Hard- und Software für die Messung und Auswertung des (Raum-)Klimas

# Vorwort

//TODO

# 2. Wahl der Komponenten und Programmiersprachen

2.1. Wahl der Hardwarebasis

Für den Bau eines Gerätes, das Daten aufzeichnen soll, gibt es mehrere mögliche Plattformen, auf denen so ein Projekt realisiert werden kann. Zum einen gibt es diverse Mikrocontroller, zum anderen (Linux) Einplatinencomputer. Ferner gibt es noch einige weitere Optionen, auf die hier aber nicht eingegangen wird, da diese nur für sehr spezielle Anwendungsfälle sinnvoll sind.

Für dieses Projekt muss die Möglichkeit gegeben sein, relativ viele Sensoren anzuschließen und die Sensordaten aufzuzeichnen; Auswertung oder größere Berechnungen am Gerät sind nicht notwendig.

Die Entscheidung, welche Plattform die geeignetere ist, fiel darauf, einen Mikrocontroller als Herzstück des Gerätes zu verwenden, da ein solcher zwar weniger Rechenleistung als ein Einplatinencomputer bietet, aber dafür meistens mehr nutzbare I/O Pins besitzt. So hat z.B. der Raspberry PI 40 Pins, von denen prinzipiell 26 als GPIO (General Purpose Input/Output) verwendet werden können, jedoch können z.B. Pins GPIO 14 und GPIO 15 nicht beliebig verwendet werden, da der Raspberry PI eine Konsole über diese beiden Pins ausgibt. Dies kann zwar deaktiviert werden, ist aber nur eine der Schwierigkeiten, die bei Einplatinencomputern auftreten können. Darüber hinaus ist die Ansteuerung der Pins und die Verfügbarkeit von Libraries dürftig bzw. äusserst aufwendig zu verwenden. Mikrocontroller wie der Arduino MEGA 2560 haben 70 I/O Pins, von denen 16 analoge Signale messen können (mit einem Integrierten ADC (=Analog-Digital-Converter)) und 15 PWM-Signale senden können (PWM=Puls Weiten Modulation; pseudo Analog). Viele Einplatinencomputer wie z.B. der Raspberry PI haben keinen ADC, der aber für einige Sensoren zwingend notwendig ist. Das wichtigste Argument gegen einen Einplatinencomputer ist die Beschränkung der Eingangsspannung auf maximal 3.3 V. Viele Sensoren arbeiten mit 5 V, weswegen diese auch ihr Ausgangssignal mit 5 V senden, was die meisten Einplatinencomputer zerstören würde. Einige Mikrocontroller wie z.B. der Arduino Due arbeiten ebenfalls mit maximal 3.3 V und scheiden deswegen aus.

Die wichtigsten Argumente sind im Folgenden nochmals als Tabelle zusammengefasst.

|  |  |  |
| --- | --- | --- |
|  | Einplatinencomputer  (Raspberry PI 3 Model B) | Mikrocontroller  (Arduino MEGA 2560) |
| Anzahl der nutzbaren Pins | 24 (26) | 70 |
| ADC Pins | 0 | 16 |
| PWM | 1 | 15 |
| Maximale Spannung | 3.3V | 5V |
| Libraries | Oft nicht für alle Programmiersprachen Verfügbar;  Meistens Kompliziert | Alles in Einheitlicher Programmiersprache;  Tendenziell simpler zu verwenden |

Nach diesen Überlegungen bleibt nur eine Option: Arduino.

Während der Entwicklung und möglicherweise im finalen Produkt wird ein Arduino MEGA 2560 verwendet werden, da dieser mehr Pins und mehr Speicher als andere Arduino-Boards hat. Daher muss während der Entwicklung nicht mit mehreren Controllern zur Evaluierung der Sensoren gearbeitet werden. Wenn möglich soll aber im finalen Produkt ein Arduino UNO oder Arduino Pro Mini zum Einsatz kommen, da diese kleiner und kostengünstiger sind.

