Maturaarbeit

Entwicklung eines Arduino-basierendem Embedded-Systems zur Höhenmessung

Unter der Verwendung von Sensor Fusion Algorithmen

Riccardo Orion Feingold

Betreut durch

Stefan Rothe

30. Julie 2019



Abb. 1 Foto von Autor: Höhenmesser

Gymnasium Kirchenfeld

Abteilung MN

Inhaltsverzeichnis

[Vorwort 4](#_Toc21007299)

[Abstrakt 4](#_Toc21007300)

[Einleitung 4](#_Toc21007301)

[Grundlagen 5](#_Toc21007302)

[Altimeter 5](#_Toc21007303)

[Barometrische Höhenmessung 5](#_Toc21007304)

[Höhenformel 5](#_Toc21007305)

[Herleitung «Internationale Höhenformel» 6](#_Toc21007306)

[DWD 7](#_Toc21007307)

[Vergleich: internationale Höhenformel vs. DWD 8](#_Toc21007308)

[Funktionsprinzip eines einfachen Barometers 9](#_Toc21007309)

[GPS 10](#_Toc21007310)

[Wie funktioniert die Positionsbestimmung? 10](#_Toc21007311)

[Effizienz der vertikalen Positionsbestimmung 11](#_Toc21007312)

[IMU Sensor 11](#_Toc21007313)

[Beschleunigungssensor 12](#_Toc21007314)

[Gyroskop 12](#_Toc21007315)

[Magnetometer 13](#_Toc21007316)

[Hardware 13](#_Toc21007317)

[Arduino vs. Raspberry Pi 13](#_Toc21007318)

[Arduino Mega 2560 14](#_Toc21007319)

[Zusammenbau des Höhenmessers 15](#_Toc21007320)

[BME280 15](#_Toc21007321)

[Adafruit Ultimate GPS Logger Shield 16](#_Toc21007322)

[BNO055 18](#_Toc21007323)

[User Interface mit LCD und Keypad 18](#_Toc21007324)

[Bedienkonzept 18](#_Toc21007325)

[State Machine 19](#_Toc21007326)

[Sensor Fusion 20](#_Toc21007327)

[Definition Sensor Fusion 20](#_Toc21007328)

[Map-Matching 20](#_Toc21007329)

[Point-to-Point-Methode 21](#_Toc21007330)

[Kalibrierung des GPS Modul 22](#_Toc21007331)

[Kalibrierung des Barometers 22](#_Toc21007332)

[Höhenmessung mit Map-Matching 23](#_Toc21007333)

[Kalman-Filter 25](#_Toc21007334)

[Definition Filter 25](#_Toc21007335)

[Prozess und Gleichungen 25](#_Toc21007336)

[Erfahrungen bei der Implementation eines Kalman-Filters in Python 26](#_Toc21007337)

[«Feingold-Filter» 27](#_Toc21007338)

[Positionsdaten vom IMU-Sensor 27](#_Toc21007339)

[Resultate: Sensor-Fusion vs. Anti-Fusion 28](#_Toc21007340)

[Fazit 28](#_Toc21007341)

[Literaturverzeichnis 28](#_Toc21007342)

Vorwort

Abstrakt

In dieser Arbeit wird die Entwicklung eines Arduino-basierenden Altimeter unter der Verwendung von Sensor-Fusion-Prinzipien und Methoden dokumentiert und kommentiert. Dieses Embedded-System besitzt einen Barometer, ein GPS-Modul und einen IMU-Sensor. Hinzu kommen noch Elemente der Bedienoberfläche wie etwa das LCD-Display und das Keypad.

Das Ziel der Arbeit ist, ein Produkt vorzustellen, dass eine Höhengenauigkeit von +/- 5m hinkriegt. Dies wird mit Hilfe eines Map-Matching-Algorithmus bewerkstellig, der grundsätzlich dafür sorgt, dass fortlaufend eine Kalibrierung des Barometers als auch des GPS stattfindet (siehe Kapitel Map-Matching).

Da aber für diese Anwendung eine genaue horizontale Positionsbestimmung Voraussetzung ist, werden auch hier Lösungen vorgestellt – zentrale Rolle wird v.a. der «Kalman»-Filter (siehe Kapitel Kalman-Filter) haben, wobei nebst dem auch eine Alternative (siehe Kapitel Feingold-Filter) präsentiert wird. Bei diesen Methoden wird der IMU-Sensor (siehe Kapitel IMU-Sensor) von grosser Bedeutung sein.

Einleitung

﻿In den letzten Jahren hat die Positionsbestimmung an Wichtigkeit zugenommen, da immer mehr Verkehrsmittel genaue Positionsdaten benötigen. Sie erlauben nicht nur eine Standort-Abfrage, sondern aus ihnen lassen sich auch Ankunftszeiten an einem gewissen Ort im Voraus bestimmen oder u.a. verlorene Smartphones wiederfinden. Doch sie werden in der Zukunft weiter an Wichtigkeit zunehmen. Zum Beispiel wird sie für selbstfahrende bzw. selbstfliegende Verkehrsmittel von enormer Bedeutung sein, vor allem für das letztgenannte. Denn hier müssen drei Koordinaten genau sein: Einerseits die X und Y Koordinaten, und die Höhe. Letzteres wirft die häufigsten Probleme auf, denn da treten die meisten Abweichungen in der Messung auf, zum Beispiel beim GPS (Globale Positioning System) oder Altimeter (Barometer, das die Höhe mit Hilfe einer Höhenformel berechnet) - die beiden einfachsten und häufig genutzten Varianten. Kombiniert man beide vernünftig, so dass sie sich gegenseitig unterstützen, lässt sich ein genaueres Höhenmessgerät entwickeln.

So ist das Ziel der Arbeit ein Embedded-System zur Höhenmessung zu entwickeln, welches nahezu in jeder Situation, bei denen GPS und Altimeter ungenaue Resultate liefern würden, zuverlässige Messungen garantiert. Dabei wird nicht nur ein Hardware Konzept geschaffen, sondern auch eine dazu passende Software geschrieben.

