich herzlich bedanken, die die vorliegende Arbeit ermöglicht haben:

Bei Herr Prof. Dr. Constantin Canavas möchte ich mich für seine Zeit, Geduld und freundliche Unterstützung bedanken.

Bei Herr Dipl.-Ing. Ilja Knippschild für die Betreuung und fachliche Unterstützung.

Ein ganz besonderer Dank bei Herr Dipl. Ing. Peter Krüß für die Unterstützung im Labor, die Hilfsbereitschaft und die immer freundliche Diskussionen.

Index

Figures and tables Index

1. Introduction

1.1 Context

Before starting with “Software development for gathering, processing and data bank storage of data taken from a sonic Anemometer” we have to understand the context of this Thesis.

A weather station is a combination of instruments and equipment for measuring atmospheric conditions to provide information for weather forecasts and to study the climate changes. In the Faculty of Life Sciences on the Bergedorf Campus of the HAW Hamburg there is a weather station in use. Right now, the meteorological parameters being measured there are Temperature, Air Pressure, Global Radiation, amount of rainfall, Wind Speed and wind direction.

It has come to the attention of the Staff working with the weather station that the measurement of the wind speed and direction could be improved. Right now a three cup anemometer is being used for the wind Speed measurements and a wind vane for the wind direction. The problems with these devices are the speed at which they adapt to new conditions and the accuracy of their readings. Furthermore, while the three cup anemometer is a great device to measure the wind speed, it can only measure it in the horizontal plane, meaning the vertical component of the wind force is lost, causing a loss of precision.

To improve these readings, a sonic anemometer has been acquired, which should adapt faster and provide more accurate readings as well as measure the missing vertical component of the wind force. The goal is not to simply replace these machines but to make a statistical comparison with the data gathered and improve the overall accuracy of the weather station.

The sonic Anemometer has already been assembled and calibrated but it cannot be added to the weather station yet for several reasons:

-It came with software used to configure it and read the data values it outputs, unfortunately this program runs on windows while the weather stations infrastructure runs on Linux.

-The sonic anemometer provides its data through a Serial communications port and the data it outputs are only the X, Y and Z components of the wind force as well as the temperature. This raw data needs to be processed in order to be useful.

-There is no way to save the data directly into the weather stations data bank to make a statistical comparison.

1.2 Scope of work

The main goal of my work is to develop software capable of reading the output from the sonic anemometer, process the raw data into useful information and save this data into the weather stations data bank.

While this task sounds simple enough, there are several parameters and conditions that need to be taken into account:

- The software needs to be programmed in the “C” Programming language.
- It needs to be capable of opening and reading from the serial communications port.
- It needs to take into account possible errors.
- It needs to be able to accept data bank and configuration changes.
- It needs to be well documented and keep an error log.

After creating the software several tests will have to be made to ensure the working conditions of the program.

1.3 Thesis Structure

I have written this thesis in a specific order so that is easy to follow and understand as it progresses. First it begins with a little background knowledge that any reader would need in order to fully understand the concepts that follow. Then I will present the equipment I will be using during this project so that the work could be easily reproduced. I will quickly describe how was it all set up and all the preparations needed to begin with the project.

I start the basic development by doing a “proof of concept”, this means I did everything I could to get the most basic functions of the program to work just to see if the project was at all possible. After successfully confirming its feasibility, I started to fully develop the main functions of the program in the advanced development section.

With the program in working conditions I then proceeded to test it under real world conditions to ensure its reliability. The tests showed areas where the program could be improved, which I address later in the “Fine tuning” section. With the main functions fully developed and improved reliability, I started the development of additional features, which are not essential for the core functions of the program but are very useful to have.

Finally I describe the final product with the help of a flowchart and write my conclusion, thoughts and future recommendations for the future development of the project.

2. Theory

To be able to understand and follow everything that will be discussed next, I will give a quick overview of the things we need to know to have a basic understanding of the problems ahead.

2.1 Physical principles

2.1.1 Fluid Mechanics

Fluid mechanics is the study of the physical behavior of fluids. It is also important in theoretical engineering, it finds its foundations in continuum mechanics, that of classical physics.

The turbulent flow is described as the movement of fluids in a chaotic fashion. This flow pattern is mostly a three-dimensional flow field with a time and space seemingly randomly varying component. In **Fig.1** there is an illustration of what the turbulent flow looks like (b). The opposite is the laminar flow (a).

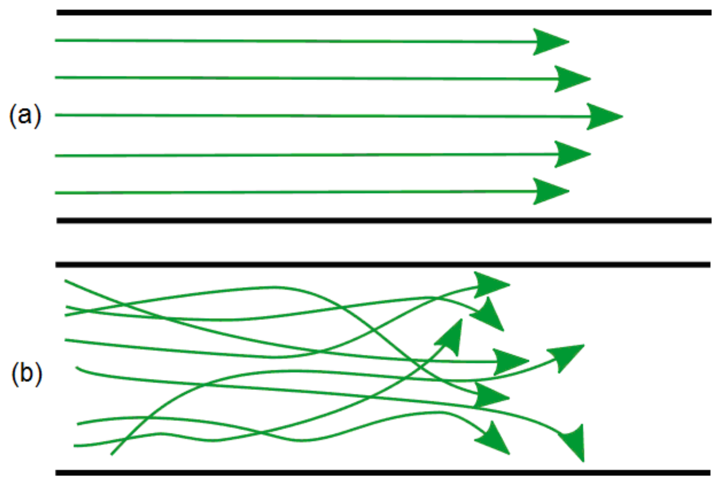


Figure 1 Laminar flow and turbulent flow [1]

Turbulence leads to increased mixing. The Reynolds number is a parameter without dimension. The turbulent behavior of geometrically similar bodies with the same Reynolds number is identical. This property allows, for example, realistic model tests in a wind tunnel or water channel. As shown in **Fig.2** flows at Reynolds numbers larger than 4000 are typically turbulent, while those at low Reynolds numbers usually remain laminar.

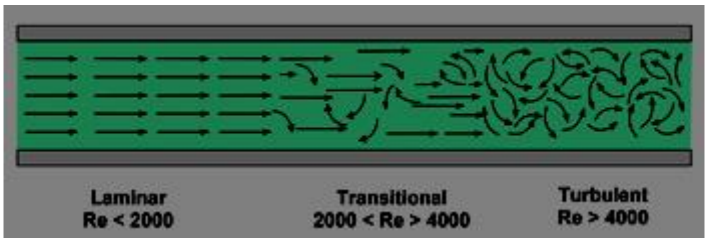


Figure 2 Change in flow based on the Reynolds number [2]

The Reynolds number is defined as:

$$Re = \frac{\rho \cdot v \cdot d}{\eta}$$

ρ	Density of the fluid (kg/m^3)
v	Mean velocity of the fluid (m/s)
d	Characteristic linear dimension (m)
η	Dynamic viscosity of the fluid (kg/(m·s))

The density of air at 20 °C at sea level is around 1,2041 kg/m^3. Given that the viscosity of air is extremely small and the characteristic linear dimension of the outside world would be huge, it can be assumed that the flow of wind will be turbulent by almost any speed.

2.1.2 Ultrasound

Ultrasound refers to sound with frequencies above the audible frequency range of people. The audible range of hearing varies from person to person but "the full range of human hearing extends from 20 to 20,000 hertz."[14]. Ultrasound covers frequencies from about 20 kHz and up. In contrast frequencies below the audible frequency range of people is called infrasound. This is better represented in **Fig.4**.

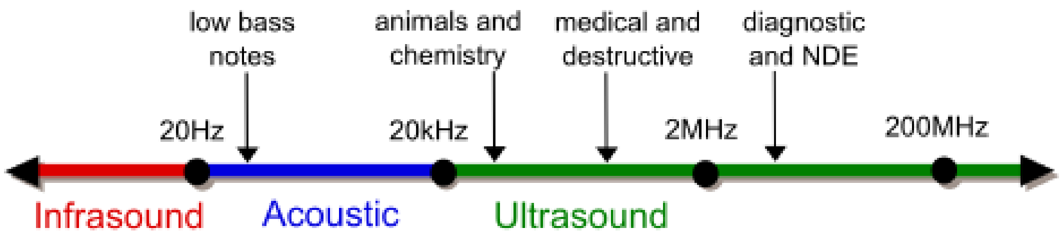


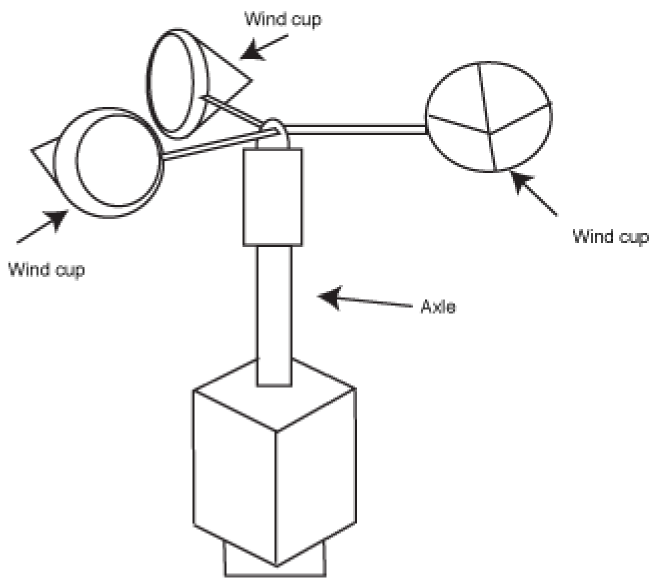
Figure 2 sound frequency ranges [3]

In gases and liquids ultrasound propagates predominantly as a longitudinal wave. The higher the frequency of the waves, the higher the acoustic impedance of air.

2.1.3 Anemometer

The anemometer is an instrument for measuring the wind speed. Anemometers are used not only in weather stations, but in mines, tunnels, and ventilation systems.

Of all anemometers, the three cup anemometer is the most widely used. It's built with three metal cups attached to the ends of horizontal shafts mounted on a vertical axle. Wind inside the cups causes them to rotate. This rotation can be measured to calculate the wind speed in meters per second. It is common for an anemometer to be connected to an electrical generator to get more precise measurements, given that the amount of current produced by the generator would vary depending on the wind speed.



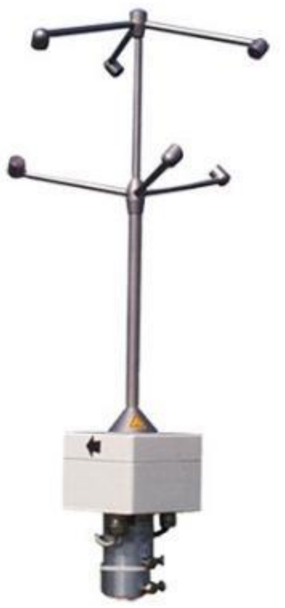


Figure 3 Schema of a 3 cup anemometer and sonic anemometer [4] [5]

Sonic anemometers use Ultrasound waves to measure the wind speed. The ultrasound waves are measured within a fixed distance so that the speed and time it takes the wave to travel will be only dependent of the air in which they propagate. High frequencies are used for short distances so that speed can be determined with more precision. Since the speed of sound depends on the temperature and the humidity, measurements are always determined after the wave travels both directions. From the difference the virtual temperature can be calculated. In **Fig.5** the difference in structure between the 3-cup anemometer and the sonic anemometer.

A sonic anemometer usually has several measuring distances between ultrasonic transmitters and receivers. Alternating the wave source, the speed of sound can be measured in different spatial directions. Advantages of the sonic anemometer are the higher accuracy, the lack of inertia in the system and the possibility of additional detection of the vertical wind component.

2.2 Software principles

2.2.1 Database

A database is a system for electronic data management. The essential task of a database is to provide large amounts of data efficiently, consistently and permanently store it in different forms to suit the demands of users and application programs.
A database system consists of two parts, the management software and the actual database with all the stored information.

The management software is called database management system (DBMS). The one used here is MYSQL. It organizes internal storage of data and controls all read and write accesses to the database. To query and manage data, MYSQL provides a database language to interact with the database. A database can be thought of as an electronic filing system.