2.2. Wahl der Programmiersprache

Die Programmiersprache mit der die Firmware des Geräts geschrieben wurde war, nachdem feststand, dass ein Arduino verwendet werden würde bereits klar: Die angepasste Mischform aus C und C++, mit der Arduinos üblicherweise programmiert werden.

Für die Auswertungssoftware gab es im Grunde zwei Möglichkeiten: Java oder C#. Für Java sprach, dass bereits bekannt war, dass ein ähnliches Projekt bereits existierte und es daher möglich sein musste das Projekt durchzuführen. Außerdem bietet Java gegenüber C# den Vorteil plattformunabhängig zu sein. C# ist zunächst nur auf einigermaßen aktuelle Windows-Versionen beschränkt. Es ist zwar möglich C# Code auf Linux auszuführen, dies bedarf allerdings einer speziellen Software und der Code muss gesondert für Linux kompiliert werden. C# bietet für die meisten Anwendungsfälle zwar weit überlegene APIs, jedoch war mir keine Library bekannt, mit der das Zeichnen von Graphen ohne weiteres möglich ist.

Aufgrund der Plattformunabhängigkeit und dem fehlenden Wissen von passenden C# Libraries fiel die Wahl auf Java, obwohl C# sehr viel angenehmer gewesen wäre.

# 3. Ursprüngliche Anforderungen und erste Pläne für die Umsetzung

3.1. Anforderungen der Fachschaften Physik und Geographie

Die Anforderungen der Fachschaft Physik halten sich in Grenzen, vor allem was die Hardware betrifft. Für die Fachschaft Physik ist es wichtig, die Luftfeuchtigkeit und die Temperatur zu messen und auf Knopfdruck auf eine (micro) SD-Karte aufzeichnen zu können. Die Messintervalle für die Aufzeichnung wurden mit etwa einer Messung alle zehn Sekunden festgesetzt. Die Software muss, um für die Fachschaft Physik ausreichend zu sein, in der Lage sein die Messwerte als Kurvenverlauf zu zeichnen und bei Bedarf genaue Messwerte für einen bestimmten Zeitpunkt auszugeben. Eine integrierte Betriebsanweisung ist ebenfalls gefordert.

Darüber hinaus wollte die Fachschaft Geographie die Möglichkeit haben, die Temperatur und die Luftfeuchtigkeit zu messen. Ferner möchte die Fachschaft Geographie, falls möglich, in der Lage sein, die Niederschlagsmenge aufzuzeichnen. Die Messintervalle für den Geographiemodus sind auf vier Messungen pro Tag (00:00 Uhr, 06:00 Uhr, 12:00 Uhr und 18:00 Uhr) festgelegt. Wie schon von der Fachschaft Physik angefordert, soll auch hier die Aufzeichnung auf eine SD-Karte erfolgen. Die Software muss nur in der Lage sein, die Werte in Kurvenform anzuzeigen.

3.2. Umsetzungspläne

Um die gerade beschriebenen Anforderungen zu erfüllen, wird eine RTC d.h. eine Echtzeituhr, zur Festlegung der Dateinamen benötigt. Deren Einsatz ergibt folgende Vorteile: Der Benutzer muss sich nicht auf umständliche Weise um die Festlegung und Eingabe eines Dateinamens kümmern. Es wird auch vermieden, dass mehrere Aufzeichnungen in derselben Datei gespeichert werden, was zu einer Fehlinterpretation von Messwerten führen könnte, die zeitlich weit auseinanderliegen.

Die micro SD-Karten Anbindung soll über die SPI-Schnittstelle (Serial Peripheral Interface) mit einem Arduino Ethernet Shield als Adapter realisiert werden. Shields sind Aufsteckmodule, die von oben auf den Arduino montiert werden und zusätzliche Schnittstellen bzw. Funktionen ermöglichen können. Das Ethernet Shield stellt die Möglichkeit bereit, einen Arduino über einen RJ45 Stecker per Ethernet mit dem Internet zu verbinden, und hat zusätzlich einen micro SD Steckplatz. Die Ethernet-Fähigkeiten sind für diesen Anwendungsfall irrelevant und werden nicht angeschlossen oder benutzt.