Um dieses Embedded-System bestehend aus einem GPS-Modul, einem Barometer und einem Beschleunigungssensor zu entwickeln, beruft sich der Autor auf ein Vorgehen, welches man unter den Namen “Sensor Fusion” kennt. Darunter versteht man das Kombinieren mehrerer Sensoren, mit dem Ziel bessere Resultate zu erhalten als mit einem einzelnen Sensor. Dieses Prinzip wird mit Hilfe zweier Methoden umgesetzt. Bei einer handelt es sich um das sogenannte Map-Matching (siehe Kap. Map-Matching) und bei der anderen um den Kalman-Filter (siehe Kap. Kalman Filter). Ausserdem wird eine Alternative für letzteres vorgeschlagen.

Zunächst werden einige theoretische Grundlagen zur Höhenmessung bekannt gegeben, die für das Verständnis der kommenden Probleme wichtig sind - ein solches Problem wäre z.B.: Wann und warum gibt das GPS unpräzise Höhendaten? Nebst dem GPS wird auch der Altimeter als weitere Variante zur Höhenmessung erklärt. Im nächsten Kapitel werden die einzelnen Hardware Komponenten vorgestellt und gerade anschliessend stellt der Autor sein Bedienkonzept vor. Danach werden die zwei Methoden – Map-Matching und Kalman-Filter sowie auch die Alternative Variante für letztgenanntes – erläutert. Zu guter Letzt wird der Autor kurz auf die Resultate der Messungen eingehen und im Diskussionsteil interpretieren. Sprich, erklären warum die Resultate so aussehen und welche Bestandteile noch fehlen, damit es zu einem besseren Ergebnis kommen könnte.

Grundlagen

# Altimeter

Das Altimeter kennt man unter dem besser verstehbaren Begriff «Höhenmesser» und dient, wie man es bereits ahnen kann, zur Höhenmessung – wobei hier noch nicht von einer konkreten Variante gesprochen wird. Es handelt sich vielmehr um einen Oberbegriff, welcher jede Art zur Höhenmessung einschliesst. Im Rahmen dieser Maturaarbeit wird der Autor auf vier unterschiedliche Arten eingehen: **barometrische Höhenmessung**, **GPS**, **IMU-basierte Höhenmessung** und **modellhafte Höhenmessung**.

# Barometrische Höhenmessung

Die barometrische Höhenmessung ist eine Variante, die am häufigsten verwendet wird, aufgrund ihrer Einfachheit. Denn das einzige, dass nämlich gemessen wird, ist der Luftdruck, welcher mit einem Barometer bestimmt werden kann. Aus diesem wird dann mit Hilfe einer Formel die aktuelle Höhe berechnet.

Dieses Prinzip funktioniert deshalb, da im idealen Fall der Luftdruck beim Aufstieg abnimmt.

Doch wie bereits im Geografie-Unterricht gelehrt wurde, kann sich dieser auf gleicher Höhe aufgrund wechselhafter Wetterbedingungen verändern. Diese Tatsache hat direkten Einfluss auf die Genauigkeit der durch die Formel berechnete Höhe. Aus diesem Grund werden häufig laufend Kalibrierungen durchgeführt, die nichts weiter als einen Höhenabgleich machen. Doch bevor darauf eingegangen wird, wäre es sinnvoll zu wissen, was eigentlich eine Höhenformel ist und wie sie definiert ist.

## Höhenformel

Die wohl bekannteste Höhenformel ist die sog. «Internationale Höhenformel». Sie geht von der Tatsache aus, dass der Luftdruck exponentiell mit zunehmender Höhe abnimmt (siehe Abb. 2). Dabei gilt zu beachten, dass einfachheitshalber bei dieser Formel auf eine sich mit der Höhe ändernde Temperatur verzichtet worden ist. Somit beschreibt sie einen Spezialfall, der in der Physik als «isothermes» System bezeichnet wird. [1]

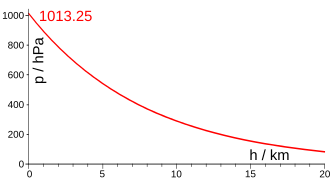


Abb. 2 Exponentielle Abnahme des Luftdrucks mit zunehmender Höhe [1]

Die Formel für diesen Fall lautet wie folgt: Den Luftdruck ***p*** in der Höhe ***h*** lässt sich als , wobei ***p0*** der mittlere Luftdruck auf Meereshöhe ist (***p0*** = 1013hPa), ***g*** die Gravitationskonstante und ***ρ0***die mittlere Dichte (ebenfalls auf Meereshöhe) ebd..

Trotzdem muss erwähnt werden, dass bei der «internationalen Höhenformel» eine Erweiterung möglich ist, die die Temperatur der aktuellen Höhe einbezieht - jedoch nur auf theoretischer Basis, d.h. sie wird nicht durch Messwerte bestimmt. Berechnet werden, kann dies mit Hilfe einer einfachen linearen Funktion: Man geht nämlich davon aus, dass die Temperatur in der Troposphäre mit zunehmender Höhe linear abnimmt – was eigentlich bei konstanten Wetterbedingungen auch der Fall ist. Dies liegt schlichtweg daran, da die Luftmassen von der Wärmebestrahlung der Erde erwärmt werden. [2]Wird nun dies mit in die bereits formulierte Formel eingebaut, so ergibt sich daraus: . Der Unterschied zu der ersten Variante ist, dass hier die Luftdichte nicht mehr als konstant gesehen wird.

Wie Physiker auf diese Definition des Luftdruckes mit Hilfe der Thermodynamik gekommen sind, wird jetzt der Autor im nächsten Unterkapitel erklären.