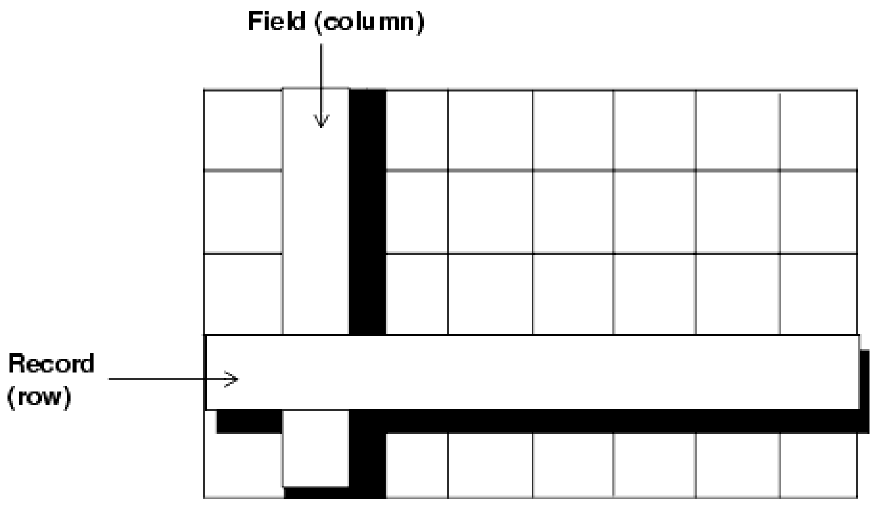


Figure 4 difference between Field and Record [6]

Traditional databases use tables which are organized by fields and records. A field is a column which has a single piece of information and a record is a row with one complete set of fields as shown on **Fig. 6**

2.2.2 Serial Programming

In order to communicate with serial connections, many settings like baud rate, stop bits, parity and data bits are required.

Baud rate

The speed in serial ports is measured in bauds or baud rate. It is often confused with bitrate, which indicates the amount of data transferred per time unit in bits per second. The baud rate is the number of “symbols” per time unit.

Common bit rates include 75, 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600 and 115200 bit/s. [15]

Data bits

The number of data bits can vary between 5,6,7,8 and 9 for each character, but 8 data bits are almost always used as it corresponds to the size of a byte.

Parity

Parity is a way to prevent errors in the data transfer. It ensures the sum of the “1” bits comes always either odd or even. If the sum is not as expected, it means the data was corrupted.

Stop bits

Stop bits are sent at the end of every character to easily detect the end of a character. Normally one stop bit is used.

3. Measurement devices and Setup

3.1 Measurement devices

Localy available sensors

In **Fig.7** and **Fig.8** are the sensors that are now in use by the weather station.

Windrichtung

Sensor:	opto-electronic scanning, 6 bit Gray-code, axis heating below 4 ° C
Manufacturer:	Vaisala (Finnland)
Type:	WAV 151
Resolution:	5,6°
Accuracy:	5,6°

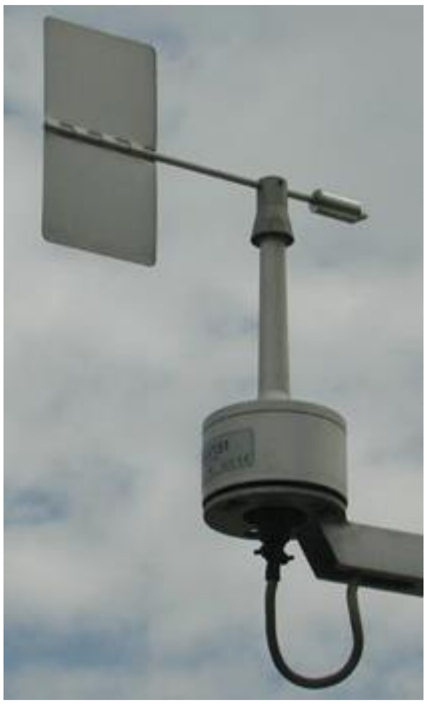


Figure 5 Wind vane [7]

Windgeschwindigkeit

Sensor:	Three wings cup anemometer, digital sampling (14 pulses / revolution)
Manufacturer:	Vaisala (Finnland)
Range:	0...75 m/s; axis heating below 4 ° C

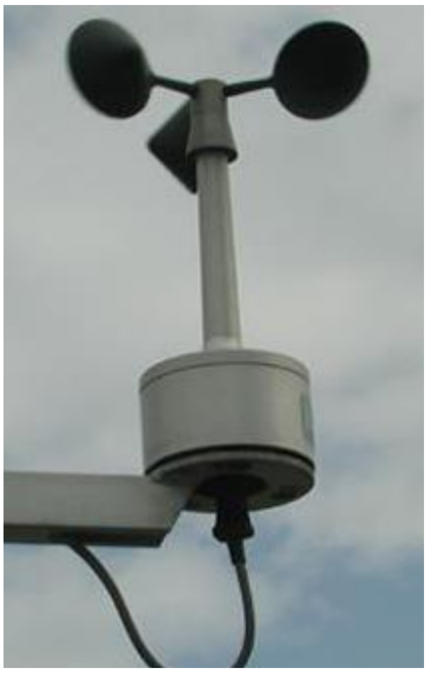
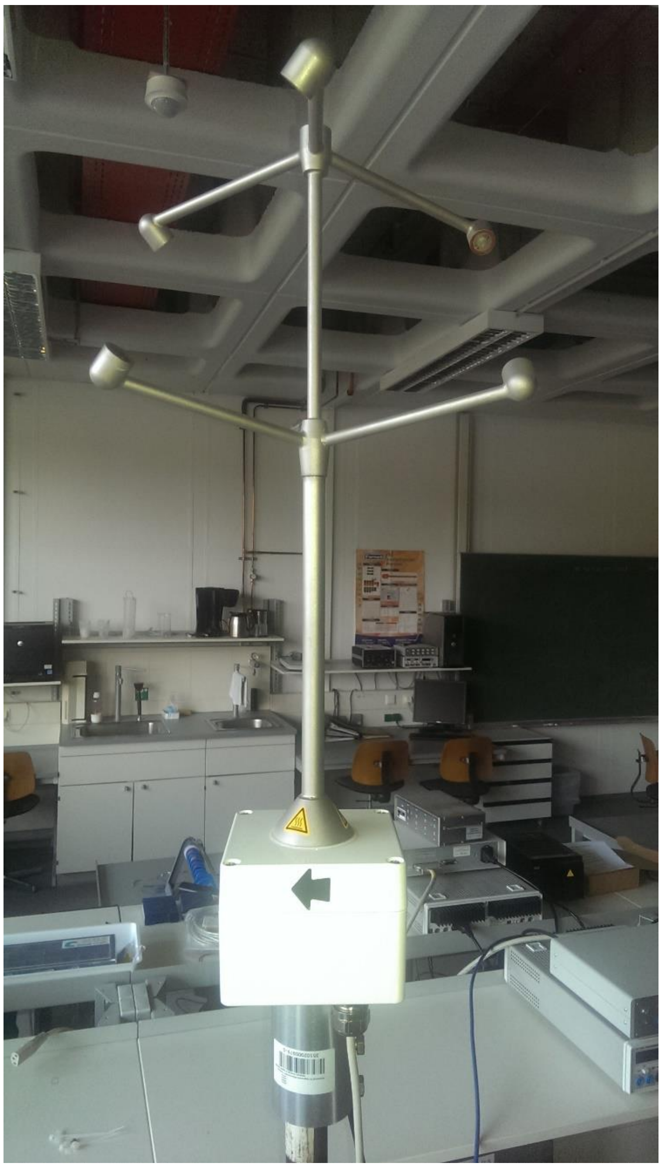


Figure 6 Three cup anemometer [8]

New Sonic Anemometer

Sensor:	Sonic anemometer, digital sampling 10Hz
Manufacturer:	Metek (Meteorologische Messtechnik GmbH)
Range:	-50…+50 m/s; -30 ° C…+50 ° C

For a more detailed view of all technical data regarding the Sonic anemometer, see Atachments



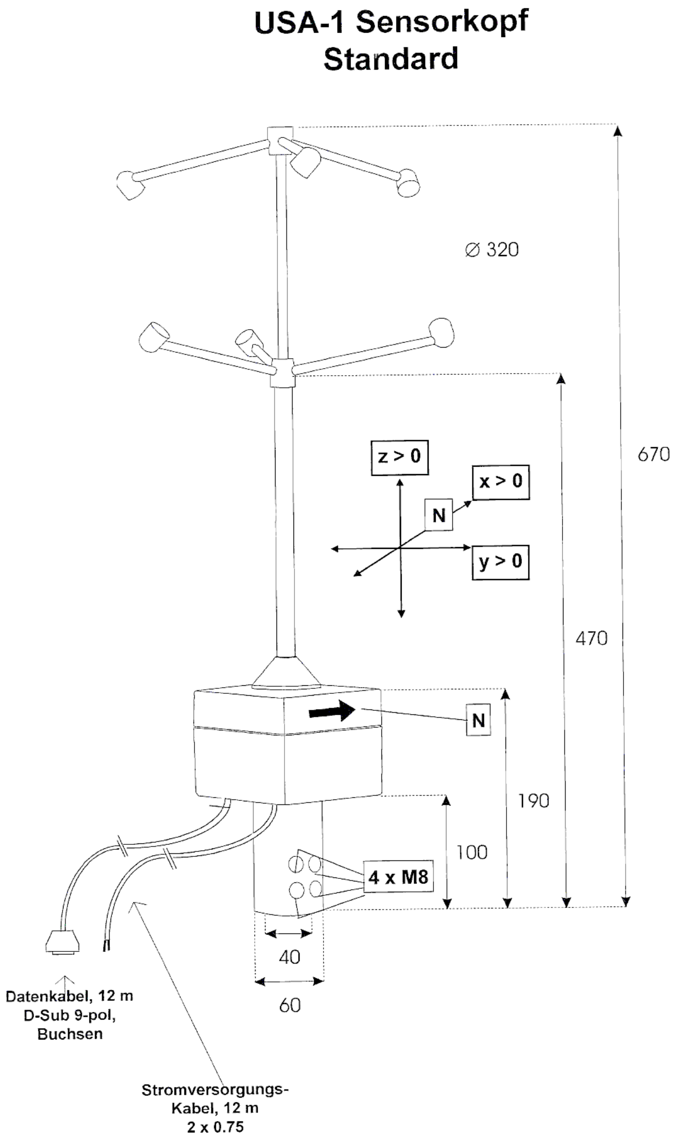


Figure 7 the USA-1 and its schematics [9]

3.2 Setup

The Sonic anemometer was first mounted inside the laboratory to facilitate its configuration. In **Fig.9** can be seen, the Sensor (1) and the electronics box (2) were mounted on a big metal rod that is strongly held by zip ties attached to the table and a bit of wood at the bottom as not to damage the floor (4). The device takes only DC power, so it was attached to a 24V DC power converter (3). The serial communications port is connected to the computer where the development will take place (5).