Des Weiteren werden Sensoren benötigt, um den eigentlichen Hauptzweck des Geräts zu erfüllen.

Für die Temperatur und die Luftfeuchtigkeit fiel die Wahl auf den AM2302/DHT22 Sensor, der im Arduino Umfeld sehr weit verbreitet ist. Ein weiter Grund, abgesehen von der hohen Verbreitung, für die Entscheidung den DHT22 zu verwenden war, die Fähigkeit des Sensors sowohl Temperatur als auch Luftfeuchtigkeit zu messen.

Der Niederschlagsmengensensor für die Fachschaft Geographie stellte eine sehr viel größere Herausforderung dar. Da kein fertiges Modul, das für dieses Projekt geeignet schien, musste ein Sensor selbst entwickelt werden. Da dieser für die Auftraggeber von geringerer Priorität ist, wurde die Entwicklung dieses Sensors auf einen späteren Zeitpunkt verschoben, da die Entwicklung der allgemeinen Funktionalität Vorrang hat.

# 4. Probleme / Technische Schwierigkeiten bei der Umsetzung

4.1. Probleme beim Bau der Hardware

Beim ersten Test des Geräts schien etwas mit der SD-Karte oder der SD-Library nicht in Ordnung zu sein, da nach einer Test-Aufzeichnung immer noch keine Datei auf der SD-Karte zu finden war. Es stellte sich heraus, dass das Problem darin lag, dass die SD-Library nur die Dateisysteme FAT16 oder FAT32, jedoch ohne die VFAT Erweiterung und damit nur Dateinamen nach dem 8.3 Standard unterstützt.[[1]](#footnote-1)[[2]](#footnote-2) Die Lösung für dieses Problem war einfach: Den Dateinamen kürzen. Nachdem nun die Messwerte aufgezeichnet werden konnten, kristallisierte sich ein weiteres Problem heraus. In der Datei mit den Messwerten fand sich häufig „NaN“ (= Not a Number) anstatt eines Messwertes. Bei genauerer Untersuchung stellte sich heraus, dass sich „NaN“ etwa alle 5-10 Minuten in den Messreihen des DHT22-Sensors fand. Diese Beobachtung führte zu dem Entschluss zwei andere Sensoren zusätzlich anzuschließen und über den Zeitraum von einigen Wochen zu testen, um zu entscheiden welcher Sensor am besten geeignet ist. Die beiden zusätzlich angeschlossenen Sensoren waren beide von Bosch: der BMP180 und der BME280. Beide Sensoren können die Temperatur und den Luftdruck messen, der BME280 kann außerdem auch, wie der DHT22, die Luftfeuchtigkeit messen. Aus den so gewonnenen Vergleichsdaten wurde der Verdacht, dass der DHT22 abgesehen von seinem niedrigen Preis keinerlei Vorteile bietet, als unbrauchbar gilt und sofern möglich vermieden werden sollte, bestätigt.

Es wird zunehmend ersichtlich, dass es Probleme beim Anschließen der Komponenten geben muss, da BMP180, BME280, die RTC (Real-Time Clock/ Uhr) und zwei LCD Displays über das I²C-Protokoll (Inter-Integrated Circuit) gesteuert werden. Der Arduino kann jedoch nur über die Pins 21 (Taktsignal) und 22 (Datenleitung) mit I²C-Geräten kommunizieren. Glücklicherweise ist I²C ein Bus-Protokoll d.h. Es können mehrere Geräte an diesen Bus angeschlossen werden, die nur dann Daten senden, wenn der Master z.B. vom Gerät „LCD-Display 1“ mit der Adresse 0x27 Daten anfordert. Das bedeutet, dass der Arduino in regelmäßigen Intervallen ein einzelnes Gerät ansprechen muss, das daraufhin seine Messwerte zurücksendet und danach wartet bis wieder Daten angefordert werden.