## Herleitung «Internationale Höhenformel»

Für die Herleitung der «Internationalen Höhenformel» wird ein quaderförmiges Luftpaket mit einer Grundfläche ***A*** und einer infinitesimale Höhe ***dh*** betrachtet. Darin befinden sich Luftmassen von der Dichte ***ρ***. Auf die Grundfläche ***A*** wirken grundsätzlich zwei Kräfte in gegensätzlicher Richtung. Die von oben wirkende Kraft ergibt sich aus der Summe der Gewichtskraft des Luftpakets ***FG = dm\*g = ρ\*dV\*g = ρ\*g\*dh\*A*** und der von der darüber liegenden Luftsäule verursachte Kraft ***Fo = A\*po***, wobei ***po*** der sogenannte Schweredruck ist –dies ist nichts Weiteres als der von der Gewichtskraft ausgeübte Druck auf ein Objekt. Von unten wirkt die von der darunter liegenden Luftsäule erzeugte Kraft ***Fu = A\*pu***.

Zur Vereinfachung wird bei der Formulierung der Formel davon ausgegangen, dass sich das Luftpaket in einem hydrostatischen Gleichgewicht befindet, indem keine Luftströmungen vorhanden sind. Damit dieses Gleichgewicht erhalten bleibt, muss die Summe aller Kräfte **Null** ergeben.

*Fu – Fo – FG = 0*

*A\*pu – A\*po – ρ\*g\*dh\*A = 0*

Kürzen und umstellen, liefert:

Mit Hilfe des idealen Gasgesetzes von Boyle lässt sich die Dichte ersetzen durch die Gl. 2.

Dabei ist ***M = 28,949 g/mol*** (für trockene Luft), die im Übrigen für alle Höhen innerhalb der Troposphäre den gleichen Wert besitzt, die molare Masse der Gase in der Atmosphäre, ***R = 287,1 J/(kg\*K)*** die sogenannte Gaskonstante und ***T*** ist die Temperatur in Kelvin.

Wird nun Gl. 2 in Gl. 1 eingesetzt, so ergibt sich:

Um daraus eine Formel zu kriegen, die es einem ermöglicht die Höhendifferenz zu berechnen, werden die Variablen so getrennt, damit die Differentialgleichung mittels der Integration gelöst werden kann:

Alles ausgerechnet, ergibt sich folgende Gleichung (Gl. 4):

Würde dies noch nach ***p(h1)*** umgeformt werden, so würden wir die bereits erwähnte Gleichung erhalten, die ja – wie wir schon wissen – eine isotherme Atmosphäre annimmt. Wichtig zu wissen ist, dass entspricht.

Um im nächsten Schritt die isotherme Höhenformel mit einer variablen Luftdichte zu erweitern, wird als Ansatz die Temperatur als ein sich linear ändernder Parameter angenommen. D.h. wir haben einen Temperaturgradienten ***a***,der Aussagen darüber macht, um wie viel sich die Temperatur pro Meter erhöht. Die Funktion lautet wie folgt:

Wird bei Gl. 3.1 die Temperatur ***T*** durch ***T(h1)*** ersetzt, so erhalten wir:

Von da an werden beide Seiten wieder integriert und das Resultat wird nach dem Druck ***p(h1)*** umgeformt:

Wobei ***a = 0.0065***, ***h1*** die aktuelle Höhe, ***h0*** die Referenzhöhe (meistens wird sie einfach als 0 gesetzt) und ***T(h0)*** die Temperatur auf der Referenzhöhe (auch hier wird meistens die mittlere Temperatur auf Meereshöhe verwendet) sind.

Würden wir jetzt noch die Gl. 6 nach der Höhe ***h1*** umformen, dabei gleichzeitig die Referenzhöhe auf null setzen, ***p(h0)*** gleich dem mittleren Luftdruck auf Meereshöhe (***p(h0)*** ***= 1013,25hPa***) und für ***T(h0)*** die mittlere Temperatur auf Meereshöhe nehmen (***T(h0) = 288,15K)***, so erhalten wir die weltbekannte «internationale Höhenformel»:

## DWD

Es gibt aber noch eine andere Variante wie wir die Höhe berechnen können, nämlich über die Formel des «deutschen Wetterdienst» [1]:

,

Diese Art der Höhenberechnung ähnelt der Variante mit einem konstantem Temperaturverlauf (siehe Kap. «Herleitung internationale Höhenformel») ebd. Sie unterscheidet sich vor allem in der Verwendung der Temperatur. So wird bei der «internationalen Höhenformel» die Temperatur auf Meereshöhe als Referenz genommen, während bei der Formel des Deutschen Wetterdienstes diejenige auf aktueller Höhe verwendet wird ebd. Mit dieser wird dann eine «virtuelle Temperatur» auf halber Standorthöhe geschätzt, was dank des Standard-Temperaturgradienten ***a*** möglich ist ebd.

Zusätzlich wird dabei noch der Dampfdruck des Wasserdampfanteils ***E*** – d.h. der Druck, der durch den in der Luft gespeicherten Dampf erzeugt wird – mit einbezogen ebd. Um die mittleren Dampfdruckänderungen mit der Höhe zu berücksichtigen, wird ***E*** noch mit dem Faktor ***Ch*** = 0.12 K/hPa multipliziert ebd. Falls keine Messungen für ***E*** möglich sind – aufgrund eines fehlende Luftfeuchtigkeitssensors oder des Weiteren –, so lässt sich dieser auch approximieren, welche auf langjährigen Mittelwerten von Temperaturen und Luftfeuchtigkeit beruht ebd.:

Dabei ist ***t(h)*** die aktuelle Temperatur in Grad Celsius ebd.

## Vergleich: internationale Höhenformel vs. DWD

Damit der der Unterschied deutlich erkennbar wird, hat der Autor Messungen mit beiden Höhenformel durchgeführt. Doch davor wurden beide Höhenformel auf die Starthöhe kalibriert. Das Ganze hat einem sonnigen leicht bewölkten windstillen Samstag zwischen 16:00 und 17:00 stattgefunden. Die Temperaturen lagen zu dieser Zeit zwischen 20°C und 25°C. Das Ergebnis zeigt sich in Abb. 1.