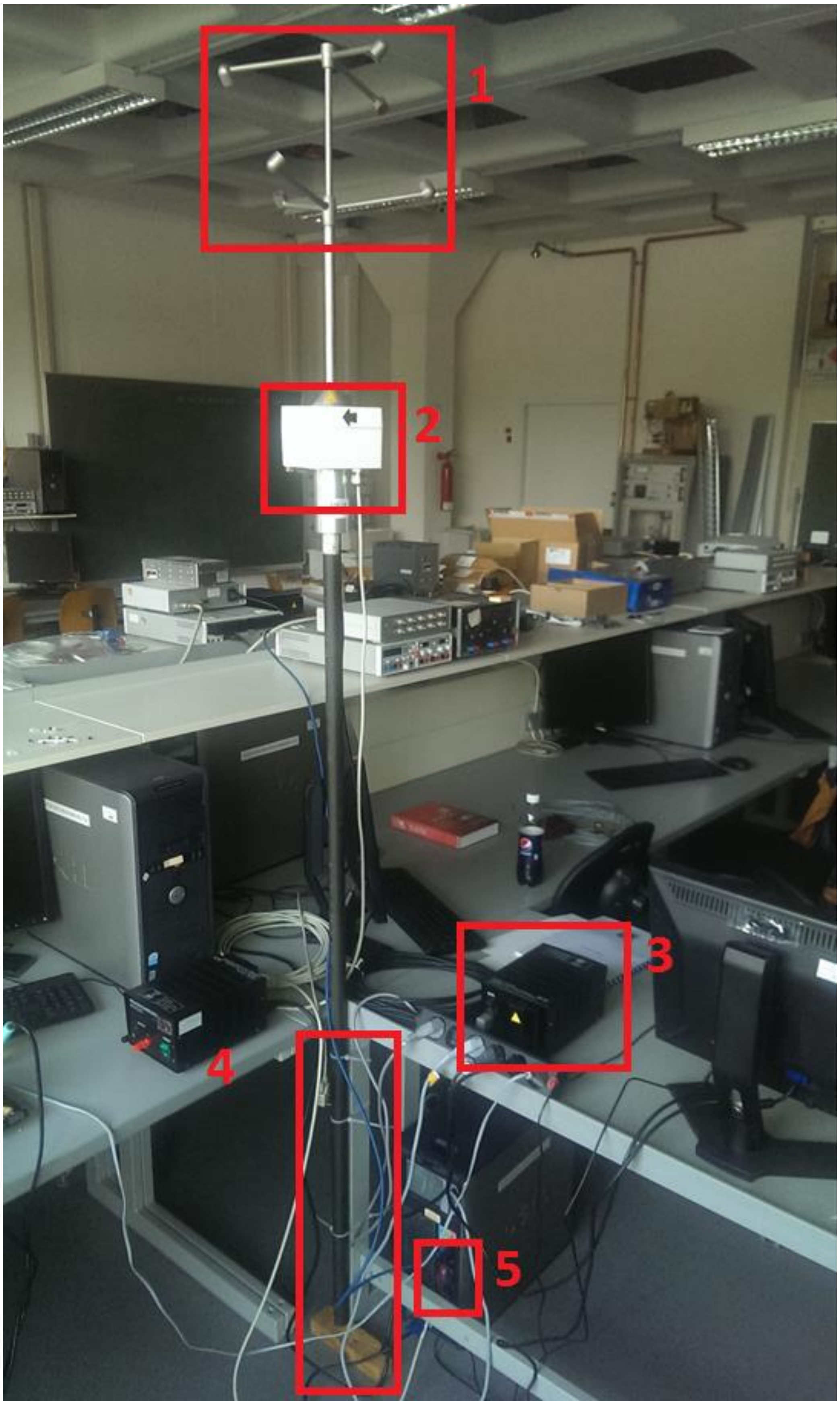


Figure 8 Anemometer setup

To begin with the project a computer was provided to install a Linux version similar to the one the weather station uses. The weather stations own environment can’t be used as it is in Production, a test environment is always needed for development, assuring no damage or downtime can affect the main Server during the development phase.

After the Linux installation an IDE was needed. The IDE eclipse was strongly recommended to be used as it was the one used for the WS. A special version dedicated to “C” development was downloaded and installed.

A MYSQL data bank server and client had to be installed on the same machine as well to assure functionality and make testing easier and cleaner

OS:	Ubuntu 14.04.1 LTS
IDE:	Eclipse IDE for C/C++ Developers
Databank:	MYSQL

4. Basic Development

No program can be made perfect on the first try, how a program looks at the beginning is nothing like what it looks like at the end. For this reason it will not be attempted to program perfection from the beginning but just a “Proof of concept”, just to try to make it work.

This is the hardest part as it involves a lot of reading and research as well as trying out many different codes.

For the basic development four milestones were set:

- Being able to read from the Serial Communications Port.

- Transform the values from X and Y into numbers.

- Use X and Y to calculate the horizontal wind speed.

- Save the wind speed value into the MYSQL data bank.

At this point, Understanding how everything works is not as important as making it work.

4.1 Reading

Goal: Being able to read from the Serial Communications Port.

To be able to read from the Serial communications device the port has to be opened first. This can be accomplished though the following command:

```
tty_fd=open("/dev/ttyS0", O_RDWR| O_NONBLOCK );
```

The main command here is the function **open**() in which is specified which file (or in this case device) it wants to read the data from and the attributes we want it to have.

In this case “/dev/ttyS0” is the name of the Serial communications device, “O_RDWR| O_NONBLOCK” are the attributes. What exactly this attributes do, can be found on the “C++ von A bis Z” [17]

Now that the port is open, the program needs to read the information being sent. For that the **read**() function will be used:

```
read(tty_fd,&c,1)
```

Where “&c” is the address of the variable “c” which will be used as buffer and is where the characters read from the device will be saved, the “1” is the number of characters that will be read from the device at a time and “tty_td” is again the variable for the device.

Finally this read function will be placed in an infinite loop, so the program can read not only one character but all the information sent by the device.

To be able to see the information being read the **printf**() function is used.

Result:

```
0000000000000000000000000000000000000000000000000000000000000000000000
```

Reading Failure!

The device sends information only once every 10 seconds while the program reads the information many times per second. This means the **printf**() function will write the standard value “0” many times before the device has the chance to send information. This is what caused the first attempt to fail.

To address this issue the read command was modified:

```
if (read(tty_fd,&c,1)>0)
{
        printf("%c",c);
}
```

The “read()” function returns the number of bytes read. If there is nothing to read 0 is returned, on error it returns -1.

Setting **if** (**read**()>0) ensures the function **printf**() will only run when information is successfully read from the device

Result:

```
M:x =    11 y =    -9 z =    -7 t =  2412

M:x =    11 y =    -9 z =    -7 t =  2409

M:x =    11 y =   -10 z =    -6 t =  2407
```

Reading Success!

Now the format in which the information is delivered is known. Every line is sent every 10 seconds.

The reading milestone was reached.

4.2 Converting

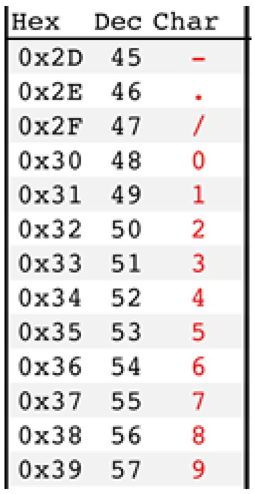
Goal: Transform the values from X and Y into numbers.

Information the program gets from the reading program is a stream of characters, for example:

```
M:x =    16 y =    -9 z =    -7 t =  2412
```

The program will now try to extract the value “16” that corresponds to the variable x.

The problem is that it doesn’t have the number 16 but the character “1” followed by the character “6”.



Hex	Dec	Char
0x2D	45	-
0x2E	46	.
0x2F	47	/
0x30	48	0
0x31	49	1
0x32	50	2
0x33	51	3
0x34	52	4
0x35	53	5
0x36	54	6
0x37	55	7
0x38	56	8
0x39	57	9

Table 1 Cut from ASCII table [14]

As seen on **tab.1** the character “1” would actually have a decimal value of 49. It also has to take into consideration that the “empty space” is actually a character as well.

X-=----16

To extract only the character that needs to be transformed into numbers, the program could look for the variable “x” in the stream, skip the next six characters and save the next two characters into a variable or array of its choosing.

Which brings out the second problem, The length of the number is not known. From the Sonic Anemometer's User Manual it can be learnt that the value range from x, y and z lies between 5000 cm/s to -5000 cm/s

This means that the length of the relevant characters needed can vary between 1 and 5 including the “-” character of the negative numbers

First of all the program will start by finding the character x in our stream. This is easily accomplished by the following command:

```
if (c=='x')
```

Where “c” is the variable where the reading program saves the characters.

As the length of the number is not known, the program needs to skip any characters that aren't “0-9” or the character “-”.

```
while (c<45 || c>58)
{
        read(tty_fd,&c,1);
}
```

As seen on **tab.1** the decimal values of “0-9” or the character “-” are all in between 45 and 58. This is why this loop was set to run while the values of “c” are either less than 45 or bigger than 58. This loop will run until the **read**() function returns a value of between 45 and 58. The values 46 and 47 weren't excluded as the characters "." and "/" don't appear anywhere on the stream.

Next it will find out if it's a negative number with the following code:

```
minus= 1;
if (c == 45)
{
        minus=-1;
        read(tty_fd,&c,1);
}
```

First it will set the variable "minus" to have a value of "1". If the decimal value of the first character is 45, it means it's a negative number, then it will make one more read to get the next character.

Now it is going to start reading the numbers and we need to start converting them and saving them into an array.

```
while (c>=48 && c<=58)
{
        i = c-48;
        Numbers[counter]=i;
        read(tty_fd,&c,1);
        counter++;
}
```

This function will only run as long as it gets decimal values between 48 and 58 which corresponds only to the numbers "0-9". After a lot of research was concluded that the easiest way to convert the characters into numbers was to simply subtract 48 from their decimal value.

Example:

character “0”	decimal value 48	48 – 48 = 0
character “5”	decimal value 53	53 – 48 = 5
character “9”	decimal value 58	58 – 48 = 9

The new numeric value will be saved on the variable “i” and into the Numbers[] array. The variable counter not only helps to save the values in different places of the array, it also tells how long the number is.

It has now converted the characters into numbers but they are still separate numbers and because the characters are read from left to right the formula to add them together varies depending on the length of the original number.

```
switch(counter)
{
case 1:
        Final = minus*Numbers[0];
        break;
case 2:
        Final = minus*(Numbers[0]*10+Numbers[1]);
        break;
case 3:
        Final = minus*(Numbers[0]*100+Numbers[1]*10+Numbers[2]);
        break;
case 4:
        Final = minus*(Numbers[0]*1000+Numbers[1]*100+Numbers[2]*10+Numbers[3]);
        break;
default:
        break;
}
```

As stated before, the variable counter tells the length of the original number. In case the length of the original number is “1”, the final number is the first number on the array (Numbers[0]) times the “minus” variable which can only take the values “1” or “-1”. In case the length is “2”, “3” or “4” the different formulas would have to be applied respectively.

It can’t be forgotten to reset the counter and the array at the end.

```
memset(Numbers, 0,4);
counter =0;
```

The function **memset**() is ideal to reset an array. The first parameter is the array to reset, the second is the value to write into the array and the third one specifies how many are to be written.

Now to see if the conversion worked with the **printf**() function.

```
printf("Converted X = %d, ",Final);
```

Result:

```
M:xConverted X = 13, y =    -2, z =    -2 t =  2462

M:xConverted X = 14, y =    -1, z =    -1 t =  2462

M:xConverted X = 13, y =    -4, z =    -7 t =  2458
```

It really is difficult to see any difference between the characters and the real numbers. The only difference is that real numbers can be used for mathematical operations. So to test if the conversion really worked the following lined were added:

```
x = Final + 23 ;
printf("Sum = %d, ",x);
```

Result:

```
M:xConverted X = 17, Sum = 40, y =    -2, z =    -2 t =  2462

M:xConverted X = 18, Sum = 41, y =    -1, z =    -1 t =  2462

M:xConverted X = 18, Sum = 41, y =    -4, z =    -7 t =  2458
```

A basic addition was added, just added 23 to the final number and it can be clearly seen that it worked. Now the code has to be copied and applied to “y”.

Result:

```
M:xConverted X = 6, Sum = 29, yConverted Y = -13, Sum = 10, z =    -8 t =  2425

M:xConverted X = 6, Sum = 29, yConverted Y = -12, Sum = 11, z =   -10 t =  2427

M:xConverted X = 4, Sum = 27, yConverted Y = -13, Sum = 10, z =   -12 t =  2432
```

Reading Success!

The converting milestone was reached. The test to see if it worked will be removed from the code as the original values are needed to begin the next step.

4.3 Processing

Goal: Use X and Y to calculate the horizontal wind speed.