Während der Testphase wurde noch ein weiteres signifikantes Problem bemerkt: die Stromversorgung des Geräts.

Manche sensoren falsche temperaturen => zu nah am mq135

4.2. Probleme bei der Software

Das erste Problem, das auftrat, war, dass es keine geeigneten Beispiele bzw. keine Dokumentation zu JFreeChart, also der Library, mit der die Wertverläufe gezeichnet werden sollten, gab. Weitergeholfen hat hier ein Video auf YouTube, in dem jemand einen Photowiderstand an einen Mikrocontroller angeschlossen hatte und die Messwerte in Echtzeit als Graph darstellen ließ (<https://www.youtube.com/watch?v=cw31L_OwX3A>). Leider konnte das aus dem Quellcode gewonnene Wissen nicht direkt für dieses Projekt verwendet werden, da in dem gefundenen Quellcode nur eine Messreihe gezeichnet werden konnte und es keine Option gab, wie von der Fachschaft Physik angefordert, die Messwerte an einer bestimmten Stelle auszugeben, nachdem auf einen Punkt in der Zeichenebene geklickt wurde. Aus der Demo-Datei, die beim Download der Library mitgeliefert wird, kann abgelesen werden wie einzelne Features heißen, jedoch nicht wie sie Implementiert werden sollen. Anhand der Namen der gesuchten Klassen konnte mithilfe der Code-Vorschläge von Eclipse erraten werden wie auch die von der Fachschaft Physik angeforderte Funktion implementiert werden konnte.

Nachdem nun eine grundlegende Funktionalität gegeben war musste das Layout geordnet werden. Dies war Notwendig, da manches so seltsam angeordnet wurde, dass Teile der Bedienoberfläche gar nicht zu sehen waren oder einfach nicht lesbar waren (siehe Bilder 1-8). Möglicherweise sind die Zuordnungen welches Element an welcher Stelle stehen soll redundant (z.B. übergeordnetes Element und untergeordnetes Element werden beide angewiesen sich am linken Rand anzuordnen). Teilweise traten hier auch sehr merkwürdige Phänomene wie das völlige Ignorieren von Anweisungen oder das Ausführen der gegenteiligen Reaktion. So kam es vor, dass zum Beispiel, sich manche Elemente, nachdem sie mit der Anweisung „element.setAlignmentX(Component.LEFT\_ALIGNMENT);“ nicht wie gewünscht an den linken Rand bewegt, sondern sich so weit nach rechts bewegt, dass das Element zu etwa 75% aus dem Sichtbereich verschwunden sind. Stattdessen hatte die Anweisung „element.setAlignmentX(Component.RIGHT\_ALIGNMENT);“ den Effekt, dass sich das Element am linken Bildschirmrand anordnete.

Um mehr Platz für die Graphen zu haben kam die Idee auf die Java Full-Screen Exclusive Mode API zu verwenden auf. Wie in den Programmierhinweisen[[3]](#footnote-3) erwähnt, wurde mit dem Methodenaufruf *isFullScreenSupported()* überprüft, ob die Funktion unterstützt wird. Das Ergebnis dieser Abfrage war stets *true,* obwohl der Aufruf der Methode *setFullscreenWindow(JFrame)* zwei unterschiedliche Ergebnisse hervorbrachte. (Siehe Bild 5 und Bild 6) Die Ursache für die unterschiedlichen Ergebnisse war, dass beim ersten Test eine AMD Radeon HD5450 benutzt wurde und beim zweiten Test diese dedizierte GPU ausgebaut war, weshalb die Bildausgabe über die integrierte Intel HD 610 erfolgte. Aufgrund der Tatsache, dass die Software, auch während die Intel Grafikeinheit für die Bildausgabe verantwortlich war, meldete, dass der Vollbildmodus möglich sei, wurden weitere Tests auf anderen Computern ausgeführt. Als Ergebnis konnte festgehalten werden, dass auf allen Computern die eine Integrierte Grafikeinheit von Intel verwenden das Problem wie in Bild 6 zu erkennen auftrat. Es gab zwei mögliche Lösungen für dieses Problem. Die erste war, die Vollbildfunktion gänzlich zu entfernen und die zweite Möglichkeit war, eine Fallunterscheidung einzufügen, die zunächst überprüft auf welchem Betriebssystem das Programm ausgeführt wird und je nach System per Befehlszeile den Namen der verwendeten Grafikeinheit abfragt. Wenn der Name der Grafikeinheit das Wort „Intel“ enthält wird das Fenster nur maximiert, und kein Vollbildmodus verwendet. Wenn der Name jedoch das Wort „Intel“ nicht enthält wird der Vollbildmodus verwendet. Das Funktionieren dieser Unterscheidung wurde sowohl auf Windows (Windows 7, 8 und 10) als auch unter Linux (Debian und Ubuntu) mit ATI/AMD Grafikkarten (sowohl Integriert als auch dediziert), Nvidia Grafikkarten und Intel Grafikeinheiten getestet.