Abb. 3 Vergleich: internationale Höhenformel vs. DWD

Wie sich herausstellt, weichen die geschätzten Höhen der DWD Formel mehrheitlich mehr ab als die der internationalen Höhenformel. So besitzt ersteres einen durchschnittlichen Fehlerwert von 1.744m, während letzteres eine von -0.1244m aufweist. Ein möglicher Grund für diesen Unterschied könnte sich beim Temperatursensor befinden, welcher nur bei der DWD Formel verwendet wird, um die aktuelle Temperatur zu bestimmen. Laut der Firma Bosch besitze der Sensor eine Standardabweichung von +/- 0.5°C bei einer Raumtemperatur von 25°C [3]. Es könnte deshalb durchaus der Fall sein, dass dieser Sensor bei Temperaturen unter oder über 25°C eine grössere Standardabweichung aufweist als die bereits erwähnte.

Was ebenfalls ein Grund sein könnte, wäre die Genauigkeit der «true values», die der Autor von SwissMap entnommen hat. Doch laut deren Angaben besitzen die Daten eine relativ geringe Standardabweichung – sie liegt bei +/- 0.5m [4]. Aufgrund dessen wird dieser Aspekt vernachlässigt.

Ein weiterer Grund, der sicher zu grösseren Abweichungen führen könnte, wäre der Dampfdruck. Dieser wird in dieser Arbeit durch eine Approximation bestimmt (siehe Kap. DWD) und dann mal einen stationsabhängigen Beiwert multipliziert (siehe ***Ch*** in Kap. DWD), wobei für ihn ein Mittelwert genommen wurde. Solche Verallgemeinerungen lassen sich nicht auf Anomalien ein. So kann es durchaus sein, dass der aktuelle Dampfdruck gar nicht mit dem approximierten übereinstimmt.

Abschliessend lässt sich sagen, dass die DWD Formel aufgrund der bereits erwähnten Gründe zwar vielleicht im Schnitt eine grössere Standardabweichung aufweist als die internationale Höhenformel, doch driftet sie weniger ab mit der Zeit. Dies erkennt man bei den letzten fünf Messungen deutlich. Dort entfernt sich nämlich der Wert der internationalen Höhenformel immer weiter vom richtigen, während sich derjenige der DWD Formel annähert. Fazit: Die DWD Formel wäre wahrscheinlich über einen längeren Zeitraum ohne eine Rekalibrierung genauer als die internationale Höhenformel.

## Funktionsprinzip eines einfachen Barometers

Solche Höhenformel sind schon praktisch zu wissen, doch bringen tun sie nicht viel, wenn der momentane Luftdruck nicht bekannt ist. Dieser lässt sich mit Hilfe eines sog. Barometer bestimmen. Das einfachste Beispiel, um dessen Funktionsprinzip zu erklären, ist der klassische Quecksilber-Barometer.

Er besteht aus einem U-förmig gebogenen Glass, gefüllt mit Quecksilber (siehe Abb. 2) [5]. Auf der linken Seite herrscht der Druck, erzeugt durch die Flüssigkeit. Rechts ist das Gefäss geöffnet, aus diesem Grund wirkt dort der Luftdruck ebd. Im Gleichgewicht befinden sie sich nur dann, wenn beide Seiten den gleichen Druck aufweisen ebd. D.h. wird an Höhe zu genommen, so verkleinert sich die Luftsäule ebd. Das Ergebnis: Der Luftdruck sinkt ebenfalls ebd. Das führt dazu, dass die Quecksilbersäule (links) an Höhe verliert, bis der Druck des Quecksilbers gleich dem Luftdruck ist – denn dann befindet sich das System wieder im Gleichgewicht ebd. Das gleiche gilt auch mit abnehmender Höhe ebd.

Daraus erschliesst sich, dass die Länge der Quecksilbersäule ein Mass für die Grösse des Luftdrucks ist ebd. Die Einheit die häufig verwendet wird, ist Torr und bedeutet nichts anderes als mm Quecksilbersäule – dabei entspricht 1 Torr = 1.3322 hPa ebd.



Abb. 4 Quecksilberbarometer [5]

# GPS

Nebst der barometrischen Höhenmessung existiert auch die Möglichkeit mit Hilfe des GPS die Höhe zu bestimmen – aber nicht nur dies, sondern auch die horizontale Position in Grad. Doch was ist das GPS?

Das GPS – ausgeschrieben Global Positioning System – ist ein satellitengestütztes System für Positions-, Geschwindigkeits- und Zeitbestimmung. Es wurde vom US-amerikanischen Militär entwickelt, mit dem Ziel ein System zu besitzen, dass weltweit zu jeder Zeit und Wetterbedingung Positionsdaten liefern kann. Grundsätzlich lasse es sich in drei verschieden Segmente unterteilen: ein Weltraum-, ein Kontroll- und ein Nutzersegment.

Beim ersten handelt es sich um die im Weltraum stationierten Satelliten. Sie befinden sich in insgesamt sechs verschiedenen Orbits, so dass zu jedem Zeitpunkt mindestens drei Satelliten von irgendeinem Ort aus auf der Erde zu sehen sind. Während den Umrundungen der Erde senden sie fortlaufend ihre momentane Position im Orbit und die Zeit, die eine Übertragung vom Weltraumsegment bis zum Nutzersegment benötigt. Das GPS kann unlimitiert viele GPS-Empfänger mit Daten versorgen, da es lediglich nur aussendet.

Das nächste Segment ist eines der wichtigsten, da es sich um die «Gesundheit» der Satelliten kümmert. Es besteht aus global verteilten Überwachungsstationen, die fehlerhafte Lagedaten des Satelliten und auch Ungenauigkeit in der Zeitmessung korrigieren. Dies muss deshalb gemacht werden, da aufgrund externer Störfaktoren der Satellit nicht immer auf dem fixen Orbit unterwegs sein kann.

Beim letzten handelt es sich um den Nutzer, der mit Hilfe eines GPS-Empfänger die vom Satelliten ausgesendeten Daten erhält. Je mehr davon in Sicht sind, desto genauer wird die 3D Positionsbestimmung des Nutzers – dabei wird die Position wie bereits erwähnt in Grad angegeben.