To be able to use mathematical functions, a new library has to be added to the repository.

```
#include <math.h>
```

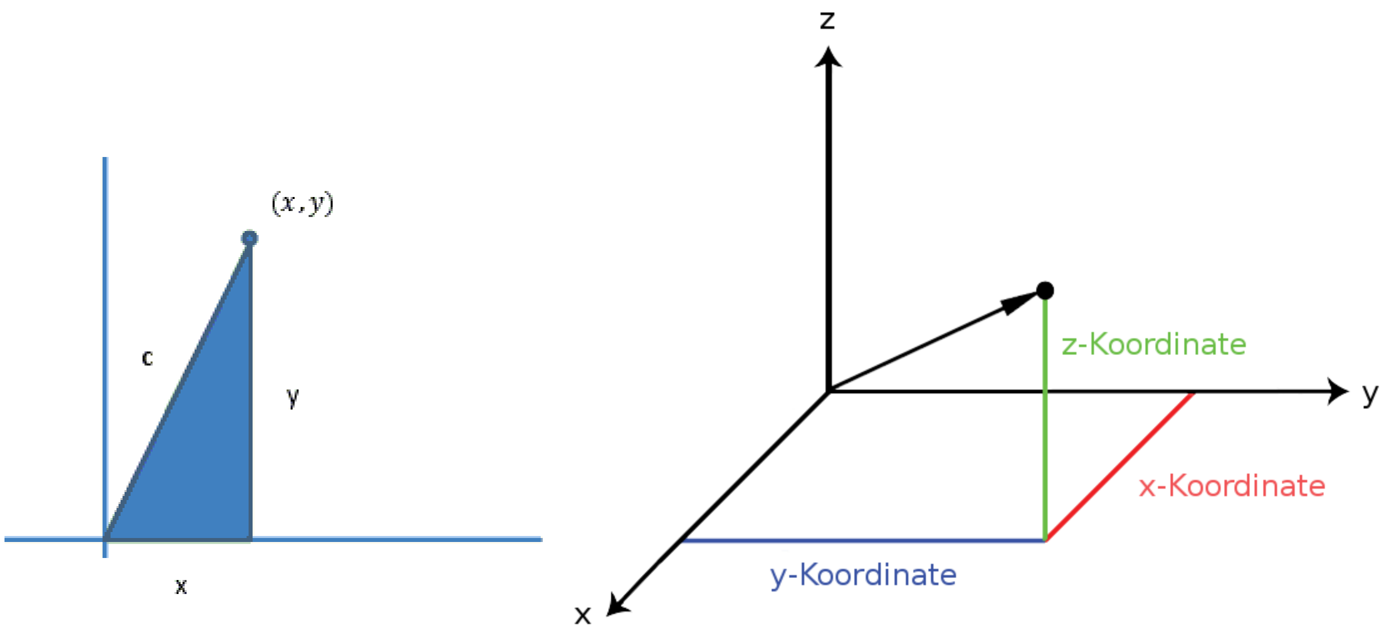


Figure 9 2D and 3D vectors [11] [12]

To calculate the Horizontal velocity the values from X and Y are needed. As X and Y stand in a right angle to one another, the Pythagorean Theorem needs to be applied.

```
hv = sqrt(x*x+y*y);
```

Result:

```
M:xConverted X = 11, yConverted Y = -13, hv = 17.029387

M:xConverted X = 12, yConverted Y = -12, hv = 16.970562

M:xConverted X = 15, yConverted Y = -11, hv = 18.601076
```

The processing milestone was reached.

4.4 Save to database

The database that will be used to store the information is a MYSQL database.

```
#include <mysql/mysql.h>
```

First initialized a MYSQL object.

```
MYSQL *con = mysql_init(NULL);
```

Using the Address, username, password and database name used to create the database, a connection to the database was established.

```
mysql_real_connect(con, "localhost", "root", "Ultraschall1","Wind", 0, NULL, 0);
```

The function **mysql_query**() can only accept two arguments, the MYSQL object and a string. As the string used had too many arguments to procces, **sprint**() was used to save the arguments into the string "query". In this case "WindTable" is the name that the table where we store the data within the database was given.

```
sprintf(query,"INSERT INTO WindTable (X,Y,Hv) VALUES(%d,%d,%f)",x,y,hv);

mysql_query(con,query)
```

At the end the connection can be closed again using:

```
mysql_close(con);
```

The database milestone was reached.

With the four milestones reached the proof of concept was a success. Now the full development of the program can begin.

5. Advanced Development

Now that the basic development proved the project to be possible, it is the time to improve upon it and write a complete and reliable program. For this, every bit of code has to be 100% understood.

As the Program grows in size, it becomes more difficult to understand and edit. This is why functions will be used whenever possible from now on.

5.1 Reading

On chapter 4.1 it was proved that the program can get information from the serial port by using a non-blocking read() function that retrieves one character at a time and putting it in an endless loop. That approach was fine for a proof of concept program, but putting non-blocking function in an endless loop would saturate any processor no matter how fast it is. Having a faster computer would only mean the loop would run more times per second but the processor would still get saturated. This is why the way data was being read had to edited and for this a “termios structure” found within the termios.h library was used.

```
#include <termios.h>
struct termios stdio;
```

Within the termios structure what is important to the program are mainly the "VMIN" and "VTIME" attributes. As can be seen on the termios man page[19] giving VMIN and VTIME a value would result in a blocking signal which would keep reading until the VMIN number of characters is reached or until the VTIME number in tenths of seconds between characters has passed.

```
        stdio.c_cc[VMIN]=50;
        stdio.c_cc[VTIME]=5;
```

From Capter 4 was learned that the string received always is 43 characters long. VMIN was set to 50 so the deciding factor between reads would be the VTIME which was set to 0.5 seconds.

With these attributes set, the "O_NONBLOCK" option was removed. If it is not removed, the VMIN and VTIME attributes will be ignored.

```
        tty=open("/dev/ttyS0", O_RDWR);                //| O_NONBLOCK
```

The Baud speed was set at 9600 baud to match the settings of the device.

```
        cfsetospeed(&stdio,B9600);
        cfsetispeed(&stdio,B9600);
```

The **tcsetattr**() function was used to update the communications with the "tty" device with the settings written in the "stdio" termios structure.

```
        tcsetattr(tty,TCSANOW,&stdio);
```

Finally the **read**() function was used again, but this time "M" is a character array capable of storing all 43 characters at once and the maximum characters allowed to return were changed from 1 to 50.

```
Char N[50];
read(tty,N,50);
```

5.2 Converting

Given that the information the program gets from the **read**() function are no longer single characters but a whole string with 43 characters, it is now much easier to convert the numbers stored within.

The parts of the string where the values are were copied and stored into different strings with the **memcpy**() function.

```
memcpy( Xstring, &N[6], 5 );
memcpy( Ystring, &N[16], 5 );
memcpy( Zstring, &N[26], 5 );
memcpy( Tstring, &N[36], 5 );
```

Where the first argument is the new string, the second argument is the place in the original string where we will start to copy and the third argument is the number of characters that will be copied.

The character “/0” was added at the end to indicate it is a string and not just an array of characters.

```
Xstring[5] = '\0';
Ystring[5] = '\0';
Zstring[5] = '\0';
Tstring[5] = '\0';
```

Finally the **atoi**() function was used to transform the strings into integers.

```
x = atoi(Xstring);
y = atoi(Ystring);
z = atoi(Zstring);
t = atoi(Tstring);
```

5.3 Processing

Besides the horizontal velocity we calculated in Chapter 4, the program also need to calculate the total velocity, the horizontal wind direction and vertical wind direction

5.3.1 Total Velocity

The total velocity can be calculated just like the horizontal velocity using Pythagoras theorem. This is because the Z component is at a right angle with the horizontal velocity

```
tv = sqrt(x*x+y*y+z*z);
```

5.3.2 Horizontal Wind Direction

The horizontal wind direccion can be obtained using the inverse tangent function of the components X and Y. I used the **atan2**() function instead of **atan**() because **atan**() can only process one value, making it imposible to determine wich quadrant came the wind from.

Example:
X and -Y would yield the same result as -X and Y
X and Y would yield the same result as -X and –Y

Apart from being able to give both X and Y as attributes, **atan2**() also sets the X coordinate as “0”. This is convenient given that our device marks the X coordinate as “North”

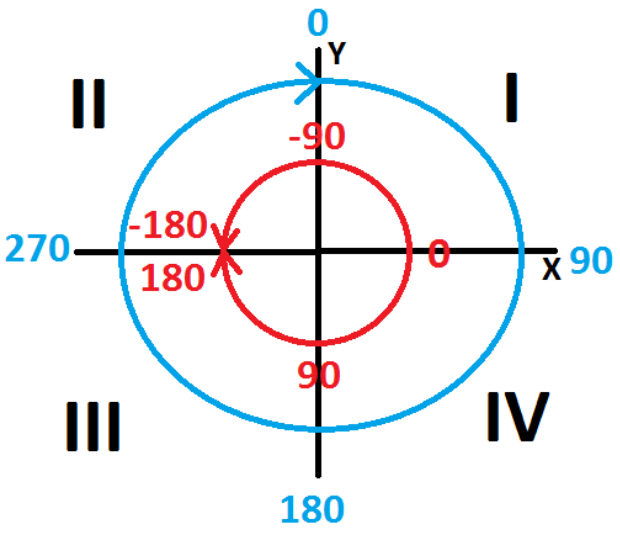


Figure 10 Differences between atan and atan2 [13]

The results yielded by **atan2**() are given in radians, which is why PI was defined and converted it to degrees.

#define PI 3.14159265

hd = **atan2**(x,y)* 180 / PI;

As seen in **Fig.12** **atan2**() doesn't deliver angles from 0 to 360 but angles from 0 to 180 and from 0 to -180 degrees. This can be easily fixed as shown below.

if (hd < 0) hd= hd + 360;

If hd is a negative number, 360 is added to its value.

5.3.3 Vertical wind direction

To calculate the vertical wind direction the **atan2**() was used function again, but this time using Z and the horizontal velocity. As the value of the horizontal velocity will never be negative, the result will always be in the 0 to 180 range. For this reason a value of 90 was substracted, so the new range will vary between -90 and 90 degrees with 0 being the horizontal.

```
zd = ((atan2(hv,z)* 180 / PI)-90)*(-1);
```

5.3.4 Time

For a better statistical analysis I was asked to get the mean values of all the data I gather. This also helps not to flood the database with useless information. The intervals in which the data has to be gathered are every minute and every ten minutes. I was also asked to get the Minimal and Maximal values for the horizontal velocity and the total velocity within these intervals in time.

For this endeavor the program has now to be able to track time.

```
#include <time.h>
```

I created a function to help me get the actual time and set the timer for the one minute and the ten minutes mark.

```
unsigned long GetTime()
{
        time_t rawtime;
        rawtime = time(NULL);
        if (OneMinTime == 0) OneMinTime = rawtime;
        if (TenMinTime == 0) TenMinTime = rawtime;
        return(rawtime);
}
```

The function creates a “time_t” object “rawtime”, gets the UNIX time[20] and stores it into “rawtime”. This function returns “rawtime” as an unsigned long. If it is the first time this function has been called it will store “rawtime” into the “OneMinTime” and “TenMinTime” variables.

I store the return value of **GetTime**() into the variable “timer” and then set a condition that will only run when the actual time “timer” minus the time that was stored when the function ran for the first time “OneMinTime” is equal or more than 60 seconds.

```
Timer = GetTime();

if (timer - OneMinTime >= 60)
```

The same was done for the “TenMinTime” variable with 600 seconds.

5.3.5 Minimal and Maximal values

For the minimal and maximal values I created another function. This function will first check if the value of “hV_minOne” equals -1. In this case, all the minimal and maximal values of “hv” and “tv” will be replaced with the actual values of “hv” and “tv” respectively.

```
if (hv_minOne == -1)
{
        hv_minOne = hv;
        hv_maxOne = hv;
        tv_minOne = tv;
        tv_maxOne = tv;
}
```

It is imposible for the horizontal velocity to have a negative value on its own. I set it to -1 so I can reset the minimal and maximal values after each minute.

After the function sees if the values have to be reset, it proceeds to check if the actual values are bigger than the Maximal or smaller than the minimal and replace them accordingly.

```
if (hv < hv_minOne) hv_minOne = hv;
if (hv > hv_maxOne) hv_maxOne = hv;
if (tv < tv_minOne) tv_minOne = tv;
if (tv > tv_maxOne) tv_maxOne = tv;
```

5.3.6 Mean value

For the mean value I decided it would be easier to get the mean values of the raw data and then calculate the rest using those mean values. I decided to do things this way as it would be very difficult to get the mean value of the wind direction.

Example:
0 and 360 represent the same direction and the mean value 180 represents the opposite.