Nachdem die Software weitgehend stabil lief wurden auch Aufzeichnungen langer Messperioden mit etwa 1.000.000 Datensätzen zu je 16 Messwerten durchgeführt. Bereits beim ersten Test wurde klar, dass Anpassungen zwingend notwendig waren: die Ladezeit für diese etwa 16.000.000 Messwerte betrug ungefähr 4 Stunden. Als Lösung für dieses Problem wurde die Möglichkeit die Werte zu interpolieren eingefügt d.h. dem Benutzer wurde die Möglichkeit gegeben auszuwählen wie viele Werte zwischen den zu zeichnenden Werten übersprungen werden sollten. Außerdem werden in der finalen Version nicht mehr 16 Werte pro Datensatz gespeichert, sondern voraussichtlich nur noch 5 Werte. Dies sollte dazu führen, dass die Ladezeiten im akzeptablen Bereich bleiben sollten.

# 5. Erweiterte Anforderungen und neues Konzept für die Umsetzung

5.1. Hinzugefügte Anforderung

Bei den Rücksprachen mit den Auftraggebern wurden im Verlauf des Projekts noch einige weitere Anforderungen hinzugefügt. Die Fachschaft Physik wünschte sich die Möglichkeit CO2 Werte, die Umgebungshelligkeit und, falls möglich bzw. aussagekräftig, die Lautstärke zu messen. Ferner wurde der Wunsch, per Knopfdruck die Auswertungssoftware starten zu können geäußert.

Die Fachschaft Geographie fügte die Anforderung, dass der Luftdruck und die Helligkeit zu messen seien hinzu. Ferner sollte ein Sondermodus in der Auswertungssoftware geschaffen werden, der es erlaubt den Tagesdurchschnitt der Temperatur und die Summe der Niederschläge des Tages anzuzeigen.

5.2. Überarbeitetes Konzept

Wie bereits in Kapitel 4.1 erwähnt, wurde die ursprüngliche Entscheidung den DHT22 Sensor zu verwenden verworfen. Da BMP180 und BME280 immer die Gleichen werte lieferten und nun auch noch der Luftdruck gemessen werden soll, fällt die Endgültige Entscheidung, welcher Sensor am besten geeignet ist, zugunsten des BME280 aus.

5.3. Probleme bei der Fertigstellung

# 6. Fertiges Projekt

# 7. Betriebsanleitung

# 8. Inhaltsverzeichnis

1. Wikimedia Foundation Inc. „File Allocation Table“ in <https://de.wikipedia.org/w/index.php?title=File_Allocation_Table&oldid=168585102>, abgerufen am 16.10.2017 [↑](#footnote-ref-1)
2. Arduino AG (Hrsg.): „SD Library“ in <https://www.arduino.cc/en/Reference/SD>, abgerufen am 22.10.2017 [↑](#footnote-ref-2)
3. ORACLE Deutschland B.V. & Co. KG (Hrsg.)„Full-Screen Exclusive Mode” <http://docs.oracle.com/javase/tutorial/extra/fullscreen/exclusivemode.html> [↑](#footnote-ref-3)