Das Koordinatensystem, das hier verwendet wird, nennt sich World Geodetic System; kurz WGS oder WGS84. Die Position wird hier mit drei verschiedenen Komponenten definiert: Longitude, Latitude und Altitude. Die ersten beiden geben die horizontale Lage in Grad an, wobei die Longitude den Winkel östlich oder westlich des Greenwich Meridians[[1]](#footnote-1) (östlich: 0° bis 180°, westlich: 0° bis -180°) misst, während die Latitude denjenigen nördlich oder südlich vom Äquator ermittelt. Die Altitude gibt die Höhe an in Meter über Meer. (vgl. [6])

## Wie funktioniert die Positionsbestimmung?

Um die 3D Position zu bestimmen, berechnet der GPS-Empfänger die Differenz zwischen der vom Satelliten gesendeten Zeitangabe und der eigenen Uhrzeit. Daraus kann er dann die Distanz zum Satelliten ermitteln. Da die Signale des Satelliten kugelförmig ausgesendet werden, kann der GPS-Empfänger nur Rückschlüsse daraus schliessen, dass er sich irgendwo auf der Aussenseite einer Kugel, die den berechneten Abstand als Radius besitzt, befindet.

Damit aber die Position bestimmt werden kann, müssen die Signale von mindestens vier Satelliten den GPS-Empfänger erreichen. Dadurch bilden die «Signal-Kugeln» eine Schnittfläche, die zwei Schnittpunkten aufweisen – für dies sind nur drei Satelliten nötig. Eine befindet sich im Weltraum, während sich die andere auf der Erdoberfläche befindet. Der GPS-Empfänger ist intelligent genug, um die richtige Position auszuwählen. In Abb. 4 wird das Ganze nochmals bildlich veranschaulicht.



Abb. 5 Darstellung der GPS-Lokalisation (Bild von Autor)

Da die Satelliten keine Atom-, sondern Quarzuhren besitzen, treten häufig zeitliche Abweichungen auf. Eine Ungenauigkeit von einem Tausendstel in der Zeit erzeugt eine Unschärfe bei der Positionsbestimmung von etwa 300km. Glücklicherweise sind die Abweichungen immer gleichbleibend. D.h. der GPS-Empfänger kann dies mit den Informationen des vierten Satelliten ausgleichen – zusätzlich kann die Höhe noch berechnet werden. (vgl. [6])

## Effizienz der vertikalen Positionsbestimmung

Wie genau die gemessene Höhe ist, «hängt massgeblich von der Qualität des Signals ab» [7]. Falls viele Satelliten vom GPS-Gerät aus in Sicht sind, so erhält der Nutzer genauere Höhenangaben, da das Gerät sich die stärksten Signale raussuchen kann. Doch sind nur knapp vier Satelliten zur Verfügung, so muss der Nutzer mit sehr stark abweichenden Positionsdaten rechnen.

Der Grund liegt vor allem an der physikalischen Ähnlichkeit der Signalwellen mit dem Licht. D.h. die Wolken können die Signale abschwächen – aber nicht nur sie, sondern auch die Atmosphärenschichten oberhalb der Troposphäre. So grundsätzlich alles, was auf irgendeine Art und Weise absorbiert oder reflektiert, kann die Signale in ihrer Ausbreitung stören. Die Folge sind unterschiedliche Laufzeiten vom Sender zum Empfänger. Dies kann zu Positionsabweichungen führen von bis zu 100m. (vgl. [7])

# IMU Sensor

Eine weitere Art die Orientierung und Position zu bestimmen ist der IMU-Sensor. Er ist eine inertiale Messeinheit, welche aus drei verschiedenen Komponenten besteht: ein Beschleunigungssensor, ein Gyroskop und ein Magnetometer. Durch sie kann die Orientierung sowie auch die Position des Objekts (auf welchem sich der IMU Sensor befindet) bestimmt werden. Sie wird deshalb häufig in der Raumfahrt und bei Flugzeugen zur Flugnavigation verwendet. (vgl. [8]) In dieser Maturarbeit wurde der IMU-Sensor einerseits als Kompass und andererseits als Alternative zum GPS verwendet (siehe Kap. Sensor Fusion). Doch dazu später noch mehr.

## Beschleunigungssensor

Ein Beschleunigungssensor kann die von einer Kraft ausgeübte Beschleunigung mittels eines Feder-Masse-Systems messen, welches im Prinzip wie in Abb. 5 aussieht:



Abb. 6 Feder-Masse-System eines Beschleunigungssensor [9]

Dabei ist eine Masse an eine Feder angemacht, welche eine bekannte Federkonstante ***k*** besitzt. Die Feder selbst ist wiederum an ein Gehäuse befestigt. Durch die Einwirkung der von aussen wirkenden Kraft ***Fa***, welche durch die Beschleunigung verursacht wird, verschiebt sich die Masse um ***∆z*** relativ zum Gehäuse. Sie kommt zum Stillstand sobald ***Fa*** gleich der Federkraft ist. Wird nun ***∆z*** gemessen, kann daraus über die Formel für die Federkraft die Kraft ***Fa*** berechnet werden. Durch anschliessendes Dividieren durch die Masse lässt sich die momentane Beschleunigung ermitteln. (vgl. [9])

## Gyroskop

Beim nächsten handelt es sich um einen Sensor, der die Rotationsgeschwindigkeit in °/s messen kann. Diese kann auf verschiedene Art und Weise bestimmt werden, aber die kostengünstigste und von der Baugrösse kleinste Variante ist über die Corioliskraft, welches eine auf eine Masse wirkende Trägheitskraft ist, die durch die Rotation des Bezugssystem hervorgerufen wird[[2]](#footnote-2). (vgl. [9])

Doch da der Autor diesen Sensor in dieser Maturaarbeit nie verwendet hat, wird aus diesem Grund auch nicht spezifisch auf seine Funktionsweise eingegangen. Für die absolute Orientierung hat sich der Autor für den Magnetometer entschieden.