Normaly the program would get the same number of data strings every minute, but if for some reason it gets less or more than expected, the data would get corrupted. For this reason I created an array to store the sum of the raw data and set a counter to measure how many times the data was added.

```
SumdataOne [0] += xRaw;
SumdataOne [1] += yRaw;
SumdataOne [2] += zRaw;
SumdataOne [3] += tRaw;
CounterOne++;
```

Once the time is up, the mean is calculated by just dividing the sum of the gathered data by the counter

```
xOne = SumdataOne [0] / CounterOne ;
yOne = SumdataOne [1] / CounterOne ;
zOne = SumdataOne [2] / CounterOne ;
tOne = SumdataOne [3] / CounterOne ;
```

Finally it is very important to reset the counter, “hv_minOne” value and the timer

```
CounterOne = 0;
hv_minOne = -1;
OneMinTime = timer;
```

5.4 Save to database

I was asked to create two different tables for the one minute data “1minTable” and for the ten minutes data “10minTable”. The script used to create these tables can be found in Atachment. This new tables now have to store twelve different figures, not only the three it had in Chapter 4

To differentiate between the one minute and ten minutes data, I added a simple argument “int a” to some of my functions. If “a” equals 1 the functions follow the parameters needed for the one minute data. If it’s anything else it follows the parameters needed for the ten minute data.

```
if (a==1)
{
sprintf(query,"INSERT INTO 1minTable
(X,Y,Z,Hv,Tv,Hd,Zd,Hv_min,Hv_max,Tv_min,Tv_max,Temp)"
"VALUES(%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f)"
,xOne,yOne,zOne,hv,tv,hd,zd,hv_minOne,hv_maxOne,tv_minOne,tv_maxOne,tOne);
}
else{…}
```

With this, the Main function the program is done.

6 Testing

Now that the main functions of the program have been completed, is now time to start testing it. The program might work under ideal conditions but in the real world, conditions are never ideal. Now is the time to think about everything that can go wrong and make it happen, or at least try to simulate it.

6.1 Wind test

Now it's time to test if the program really works as intended. For this I'm going to use a fan to simulate outside wind conditions. Sadly the only fans available at first were small computer fans, which were useless as the fan has to be big enough reach all three sensors in the anemometer to accurately represent wind. Later I was allowed to bring in a bigger fan to help with the testing.

Figure 11 Fan used to simulate wind

Test:
Set the fan at different angles to see if all values are represented accurately.

Result:
The raw values X, Y, Z, T were as expected as they weren't modified by the program.
All other values worked just as it was intended.

Observations:

Even though everything worked as intended, it was pointed out to me that when talking about wind "wind direction" refers to the direction the wind is coming from and not where it is going. Because of this the following functions were changed:

From: hd = **atan2**(x,y)* 180 / PI; To: hd = **atan2**(-y,-x)* 180 / PI;

From: zd = ((**atan2**(hv,z)* 180 / PI)-90)*(-1); To: zd = ((**atan2**(hv,z)* 180 / PI)-90);

After these changes, the program showed the correct wind direction.

6.2 Disconection

In the real world the serial cable could get cut, damaged or simply disconnected.

Test:

Disconnect and Connect serial cable multiple times.

Result:

When the cable was disconnected, the program stopped until the cable was connected again. It read the data correctly as if nothing had happened.

Observations:

While it's true it appeared to work perfectly, the fact is the program stops completely until the cable gets connected again. This means the program cannot log errors or warn anyone that a problem has occurred.

6.3 Power failure

In the real world a power failure is always a possibility. While a complete power failure would obviously turn off the computer and the program with it, it is possible in some cases that only the anemometer would be the one who loses power.

Test:
Turn the device off and on multiple times.

Result:
Same as before, the program stopped until the device was turned on again. The program failed converting the first string but then proceeded to process the rest correctly.

Observations:
When its turn on, the Anemometer sends a string with its name and other kind of information.

6.4 String testing

As it was just observed, we might sometimes get strings of different length and content. To simulate this I used the **fgets**() function to save whatever I write on the console into the same array our devices uses and overwrite it before converting it.

```
fgets (N, 80, stdin);
```

Tests:

- Write a string shorter than 43 characters
- Write a string longer than 43 characters
- Write a string exactly 43 characters long but with different content

Result:
In all cases the program failed without clearly reporting what went wrong and sometimes crashing completely.

Observations:
Testing the length and quality of the string before trying to use it would be useful.

7. Fine tuning

The testing showed there is definitely room for improvement. Some details need to be polished to make the program more reliable and harder to crash. The “errno.h” library was added to help identify different kinds of errors.

```
#include <errno.h>
```

7.1 Reading

Reading had to be revisited. It worked well but it would be a lot better if the program wouldn’t stop completely until it gets the next signal. For this I revisited the options available for VMIN and VTIME.

```
stdio.c_cc[VMIN]=0;
stdio.c_cc[VTIME]=50;
```

Setting VMIN to 0 changes the way VTIME behaves. It is no longer the time it waits between characters, it is now a timer. If 5 seconds pass since the time it tried to read but no information comes, the timer expires and ends the **read**() function with a return value of 0.

The only unexpected development is that the string is no longer read as a whole but as five bursts of 8 characters and a final burst of 3 characters. A condition was added to keep the loop going until the string reaches a length of 43.

```
if (strlen(N) >= 43)
```

If the **read**() function returns a value bigger than 0, it means something was read and successfully saved into the “M” array. The return value of **read**() is now saved under the “s” variable.

The function **strcat** () is used to take the 8 character string “M” and add it to the end on “N” until “N” reaches a size of 43.

```
s = read(tty,M,100);
if (s>0) {
        strcat (N,M);
        if (flag3==1)
        {
                printf ( "Connection regained at: %s\r\n", Timestr );
        }
        flag3=0;
} else if (s==0) {
        if (flag3==0) {
                printf("\r\nTimeout! Read exceeded 5 seconds!\r\ ");
                printf ( "Connection lost at: %s\r\n", Timestr );
        }
        flag3=1;
}
```

If the timeout expires and **read**() returns 0, a message explaining the error and the time the connection was lost are displayed. The variable “flag3” is simply used so the program doesn’t spam the same error message every five seconds if the cable is disconnected. It also helps to mark the time when the connection was regained.
If an error occurs in the **read**() function, the value returned will be -1. If this happens there

is no point trying to keep the program running, the program will just simply print out the error with the help of “errno” and exit.

```
else
{
        printf("Error with read(): %s\r\n", strerror(errno));
        exit(0);
}
```

7.2 Converting

While there was no problem in the way the program was converting the string into integers, but it was pointed out to me that it could be done a bit more elegantly using the function **sscanf**(). With it, I was able to directly extract all four integer values from the string at once.

```
sscanf(N, "M:x =%d y =%d z =%d t =%d ", &xRaw, &yRaw, &zRaw, &tRaw);
```

7.3 String Testing

To test the string I created two functions, one to test the string length and another to test the string quality. Both functions would return an integer value of one if successful and a value of 0 if the test failed.

The length function would take the “N” string and only return successful if the string had a length of 43.

```
int LengthTest(char *N)
{
        if (strlen(N)==43)
        {
                printf("String length Correct");
                return(1);
        }
        else
        {
                printf("String is too long");
                return(0);
        }
}
```

The string can only be either correct or too long given that the **read**() function will continue until the string “N” reaches a length of at least 43 characters.

For the quality test the program has to make sure that the string is not corrupted or damaged in any way. It is known exactly how a correct string looks like, this is why the program can simply look for characters in places where it knows they are going to be.

```
int QualityTest(char *N)
{
        if (N[0] == 'M' && N[2] == 'x' && N[12] == 'y' && N[22] == 'z' && N[32] == 't')
        {
                printf("Stringstatus: OK\r\n");
                return(1);
        }
        else
        {
                printf("Stringstatus: Damaged\r\n");
                return(0);
        }
}
```

In this test, the function looks for “M”, “x”, “y”, “z” and “t” at the correct places in the “N” string and it only returns a value of 1 if all of them are correct.

Both test functions are called and their return value stored into q1 and q2.

```
q1=LengthTest(N);
q2=QualityTest(N);
```

In case the tests were successful and q1 and q2 are equal to 1, the rest of the code is allowed to run.

```
if (q1 == 1 && q2 == 1)
{
        …
}
else
{
        sleep(1);
        s = read(tty,N,200);
}
```

Otherwise the program will wait for one second allowing any incomplete parts of the string to arrive and then proceeds to read up to 200 characters form the serial communications buffer and delete them allowing the program to resynchronize itself with the new incoming strings.

7.4 Mysql error Proofing

There is a lot that can go wrong while trying to connect to the database. Thankfully MYSQL has its own error reporting function **mysql_error**().

Any error with the database is considered a critical error as it makes no sense to continue if the connection with the database cannot be stablished. I made a simple function that just prints the error, closes the connection and exits the program.

```
void finish_with_error(MYSQL *con)
{
        printf("%s\n", mysql_error(con));
        mysql_close(con);
        exit(0);
}
```

8 Additional Features (out of scope)

While the main function of the program has been improved, tested and protected against most errors, there are still many ways in which the program could still be improved. These features were not part of the original assessment of the program but it is always the case that new ideas and indispensable features are discovered during development.

8.1 Command line arguments

It would be very useful if the program was able to use command line arguments in case another serial port or another database needs to be used, without the need no rewrite or recompile the program.

First the main function needs to be able to read command line arguments.

```
int main(int argc,char** argv)
```

Then the program needs a new library to be able to process these arguments.

```
#include <getopt.h>
```

The function **getopt**() needs to be in a loop as it processes all the arguments one by one and stores them into the variable "c"

```
while ((c = getopt (argc, argv, "Dd:n:h:u:p:L:")) != -1)
```

The function **getopt**() will search for the letters "Dd:n:h:u:p:L:" and return "-1" when there are no more arguments left. In the case of "D" **getopt**() will just search for the letter, but if the letter is followed by ":" it means that after the letter a string is expected and it will be saved under the string "optarg".

A switch case for “c” is needed to separate each of the possible arguments.

```
switch (c)
{
        case 'D':
                Dflag = 1;
                break;
        case 'd':
                dvalue = optarg;
                m1 = 1;
                break;
        …
        …
```

In the case of “D” the program just sets a flag to mark if “D” was written or not but in the case of “d:” the string gets saved into “dvalue”. I also set a flag “m1” on the fields I deemed mandatory to start the program.

If **getopt**() retrieves an argument that wasn’t expected, the “?” case gets called.

```
case '?':
        if (optopt == 'd' || optopt == 'n' || optopt == 'h' || optopt == 'u'
                || optopt == 'p' || optopt == 'L')
                printf ("Option -%c requires an argument.\n", optopt);
        else if (isprint(optopt))
                printf ("Unknown option `-%c'.\n", optopt);
        else
                printf ("Unknown option character `\\x%x'.\n",optopt);
        break;
```

If optopt has one of the letters that were expected it means the correct letter was written but without a string afterwards. If optopt is none of the expected letters, the program tests if its one of the standard printable letters with isprint(). If it is, it just prints the letter out but if it is one of the non-standard letters like the german letter “ä” for example, it prints out the character code.

The program was set to show all argument options if no arguments or incomplete arguments were given. The program would then check for all mandatory fields and if one was missing, the program would just exit.

```
if (m1 != 1 || m2 != 1 || m3 != 1 || m4 != 1 || m5 != 1)
{
        printf ("****Please fill in all mandatory fields****\r\n");
        exit(0);
}
```

8.2 Run as deamon

It would be best if the program could just run in the background without the need of any interaction whatsoever. This means running as a service or in linux is more commonly known as “deamon”.

To make a program run as deamon is a little complicated, first the program has to use **fork**(), which creates another process. This new process is known as a child process and runs at the same time as the original process. The only difference between them is the process ID. The child process has a process ID of 0.