## Magnetometer

Ein Magnetometer ist in der Lage die Stärke eines Magnetfelds (z.B. das der Erde) zu messen. Das Magnetfeld wird in der Einheit Gauss angegeben und kann entweder mittels des Hall-Effekts oder des anisotropen magnetoresistiven Effekts – für genauere Erklärungen empfehle der Autor die Arbeit von Tobias Michaelson (vgl. [9]).

Der Magnetometer kann aber wie bereits erwähnt auch als Kompass verwendet werden, der die Lauf- bzw. Fahrtrichtung relativ zu Norden[[3]](#footnote-3) in Grad angibt. Dies ist möglich mit Hilfe von zwei orthogonal ausgerichteten Magnetometern. Doch damit vernünftige Richtungsangaben bestimmt werden, gelten je nach Stärke des Magnetfeldes unterschiedliche Gleichungen. Diese sehen, in C/C++ implementiert, wie folgt aus:

if (int(mag.y()) > 0) theta = atan2(-mag.x(),-mag.y())\*180/Pi;

if (int(mag.y()) == 0 && int(mag.x()) > 0) theta = -90;

if (int(mag.y()) == 0 && int(mag.x()) < 0) theta = 90;

if (int(mag.x()) == 0 && int(mag.y()) > 0) theta = 180;

if (int(mag.x()) == 0 && int(mag.y()) < 0) theta = 0;

if (int(mag.y()) < 0 && int(mag.x()) > 0) theta = atan2(-mag.x(), -mag.y())\*180/Pi;

if (int(mag.y()) < 0 && int(mag.x()) < 0) theta = atan2(-mag.x(), -mag.y())\*180/Pi;

Die Werte der Magnetometer werden noch zu ***integer*** umgewandelt, so dass auch wirklich die Bedingung **mag.y()=0** aufkommt. Bei den Berechnungen hingegen werden sie wieder als ***floats*** verwendet.

Hardware

# Arduino vs. Raspberry Pi

Abb. 7 Arduino Mega 2560 (links)[[4]](#footnote-4); Raspberry Pi 3 B+[[5]](#footnote-5) (rechts)

Bevor es zur Umsetzung des Höhenmessers gekommen war, wurde zuerst lange überlegt, mit welcher Recheneinheit der Autor arbeiten möchte. Die zwei bekanntesten und billigsten Varianten bei der Entscheidung waren auf der einen Seite das Arduino Mega 2560 und auf der anderen das Raspberry Pi 3B+.

Der wesentliche Unterschied zwischen den beiden Rechnereinheiten liegt in der Leistung und Komplexität. Während wir beim Arduino einen einfachen «Microcontroller» besitzen, welches ein Programm auf einmal ausführen kann, so haben wir beim Raspberry Pi einen kompletten Computer, der in der Lage ist, mehrere Programme parallel laufenzulassen. Der Vorteil bei der Arduino Plattform hingegen liegt in der leicht verständlichen Nutzung des Boards als auch der übersichtlichen Programmierungsumgebung für C/C++.

Ein weiterer Vorteil ist auch, vor allem in Bezug auf den MEGA 2560, die grosse Anzahl an Pins, über welche analoge oder digitale Ein- und Ausgänge zugänglich sind. Als kleiner Vergleich: der Raspberry Pi 3 Model B hat 40 Pins, während der Arduino Mega 2560 rund 54 von denen besitzt. Der zahlenmässige Unterschied mag vielleicht verschwindend klein erscheinen, aber wird bei einem Projekt wie diesem mit mehreren Sensoren gearbeitet, so lohnt es sich durchaus einige Pins mehr zu haben – vor allem wenn diese SCL, SDA oder Digital Pins heissen.

Als letztes möchte der Autor noch darauf hinweisen, dass die Datenübertragung vom Computer auf das Arduino Board über eine USB-Schnittstelle erfolgen kann. Im Gegensatz dazu muss bei einem Raspberry Pi Board der Zugang über ein lokales IP-Netzwerk hergestellt werden. Damit die Software auf einem Raspberry Pi aktualisiert oder installiert werden kann, muss es zudem selbst einen direkten Zugang zum Internet herstellen können. Hinzu kommt, dass bei diesem Board umfassende Kenntnisse über die Konfiguration und Anwendung von Unix/Linux vorausgesetzt wird.

# Arduino Mega 2560

Da bei diesem Projekt das «Arduino Mega 2560» Board verwendet wurde und technisches Wissen Voraussetzung ist für das Verständnis, wird hier deshalb kurz auf ein paar seiner Spezifikationen (siehe Abb. 1) eingegangen:



Abb. 8 Technische Spezifikationen des Arduino Mega 2560 Board

* Der ***Flash Speicher*** ist ein «nichtflüchtiger Speicher» [10] bei dem das Programm, welches mit der Arduino IDE geschrieben wurde, abgespeichert wird. Es besitzt einen Speicherplatz von 256 KB von denen 8 KB für eine kleine Software benötigt werden, den sog. «Bootloader». Seine Hauptaufgabe besteht darin, auf ein vom Computer neu hochgeladenes Programm zu warten, welches er nach Empfang im ***Flash Memory*** des Arduino abspeichert [11]. Dies ist der Grund, warum das Arduino-Board per USB-Kabel programmierbar ist.
* «Ein ***SRAM*** (Static Random Access Memory) ist ein flüchtiger Speicher, welcher seine Daten nach Trennen der Versorgungsspannung verliert.» [10] Dort kreiert und manipuliert das Programm seine Variablen, während es am Laufen ist [12]. Dieser Speicher ist ein sehr hilfreiches Tool, wenn das Programm mit grossen Datensätzen arbeiten muss (siehe Kapitel Map-Matching).
* «Das ***EEPROM*** (Electrically Erasable Programmable Read Only Memory) ist wie der Flash Speicher ein nichtflüchtiger Speicher.» [10] D.h. also, dass die dort gespeicherten Informationen nach dem Trennen der Versorgungsspannung erhalten bleiben ebd. Dies hat den Vorteil, dass man z.B. Grundeinstellungen abspeichern kann, so dass man diese nicht immer wieder aufs Neue definieren muss ebd. Ein konkretes Beispiel wären die Kalibrierungswerte des IMU-Sensors.