First I created two “pid_t” objects. One for the process ID and another for the new session ID.

```
pid_t pid                    //Our process and Session ID
pid_t sid;
```

The program runs **fork**() and stores its process ID on the “pid_t” object “pid”. At this point in time there are two processes running the same code in the program at the same time.

```
pid = fork();
```

In case “sid” had a value of “-1”, it means the **fork**() failed and the program should just end.

```
if (pid < 0)
{
        exit(EXIT_FAILURE);
}
```

If “pid” has a value bigger than 0, it means it’s the parent process and it should just end as well.

```
 if (pid > 0)
{
        exit(EXIT_SUCCESS);
}
```

At this point only the child process should still be running. First the child process has to change the file mode mask it inherited from its parent.

```
umask(0);
```

Now a new session ID is given to the child process.

```
sid = setsid();
```

Again, if the “sid” returns with a value of “-1”, it means **setsid**() failed and the program needs to exit.

```
if (sid < 0)
{
        exit(EXIT_FAILURE);
}
```

As the program might be run in different Linux systems with different folder structures, it is best to change the working directory to root “/”.

```
chdir("/")
```

Finally the program need to close the standard file descriptors, so it cant write or read any information from “stdout”, “stdin” and “stderr”.

```
close(STDIN_FILENO);
close(STDOUT_FILENO);
close(STDERR_FILENO);
```

8.3 Error logging

If the program is going to run in the background, in the case it fails, it would be impossible to know what went wrong with it. For this an error log should be implemented. I created a function to both print the information and write it to a file.

```
void WriteLog (char *message)
```

First I created a “FILE” object

```
    FILE *log;
```

Then I used the function **fopen**() with the name of the log file I want to create as parameter. The second parameter is “a” which means it will create the file if it doesn’t exist and if it exists, it will just write new data at the end of the file without overwriting anything.

```
    log = fopen("Wind.log", "a");
```

If it fails it will print out the error, close the file and exit.

```
    if (log == NULL)
    {
        printf( "Write Log: %s\r\n",strerror(errno));
        fclose(log);
        exit (EXIT_FAILURE);
    }
```

If it succeeds it will write in the message and then close the file again.

```
    else
    {
        fputs(message, log);
        fputs("\n", log);
        fclose(log);
    }
```

8.4 Device Configuration

The Anemometer originally sent a string every ten seconds. I was told it would be best if it was configured to send data every second. For this I wrote a little segment that had to be removed from the final program as it uses a blocking user input.

```
fgets (N, 80, stdin);

while (N[0] == 'w')   //revierte SY!
{
        memset(F, 0, 10);
        printf("write to anemometer:");
        fgets (F, 10, stdin);
        write(tty_fd,F,4);
        if (read(tty_fd,M,80)>0)
        {
                printf("%s\r\n",M);
        }
}
```

During the first user input “w” has to be pressed to enter the loop. It is an endless loop that would prevent the rest of the program to run. It is very important that the **write**() function has the exact number of characters we intend to write (in this case “4”) otherwise the anemometer won’t recognize the command.

9. Conclusion and Recommendations

The weather station on campus Bergedorf will now be able to improve its measurements thanks to the new Sonic Anemometer that has been configured and has a new software to manage its output.

The Sonic Anemometer is a great improvement over the three cup anemometer and wind vane originally installed on the weather station. The sonic anemometer has a higher accuracy as it retrieves information 10 times every second and automatically sends out the mean values. The lack of moving parts on the device makes it possible to adapt quickly to changes and the fact that it can measure the vertical wind speed, gives the device an overall better three dimensional wind speed measurement. The device was left running over the entire duration of this project without ever giving any sign of errors. The only time it was turned off was during the testing and then it renewed operation within seconds.

The project itself was a huge undertaking that required a lot of time spent on research. There are many ways to develop a program with the same functions and no real guidelines to do it. As the software development progresses, the code has to be improved, deleted and overwritten over and over again. New ideas for improvement appear and new features needed show up that were not part of the original scope but are of critical importance to the project.

There is one thing I couldn’t really explain and it is that the data I gathered with my final “read” function came in in bursts of eight characters at a time. At first I thought the cause was a bad configuration of the serial communications, but after trying out many different combinations of Baud rate, data bits, stop bits and parity and after double checking and triple checking the sonic anemometers user manual, it is my opinion that the bursts of eight characters might be a standard configuration of the device itself.

Overall I am satisfied with the results, the program reads from the serial communications port, extracts the raw data, calculates the wind speed, direction and six other values and stores the data every minute and every ten minutes in two different tables within a MYSQL database reliably. It can also take command line arguments, run as a deamon and log its errors on a text log file.

Due to a lack of time, the “run as deamon” and “log” features were added but not fully tested. Some error detection was written but without further testing I wouldn’t feel safe calling it reliable. It also might be a good idea for the future not only to read but also be able to write to the serial communications port within the same program.

This might be obvious, but it can’t be forgotten that the name of the database as well as its tables, username and password should be changed for security reasons.

If someone were planning to further advance this project or undertake a similar one in software development, it is my recommendation to comment any changes within the code, write the reasons behind the changes and if possible, keep a changelog. It was my mistake not to keep to these good practices and I found myself many times wondering if a piece of code was critical for the program or completely irrelevant.

The only thing left to do is to mount the anemometer on the celling and see how it fares in the real world.

10. Sources

Image Sources

[1] Figure 1 Laminar flow and turbulent flow
Internet source: 28.09.2014. http://cnx.org/resources/0cee14e8cee 601e40e5a0ec12e83454d/graphics12.png

[2] Figure 2 Change in flow based on the Reynolds number
Internet source: 28.09.2014. http://www.flowcontrolnetwork.com/ext /resources/files/assets/uploaded/fc-0307-fb%20figure%205.jpg

[3] Figure 3 sound frequency ranges
Internet source: 28.09.2014. http://upload.wikimedia.org/wikipedia /commons/thumb/7/74/Ultrasound_range_diagram.svg/1280px-Ultrasound_range_diagram.svg.png

[4] Figure 4 Schema of a 3 cup anemometer
Internet source: 28.09.2014. http://www.infoplease.com/images/cig /weather/21fig04.png

[5] Figure 4 Sonic anemometer
Internet source: 28.09.2014. http://www.th-friedrichs.de/assets /ProductPage/ProductImage/_resampled/SetRatioSize600400-P4302.jpg

[6] Figure 5 difference between Field and Record
Internet source: 28.09.2014. http://www.eww.com.hk/cfdocs /Getting_Started_Building_ColdFusion_MX_Applications /images/db_basicsa.gif

[7] Figure 6 Wind vane
Internet source: 28.09.2014. http://141.22.116.42/bilder /wetterstation/windrichtung.jpg

[8] Figure 7 Three cup anemometer
Internet source: 28.09.2014. http://141.22.116.42/bilder /wetterstation/anemometer.jpg

[9] Figure 8 the USA-1 and its schematics
USA-1, Ultraschalall anemometer, Benutzer handbuch, Page 42

[10] Figure 10 2D vectors
Internet source: 28.09.2014. http://cdn-2.cut-the-knot.org/pythagoras/CalculusProof.gif

[11] Figure 10 3D vectors
Internet source: 28.09.2014. http://upload.wikimedia.org/wikipedia /commons/f/f6/Cartesian_xyz.png

[12] Figure 11 Differences between atan and atan2
Internet source: 28.09.2014. http://i.stack.imgur.com/BZo2D.png

[13] Table 1 Cut from ASCII table
Internet source: 28.09.2014. http://benborowiec.com/wp-content/uploads/2011/07/better_ascii_table.jpg

Literary Sources

[14] Caldarelli, David D. and Ruth S. Campanella. Ear. World Book Online Americas Edition. 26 May 2003

[15] Windows DCB structure Page
Internet source: 28.09.2014. http://msdn.microsoft.com/en-us/library/aa363214(VS.85).aspx

[16] Ultraschall anemometer settings and information
USA-1, Ultraschalall anemometer, Benutzer handbuch

[17] C Programming Reference
Free online Handbook: C++ von A bis Z, Judith Stevens-Lemoine
Lektorat Galileo Computing, Galileo Press · Rheinwerkallee 4 · 53227 Bonn

[18] MYSQL reference
Mark Maslakowski, MYSQL, Markt+ Technik verlag, München, Germany 2001

[19] Linux termios man page
Internet source: 28.09.2014. http://man7.org/linux/man-pages/man3/termios.3.html

Anemometer technical data

13 Technische Daten

Meßbereich

Windgeschwindigkeit		0	...	50 m/s
Windkomponenten		-50	...	50 m/s
Windrichtung		0	...	360 °
Windrichtung	(mit Hysteresis)	0	...	540 °
Temperatur		-30	...	50 °C
Zählereingänge	(Option)	0	...	10 Hz
Pt100 Eingänge	(Option)	-30	...	50 °C
Analog Eingänge	(Option)	-10	...	+10 V
	oder	-5	...	+5 V

Meßauflösung

Windgeschwindigkeit		± 0.01 m/s
Windkomponenten		± 0.01 m/s
Windrichtung		± 0.4 °
Temperatur		± 0.01 K
Pt100 Eingänge	(Option; 12 Bit ±100 °C)	± 0.025 K
Analog Eingänge	(Option; 12 Bit ±10 VDC)	± 2.5 mV

Zeitliche Auflösung

Meßrate[1]	0.004	...	25 Hz
Mittelungsintervall	1	...	65535 Meßwerte

Zeitliche Auflösung (Option für *Online berechnete Turbulenz Parameter*)

Meßrate[2]	1	...	10 Hz
Mittelungsintervall	0,5	...	3600 s

Analog Datenausgabe 12 Bit Auflösung,
0 (4)...20 mA , 500 Ω max. oder 0...10 (5) V

Windgeschwindigkeit	(max. Bereich)	0	...	60 m/s
Windkomponenten	(max. Bereich)	-60	...	60 m/s
Windrichtung		0	...	360 °
Windrichtung	(mit Hysteresis)	0	...	540 °
Temperatur	(max. Bereich)	-30	...	50 °C

Bereich einstellbar für Geschwindigkeiten und Temperatur

Sensor Ausrichtung

Azimut	(einstellbar)	0... 360 °

Leistungsaufnahme

Sensor Elektronik	etwa	2.5	W
Low Power Mode (keine Heizung, *SF* < 1000)	etwa	1.5	W
Sensor Heizung (Option)	etwa	50.0	W

Abmessungen

Schallstrecken	175	mm
Sensorkopf, Mittelstangentyp		
Meßkopf (Ø × Höhe)	320 × 240	mm
Sensorhöhe mit integrierter Elektronik Box	690	mm
Sensorhöhe mit separater Elektronik Box	660	mm
Sensorkopf, Außenbügeltyp		
Meßkopf (Ø × Höhe)	320 × 330	mm
Sensorhöhe mit integrierter Elektronik Box	780	mm
Sensorhöhe mit separater Elektronik Box	750	mm
Separate Elektronik Box (Länge × Breite × Höhe)	280 × 180 × 330	mm
Integrierte Elektronik Box	120 × 120 × 90	mm
Befestigungsklemme (innen Ø × Länge)	40 × 100	mm

Gewichte - ohne Kabel

Sensorkopf, Mittelstangentyp :

Sensorkopf (Integrierte Elektronik Box)			2.8	kg
Sensorkopf (Separate Elektronik Box)			7.1	kg
Separater Sensorkopf	1.8	kg		
Separate Elektronik Box	3.8	kg		

Sensorkopf, Außenbügeltyp :

Sensorkopf (Integrierte Elektronik Box)			3.1	kg
Sensorkopf (Separate Elektronik Box)			7.4	kg
Separater Sensorkopf	2.1	kg		
Separate Elektronik Box	3.8	kg		

USA-1 ...H......

Serial No. 0101111761

Adjustments of Operation Parameters

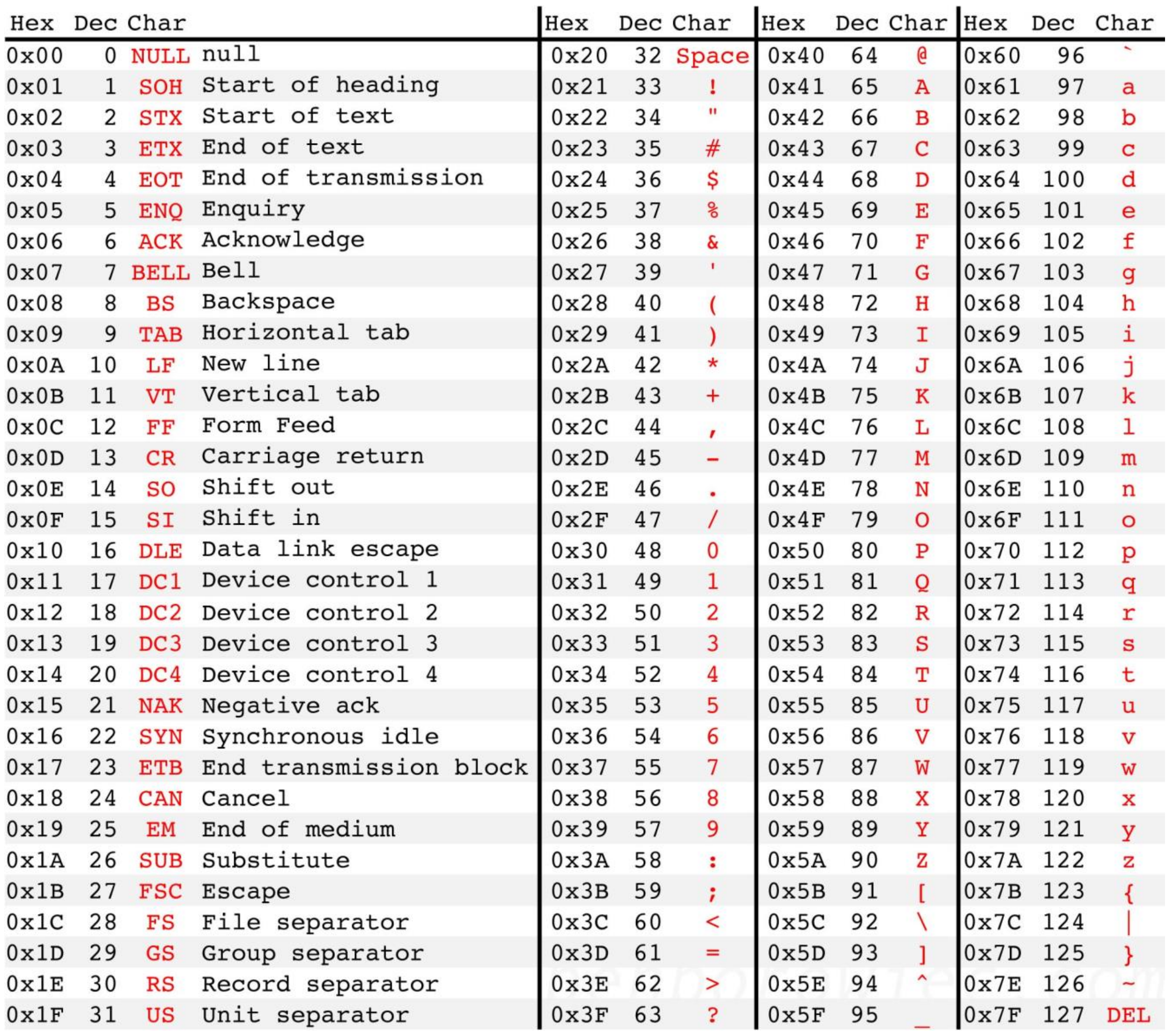
Geändert auf AT: 1 →

AD = 0	/	Device Address	M1 =	/	Modem String
AI = 1	/	Analogue Instantaneous	M2 =	/	Modem String
AO = 0	0	Analogue Offset	M3 =	/	Modem String
AT = 10	10	Averaging Time	MD = 20	20	Data Quality Check
AV = 1	1	Averaging Number	N1, N2, N3	uvcall	Name Parameters
AZ = 0	0	Azimuth	NO = 31	31	NMEA Parameter
BR = 9600	9600	Baud Rate	OA = 0	0	Output Analogue
D1 = 0	/	Delay	OD = 1	1	Output Digital
D2 = 0	/	Delay	OI = 0	/	Output Instantaneous
D3 = 0	/	Delay	PR = 0	0	Protocol
D4 = 0	/	Delay	SA = 0	0	Scalar Averaging
D5 = 0	/	Delay	SF = 10000	10000	Sampling Frequency
D6 = 0	/	Delay	SY = 0	0	Synchronized Averaging
D7 = 0	/	Delay	TR = 4000	4000	Temperature Range
D8 = 0	/	Delay	TV = 0	0	Test Value
FR = 0	/	Frame	TZ = 0	/	Time Zone
HC =	1	Head Correction	US=1	/	Mode of ustar-calculation
HT = 0	0	Heater Control	VR = 6000	6000	Velocity Range
LD = 0	/	Log Data	ZR = 100	100	Velocity Range (z-comp.)
~~Analogue Output~~					
~~Analogue Input~~					
~~Resolution~~		~~a2/e3= ±~~	~~a3/e4= ±~~		~~a4/e5= ±~~
~~PT100~~		~~a5/e6= ±~~	~~a6/e7= ±~~		~~a7/e8= ±~~

Calibration Parameters

Date:	25.11.08						
O1	2277						
O2	2273						
O3	2301						
O4	2293						
O5	2286						
O6	2293						
P1	1761						
P2	1755						
P3	1755						
TC	2106						

Complete ASCII Table



Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Source code

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <math.h>
#include <mysql/mysql.h>
#include <time.h>
#include <errno.h>
#include <getopt.h>

int xRaw0 = 0;
int yRaw0 = 0;
int zRaw0 = 0;
int tRaw0 = 0;

float xRaw = 0;
float yRaw = 0;
float zRaw = 0;
float tRaw = 0;

float xOne = 0;
float zOne = 0;
float tOne = 0;
float yOne = 0;

float xTen = 0;
float zTen = 0;
float tTen = 0;
float yTen = 0;


float hv_minOne = -1;
float hv_maxOne = -1;
float tv_minOne = -1;
float tv_maxOne = -1;

float hv_minTen = -1;
float hv_maxTen = -1;
float tv_minTen = -1;
float tv_maxTen = -1;
float hv = 0;
float tv = 0;
float hd = 0;
float zd = 0;

int CounterOne = 0;
int CounterTen = 0;
float SumdataOne [5];
float SumdataTen [5];

int flag1 = 0;
int flag2 = 0;
int flag3 = 0;
int flag4 = 0;

unsigned long OneMinTime = 0;
unsigned long TenMinTime = 0;
unsigned long timer = 0;
char Timestr[30];
char str[80];

#define PI 3.14159265
char *LOGFILE;
mode_t umask(mode_t mask);

void database_connect(char *localhost,char *user,char *password,char *DB, int a);
void ReadSerial(int tty,char *N,char *device);
unsigned long GetTime();
int LengthTest(char *N);
int     QualityTest(char *N);
int GetValue(char *N);
void GetVelocity(float x,float y,float z);
void GetDirection(float x,float y,float z);
void MinMax(int a);
void OneMinSum(int a);
void OneMinMean(int a);
void Reset(int a);
void WriteLog (char *message);
int isprint(int c);



int main(int argc,char** argv)
{
```

```
//***************************Comand Line arguments

          int Dflag = 0;
          char *device = NULL;
          char *DB = NULL;
          char *localhost = NULL;
          char *user = NULL;
          char *password = NULL;
          LOGFILE = "Wind.log";

          int m1 = 0;
          int m2 = 0;
          int m3 = 0;
          int m4 = 0;
          int m5 = 0;
          int c;
          int help = 0;


          while ((c = getopt (argc, argv, "Dd:n:h:u:p:L:")) != -1)
          {
                    switch (c)
             {
             case 'D':
                    Dflag = 1;
                    break;
               case 'd':
                    device = optarg;
                    m1 = 1;
                        break;
               case 'n':
                    DB = optarg;
                    m2 = 1;
                    break;
               case 'h':
                    localhost = optarg;
                    m3 = 1;
                    break;
               case 'u':
                    user = optarg;
                    m4 = 1;
                    break;
               case 'p':
                    password = optarg;
```

```
                m5 = 1;
                break;
            case 'L':
                LOGFILE = optarg;
                break;
            case '?':
                if (optopt == 'd' || optopt == 'n' || optopt == 'h' || optopt == 'u'
                                  || optopt == 'p' || optopt == 'L')
                        printf ("Option -%c requires an argument.\n", optopt);
                else if (isprint(optopt))
                        printf ("Unknown option `-%c'.\n", optopt);
                else
                        printf ("Unknown option character `\\x%x'.\n",optopt);
                break;
            default:
              break;
            }

        help = 1;
        }

      if(help == 0)
                printf ("-D Run as deamon (Optional)\r\n-d \"Device name\"\r\n-n \"Database
name\"\r\n"
                                  "-h \"Hostname\"\r\n-u \"Username\"\r\n-p \"Password\"\r\n"
                                  "-L \"Log file\" (Optional)\r\n");
      else
                printf ("Deamonize (Optional) %d\r\nDevice name: %s\r\nDatabase
name: %s\r\n"
                                  "Hostname: %s\r\nUsername: %s\r\nPassword: %s\r\n"
                                  "Log file: %s (Optional)\r\n",
                              Dflag, device, DB, localhost, user, password, LOGFILE);

      if (m1 != 1 || m2 != 1 || m3 != 1 || m4 != 1 || m5 != 1)
      {
                printf ("****Please fill in all mandatory fields****\r\n");
                exit(0);
      }




                //-------Deamonize Process------------
      if (Dflag == 1)
      {
        pid_t pid, sid;                              //Our process and Session ID

    pid = fork();
    if (pid < 0)                              // terminate in case of failure
    {
         exit(EXIT_FAILURE);
    }

    if (pid > 0)                              // exit the parent process.
    {
         exit(EXIT_SUCCESS);
    }

    umask(0);                                    // Change the file mode mask

    // Open any logs here


    sid = setsid();                           // Create a new SID for the child process
    if (sid < 0)
    {
         // Log the failure
         exit(EXIT_FAILURE);
    }
    sprintf(str, "gdf %d",sid);
    WriteLog(str);


    if ((chdir("/")) < 0)                     // Change the current working directory
    {
        WriteLog("failure");
         exit(EXIT_FAILURE);                  // Log the failure
    }
    //Close out the standard file descriptors
    close(STDIN_FILENO);
    close(STDOUT_FILENO);
    close(STDERR_FILENO);
    if (sid == 0)                                // exit the parent process.
    {
        exit(EXIT_SUCCESS);
    }
}
```

```
        struct termios stdio;

        int tty;
        int q1 = 0;
        int q2 = 0;
        char N[200];


        fcntl(STDIN_FILENO, F_SETFL, O_NONBLOCK);      // make the reads non-
blocking

        memset(&stdio,0,sizeof(stdio));
        stdio.c_iflag=0;
        stdio.c_oflag=0;
        stdio.c_cflag=CS8|CREAD|CLOCAL;          // 8n1, see termios.h for more
information
        stdio.c_lflag=0;
        stdio.c_cc[VMIN]=0;                 //maximum number of characters before returning
        stdio.c_cc[VTIME]=50;                    //time between each character in tenths of
a second

        tty=open(device, O_RDWR);                //| O_NONBLOCK
        cfsetospeed(&stdio,B9600);           // 9600 baud
        cfsetispeed(&stdio,B9600);          // 9600 baud

        tcsetattr(tty,TCSANOW,&stdio);





        int s = 0;
        memset(N, 0, sizeof N);
        memset(SumdataOne, 0, sizeof SumdataOne);
        memset(SumdataTen, 0, sizeof SumdataTen);

        while (N[0]!='q')
        {
                timer = GetTime();
                ReadSerial(tty,N,device);

                if (strlen(N) >= 43)
                {
                        printf ( "Current Time: %s\r\n", Timestr );
                        q1=LengthTest(N);
                        q2=QualityTest(N);

                        if (q1 == 1 && q2 == 1)
                        {
                                GetValue(N);
                                GetVelocity(xRaw,yRaw,zRaw);
                                MinMax(1);
                                OneMinSum(1);