# Zusammenbau des Höhenmessers

Im nun folgenden Teil wird Schritt für Schritt die Verwendung der einzelnen Module – d.h. des Barometers, des GPS-Moduls und des IMU-Sensors – sowohl hardware- als auch softwaretechnisch beschrieben. Ihren Aufbau möchte der Autor dem Leser ersparen, da er erstens kompliziert ist und zweitens nicht für das Verständnis der gesamten Maturaarbeit relevant ist. Die Informationen des Kapitels Grundlagen weisen bereits das Nötigste auf.

## BME280



Abb. 9 BME280 Sensor (Bild vom Datenblatt [13])

Der BME280 Sensor ist ein Luftdruck-, Temperatur- und Luftfeuchtigkeitssensor zu gleich. Seine Präzision ist bemerkenswert: Laut dem Datenblatt des Sensors [13], ist er in der Lage die relative Luftfeuchtigkeit von 0% bis 100% mit einer Genauigkeit von ± 3% zu messen. Aber auch die Temperatur kann er mit ± 1°C Abweichung in einem Umfang von -40°C bis 85°C bestimmen. Das gleiche gilt auch für den Luftdruck: Hier weist er eine Präzision von ± 1hPa für einen Bereich zwischen 300Pa bis 1100hPa auf. Dieser Parameter wird sogar so genau gemessen, dass die Höhe mittels der Formel in Kapitel «Höhenformel» mit einer Abweichung von ± 1m gemessen werden kann, vorausgesetzt ein Höhenabgleich wurde gemacht und die Wetterbedingungen sind konstant.

Damit dieser Sensor mit dem Arduino Mega 2560 verwendet werden kann, muss er über eine I2C- oder eine SPI-Schnittstelle mit dem Board verbunden werden. Der Autor empfiehlt die I2C-Schnittstelle, da sie einerseits weniger Pins des Boards benötigt und weil mehrere Sensoren durch die gleichen Pins parallelgeschaltet werden können. Dies ist möglich, da es sich bei der I2C-Schnittstelle um ein Bussystem handelt. Aus diesem Grund können mehrere Sensoren auch an denselben Bus angeschlossen werden, da jeder Sensor über seine Adresse - vergleichbar mit einer Hausnummer - gezielt beschrieben (falls er das unterstützt) oder ausgelesen werden kann:

Abb. 10 BME280 über I2C mit Arduino Mega 2560 verbunden; rot: 5V, schwarz: GND (Ground), gelb: SCK, grün: SDI

Normalerweise müsste der Sensor mit einer Spannung von 3.3V versorgt werden, da er aber einen 3.3V Regulator besitzt, kann eine Spannung von bis zu 6.5V ohne jegliche Probleme angeschlossen werden (siehe Datenblatt [13]).

Um auf die Daten der Sensoren per Programm zugreifen zu können, empfiehlt der Autor die beiden Bibliotheken «Adafruit\_Sensor» und «Adafruit\_BME280» herunterzuladen – für letzteres empfiehlt der Autor seine erweiterte, welche zusätzlich noch die DWD Höhenformel beinhaltet[[6]](#footnote-6). Sind sie im Programm per ***#include*** implementiert, so kann ein Objekt der Klasse «Adafruit\_BME280» erstellt werden, mit der dann alle Funktionen benutzt werden können. Für diejenigen, die gerne noch den Modus und die ***Sampling rate*** der einzelnen Sensoren anpassen möchten, gibt es die Funktion ***Adafruit\_BME280::setSampling()*** – sie wird in der Regel in der ***setup()*** Funktion aufgerufen.

## Adafruit Ultimate GPS Logger Shield



Abb. 11 Adafruit Ultimate GPS Logger Shield mit externer GPS Antenne (Bild von «adafruit learning system» [14])

Wie der Name bereits sagt, handelt es sich hier um ein GPS-fähiges Modul, welches bis zu 22 Satelliten gleichzeitig verfolgen kann. Es ist in der Lage bis zu zehnmal in einer Sekunde die Position zu aktualisieren, benötigt nur 20mA und kann jegliche Art von Daten auf eine Mikro-SD-Karte abspeichern. Seine Präzision liegt laut dem Datenblatt bei 1.8m – doch den Erfahrungen des Autors zu folge trifft dies nur bei idealen Bedingungen zu, d.h. bei schönem Wetter und wenig Infrastruktur. Für die Geschwindigkeit muss mit einer Abweichung von +/- 0.1m/s gerechnet werden, aber auch hier trifft dies nur für ideale Bedingungen zu – die maximale messbare Geschwindigkeit liegt bei 515m/s.

Damit dieses Modul mit dem Mega Board benutzt werden kann, müssen eigentlich keine speziellen Kabel Verbindungen gemacht werden – es wird einfach aufs Board gesteckt:

Abb. 12 Verbindung des GPS-Moduls mit dem Arduino Mega 2560; rot: Pin 8 mit RX2 und grün: Pin 7 mit TX2 verbunden

Beim GPS-Shield ermöglicht ein Schalter zwei verschiedene Arten der seriellen Kommunikation. Falls die «direct» Option gewählt wurde, so wird der «GPS serial TTL UART[[7]](#footnote-7)» direkt mit dem «usb-serial converter chip» des Arduino Mega Boards verbunden – hier sind keine zusätzlichen Verkabelungen nötig. Wenn hingegen die «Soft serial» Option verwendet wird, welches die beiden Pins 7 und 8 zu einem weiteren UART umwandeln sollte (funktioniert beim Arduino Mega 2560 nicht), müssen diese Pins mit einem der anderen UARTs verbunden werden – hierbei gilt: Pin 8 mit RX und Pin 7 mit TX. Dank dieser Variante ist der Haupt-UART frei für das Debuggen und Hochladen der Programme – der Autor empfiehlt diese Option auszuwählen.