                                printf("hv = %f     tv = %f     hd = %f     zd = %f\r\n",hv,tv,hd,zd);
                                printf("hv_min = %f hv_max = %f tv_min = %f tv_max
= %f\r\n\r\n",hv_minOne,hv_maxOne,tv_minOne,tv_maxOne);
                                flag1=1;
                        }
                        else
                        {
                                usleep(500000);
                                s = read(tty,N,200);
                        }
                        q1=0;
                        q2=0;
                        memset(N, 0, sizeof N);
                }

                if (timer - OneMinTime >= 60 && flag1==1)
                {
                        printf("              ///////////////  One Minute //////////////\r\n");
                        OneMinMean(1);
                        GetVelocity(xOne,yOne,zOne);
                        GetDirection(xOne,yOne,zOne);
                        MinMax(2);
                        OneMinSum(2);

                        database_connect(localhost,user,password,DB,1);
                        printf("Sum1 = %f   Sum2  = %f  Sum3 = %f   Sum4 = %f COne
= %d\r\n",SumdataOne [0],SumdataOne [1],SumdataOne [2],SumdataOne
[3],CounterOne);
                        printf("x = %f      y  = %f     z = %f      t
= %f\r\n",xOne,yOne,zOne,tOne);
                        printf("hv = %f     tv = %f     hd = %f     zd = %f\r\n",hv,tv,hd,zd);
                        printf("hv_min = %f hv_max = %f tv_min = %f tv_max
= %f\r\n\r\n",hv_minOne,hv_maxOne,tv_minOne,tv_maxOne);
                        flag2=1;
                        Reset(1);
                }
                if (timer - TenMinTime >= 600 && flag2==1)
                {
                        printf("              ///////////////  Ten Minutes //////////////\r\n");
                        OneMinMean(2);
                        GetVelocity(xTen,yTen,zTen);
                        GetDirection(xTen,yTen,zTen);

                        database_connect(localhost,user,password,DB,2);
                        printf("Sum1 = %f   Sum2  = %f  Sum3 = %f   Sum4 = %f COne
= %d\r\n",SumdataTen [0],SumdataTen [1],SumdataTen [2],SumdataTen [3],CounterTen);
                        printf("x = %f      y = %f      z = %f      t = %f
\r\n",xTen,yTen,zTen,tTen);
                        printf("hv = %f     tv = %f     hd = %f     zd = %f\r\n",hv,tv,hd,zd);
                        printf("hv_min = %f hv_max = %f tv_min = %f tv_max
= %f\r\n\r\n",hv_minTen,hv_maxTen,tv_minTen,tv_maxTen);
                        Reset(2);
                }


                fgets (N, 80, stdin);

                /*while (N[0] == 'w')  //revierte SY!
                {
                        memset(M, 0, 100);        //to use this revert nonblocks, set F
                        memset(F, 0, 10);
                        printf("write to ann %s",N);
                        fgets (F, 10, stdin);
                        printf("you wrote %d %d %d %d %d %d",F[0],F[1],F[2],F[3],F[4],F[5]);
                        printf("%s\r\n",F);
                        write(tty_fd,F,4);
                        if (read(tty_fd,M,80)>0)
                        {
                                printf("%s\r\n",M);
                        }
                }*/

        }
        close(tty);
```

```
        return EXIT_SUCCESS;
}

void ReadSerial(int tty,char *N,char *device)
{
        int s = 0;
        char M[100];
        memset(M, 0, sizeof M);
        s = read(tty,M,100);
        if (s>0)
        {
                strcat (N,M);                                   //data is sent in bursts of 8
characters
                printf("%d ",strlen(N));

                if (flag3==1)
                {
                        sprintf (str, "Connection regained at: %s\r\n", Timestr );
                        WriteLog(str);
                }
                flag3=0;
        }
        else if (s==0)
        {

                if (flag3==0)
                {
                WriteLog("Timeout! Read exceeded 5 seconds! Make sure cable is
connected!");
                sprintf (str,"Connection lost at: %s", Timestr );
                WriteLog(str);
                }
                flag3=1;
        }
        else
        {
                sprintf(str,"ERROR! %s might not be a valid port!", device);
                WriteLog(str);
                WriteLog(strerror(errno));
                exit(0);
        }
}

unsigned long GetTime()
```

```
{
        time_t rawtime;
        rawtime = time(NULL);
        struct tm * timeinfo;
        timeinfo = localtime ( &rawtime );
        memset(Timestr, 0, sizeof Timestr);
        strcat (Timestr,asctime (timeinfo));
        if (OneMinTime == 0) OneMinTime = rawtime;
        if (TenMinTime == 0) TenMinTime = rawtime;
        return(rawtime);
}

int LengthTest(char *N)
{
        printf("Recieved String: %s",N);

        if (strlen(N)==43)
        {
                printf("String length Correct    ");
                return(1);
        }
        else
        {
                WriteLog("String is too long\r\n");
                return(0);
        }
}

int QualityTest(char *N)
{
        if (N[0] == 'M' && N[2] == 'x' && N[12] == 'y' && N[22] == 'z' && N[32] == 't')
        {
                printf("Stringstatus: OK\r\n");
                return(1);
        }
        else
        {
                WriteLog("Stringstatus: Damaged\r\n");
                return(0);
        }
}

int GetValue(char *N)
{
        int n = sscanf(N, "M:x =%d y =%d z =%d t =%d ", &xRaw0, &yRaw0, &zRaw0,
&tRaw0);
        if (n==4)
        {
                xRaw = xRaw0;
                yRaw = yRaw0;
                zRaw = zRaw0;
                tRaw = tRaw0;

                xRaw = xRaw/100;
                yRaw = yRaw/100;
                zRaw = zRaw/100;
                tRaw = tRaw/100;

                printf("x = %d          y = %d          z = %d          t = %d    n=%d
\r\n",xRaw0,yRaw0,zRaw0,tRaw0,n);
                printf("x = %f          y = %f          z = %f          t = %f    n=%d
\r\n",xRaw,yRaw,zRaw,tRaw,n);
                return(1);
        }
        else
        {
                printf("ERROR Getting Values");
                return(0);
        }

}

void GetVelocity(float x,float y,float z)
{
        hv = sqrt(x*x+y*y);
        tv = sqrt(x*x+y*y+z*z);
}

void GetDirection(float x,float y,float z)
{
        hd = atan2(-y,-x)* 180 / PI;
        if (hd < 0) hd= hd + 360;
        zd = ((atan2(hv,z)* 180 / PI)-90);
}

void MinMax(int a)
{
        if (a==1)
        {
                if (hv_minOne == -1)
                {
                        hv_minOne = hv;
                        hv_maxOne = hv;
                        tv_minOne = tv;
                        tv_maxOne = tv;
                }

                if (hv < hv_minOne) hv_minOne = hv;
                if (hv > hv_maxOne) hv_maxOne = hv;
                if (tv < tv_minOne) tv_minOne = tv;
                if (tv > tv_maxOne) tv_maxOne = tv;
        }
        else
        {
                if (hv_minTen == -1)
                {
                        hv_minTen = hv_minOne;
                        hv_maxTen = hv_maxOne;
                        tv_minTen = tv_minOne;
                        tv_maxTen = tv_maxOne;
                }

                if (hv_minOne < hv_minTen) hv_minTen = hv_minOne;
                if (hv_maxOne > hv_maxTen) hv_maxTen = hv_maxOne;
                if (tv_minOne < tv_minTen) tv_minTen = tv_minOne;
                if (tv_maxOne > tv_maxTen) tv_maxTen = tv_maxOne;
        }
}

void OneMinSum(int a)
{
        if (a==1)
        {
                SumdataOne [0] += xRaw;
                SumdataOne [1] += yRaw;
                SumdataOne [2] += zRaw;
                SumdataOne [3] += tRaw;
                CounterOne++;
        }
        else
        {
                SumdataTen [0] += xOne;
                SumdataTen [1] += yOne;
                SumdataTen [2] += zOne;
                SumdataTen [3] += tOne;
                CounterTen++;
        }
}

void OneMinMean(int a)
{
        if (a==1)
        {
                xOne = SumdataOne [0] / CounterOne ;
                yOne = SumdataOne [1] / CounterOne ;
                zOne = SumdataOne [2] / CounterOne ;
                tOne = SumdataOne [3] / CounterOne ;
                CounterOne = 0;
        }
        else
        {
                xTen = SumdataTen [0] / CounterTen ;
                yTen = SumdataTen [1] / CounterTen ;
                zTen = SumdataTen [2] / CounterTen ;
                tTen = SumdataTen [3] / CounterTen ;
                CounterTen = 0;
        }
}

void Reset(int a)
{
        if (a==1)
        {
        hv_minOne = -1;
        hv_maxOne = -1;
        tv_minOne = -1;
        tv_maxOne = -1;
        hd = 0;
        zd = 0;
        flag1=0;

        memset(SumdataOne, 0, sizeof SumdataOne);
        OneMinTime = timer;
        }
        else
        {
        hv_minTen = -1;
        hv_maxTen = -1;
        tv_minTen = -1;
        tv_maxTen = -1;
        hd = 0;
        zd = 0;
        flag2=0;
        memset(SumdataTen, 0, sizeof SumdataTen);
        TenMinTime = timer;
        }
}

void finish_with_error(MYSQL *con)
{
        sprintf(str,"%s", mysql_error(con));
        WriteLog (str);
        mysql_close(con);
        exit(0);
}

void database_connect(char *localhost,char *user,char *password,char *DB, int a)
{
        MYSQL *con = mysql_init(NULL);

        char query[1024];
        if (con == NULL)
        {
                sprintf(str, "%s", mysql_error(con));
                WriteLog (str);
                exit(1);
        }

        if (mysql_real_connect(con, localhost, user, password, DB, 0, NULL, 0) == NULL)
        {
                finish_with_error(con);
        }
        if (a==1)
        {
                sprintf(query,"INSERT INTO 1minTable
(X,Y,Z,Hv,Tv,Hd,Zd,Hv_min,Hv_max,Tv_min,Tv_max,Temp)"
                                "VALUES(%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f)"

        ,xOne,yOne,zOne,hv,tv,hd,zd,hv_minOne,hv_maxOne,tv_minOne,tv_maxOne,tOne
);
        }
        else
        {
                sprintf(query,"INSERT INTO 10minTable
(X,Y,Z,Hv,Tv,Hd,Zd,Hv_min,Hv_max,Tv_min,Tv_max,Temp)"
```

```
                                        "VALUES(%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f)"

        ,xTen,yTen,zTen,hv,tv,hd,zd,hv_minTen,hv_maxTen,tv_minTen,tv_maxTen,tTen);
        }

        if (mysql_query(con,query))
        {
                finish_with_error(con);
        }

        mysql_close(con);
}

void WriteLog (char *message)
{
        printf("%s\r\n",message);

        FILE *logg;

        logg = fopen(LOGFILE, "a");
        if (logg == NULL)
        {
                printf( "Write Log: %s\r\n",strerror(errno));
                fclose(logg);
                exit (EXIT_FAILURE);
        }
        else
        {
                fputs(message, logg);
                fputs("\n", logg);
                fclose(logg);
        }
}
```

Create database code

```
CREATE TABLE `1minTable` (
ID int(11) NOT NULL auto_increment,
time timestamp NOT NULL,
  `X` float DEFAULT NULL,
  `Y` float DEFAULT NULL,
  `Z` float DEFAULT NULL,
  `Hv` float DEFAULT NULL,
  `Tv` float DEFAULT NULL,
  `Hd` float DEFAULT NULL,
  `Zd` float DEFAULT NULL,
  `Hv_min` float DEFAULT NULL,
  `Hv_max` float DEFAULT NULL,
  `Tv_min` float DEFAULT NULL,
  `Tv_max` float DEFAULT NULL,
  `Temp` float DEFAULT NULL,
Primary key(ID),
KEY timestamp (time)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


CREATE TABLE `10minTable` (
ID int(11) NOT NULL auto_increment,
time timestamp NOT NULL,
  `X` float DEFAULT NULL,
  `Y` float DEFAULT NULL,
  `Z` float DEFAULT NULL,
  `Hv` float DEFAULT NULL,
  `Tv` float DEFAULT NULL,
  `Hd` float DEFAULT NULL,
  `Zd` float DEFAULT NULL,
  `Hv_min` float DEFAULT NULL,
  `Hv_max` float DEFAULT NULL,
  `Tv_min` float DEFAULT NULL,
  `Tv_max` float DEFAULT NULL,
  `Temp` float DEFAULT NULL,
Primary key(ID),
KEY timestamp (time)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```