Um jetzt auf die Daten problemlos zu greifen zu können, muss zuerst die «Adafruit\_GPS» Bibliothek heruntergeladen werden. Bei der Erzeugung des Objekts sollte darauf geachtet werden, dass der richtige UART angegeben wird – für dieses Beispiel wäre das ***Serial2***. Dann können die Einstellungen des GPS-Modul mittels der ***sendCommand()*** Funktion angepasst werden. Eine praktische Einstellung für den Einstieg lautet wie folgt:

GPS.sendCommand(PMTK\_SET\_NMEA\_OUTPUT\_RMCGGA); //1

GPS.sendCommand(PMTK\_SET\_NMEA\_UPDATE\_5HZ); //2

GPS.sendCommand(PGCMD\_ANTENNA); //3

Bei 1 wird dem GPS-Modul mitgeteilt, dass nur RMC und GGA Datensätze ausgegeben werden sollten. Diese liefern die nötigsten Informationen wie Zeit, Datum, Latitude, Longitude, Höhe, Geschwindigkeit und Fix Typ – letzteres gibt nur an, ob GPS Signale erhalten werden (Fix: 1) oder nicht (Fix: 0). Doch damit das Programm auf die Daten zugreifen kann, ist ein ständiges parsen[[8]](#footnote-8) der Datensätze nötig. Auch für das gibt es wieder eine Funktion, nämlich ***parse()***. Für genauere Erklärungen empfehle der Autor die Beispiel Codes der «Adafruit\_GPS» Bibliothek.

Der zweite Befehl stellt die Frequenz ein, mit der die Daten aktualisiert werden und das letzte verlangt nach dem Status der Antenne. (vgl. [14])

## BNO055

Der nächste Sensor ist ein IMU-Sensor, welcher mit einem Beschleunigungssensor, einem Gyroskop, einem Magnetometer und einem Temperatursensor ausgestattet ist. Die Genauigkeit der einzelnen Sensoren ist im Datenblatt nicht nummerisch festgehalten. Es wird nur darauf hingewiesen, dass bei einer korrekt ausgeführten Kalibrierung (siehe [15] S. 47) und bei stabilen Bedingungen (z.B. stabiles Magnetfeld) mit zuverlässigen Messungen gerechnet werden kann.

Verbunden wird der Sensor wie beim BME280 über die I2C-Schnittstelle:

Abb. 13 BNO055 über I2C mit Arduino Mega 2560 verbunden; rot: 5V, schwarz: GND, gelb: SCL, grün: SDA

Damit die Daten des IMU-Sensors gelesen werden können, benötigt der Nutzer vorerst die «Adafruit\_BNO055» Bibliothek. Mit dieser sind nebst den Sensordaten auch die Kalibrierungswerte, die sog. «Offsets», zugänglich. So kann mittels der Funktion ***getSensorOffsets()*** auf diese Werte zugriffen werden. Mit Hilfe der EEPROM-Bibliothek von Arduino können dann diese, auf einfache Weise langzeitig gespeichert werden (siehe Kap. Arduino). Welcher Algorithmus von der Firma Bosch bei der Kalibrierung verwendet wurde, ist dem Autor unbekannt. (vgl. [15] und [16])

User Interface mit LCD und Keypad

Das Kapitel ***User Interface mit LCD und Keypad*** beschreibt das Bedienkonzept und weist insbesondere auf gewisse Schwierigkeiten hin, wie z.B. die Verwendung eines Eingabe-Systems und das hin und her wechseln zwischen den verschiedenen Modi.

Das User Interface wurde mittels eines 4x4 Keypad, welches die Zahlen von 0 bis 9 und die Buchstaben A bis F besitzt, und einem LCD-Display, das eine Anzeige-Matrix von 4x20 Zeichen zur Verfügung stellt, zusammengebaut. Für das Eingabe-System kam noch ein andere Variante in Frage, welche nur so viel Knöpfe besitzen sollte, wie nötig sind. Doch diese wurde durch das Keypad ersetzt, da dieser genügend Spielraum frei lässt, um ein vernünftiges Bedienkonzept zu entwickeln – ausserdem lässt er sich viel einfacher mit den Board verbinden und verwenden.

# Bedienkonzept

Das Bedienkonzept besteht grundsätzlich aus vier Hauptmodi, von denen aus andere Nebenmodi existieren. Sobald der Höhenmesser aufgestartet ist, wird als erstes die Internationale Höhenformel angezeigt. Im LCD-Display ist neben der Höhe auch der momentane Luftdruck und die aktuelle Temperatur zu sehen. Mit Hilfe der Knöpfe C und D kann die Höhe angepasst werden, dabei wird der Luftdruck auf Meereshöhe verändert – C verkleinert die Höhe, während D sie erhöht. Falls die Taste E gedrückt wird, so kann die Höhe durch Manipulieren der Temperatur auf Meereshöhe justiert werden. Beide Variablen werden im Display in der zweiten Zeile angezeigt.

Wenn ein manueller Höhenabgleich zu anstrengend ist, kann durch betätigen der Taste F der Sensor-Fusion Modus eingeschaltet werden. Hier wird sowohl die internationale Höhenformel als auch die DWD Formel angezeigt. Zudem sind auch zwei verschiedene Positionen zu sehen: Eine ist die durch Sensor-Fusion verbesserte Position und die andere diejenige vom GPS. Wenn bei letzteres der DGPS Modus (Fixquality 2) nicht verfügbar ist, wird das GPS Modul durch den Map-Matching-Algorithmus fortlaufend kalibriert (siehe Kap. Map-Matching).

Damit die Sensor-Fusion-Algorithmen aktiv werden und Messungen auf die SD Karte abgespeichert werden können, muss vorerst die Taste C gedrückt werden. Falls dies nicht gemacht wird, nimmt das Programm einfach den ersten Referenzpunkt im Datensatz und kalibriert so die Position und die Höhe. Gestoppt wird der Modus durch wiederholtes Drücken der Taste C und um wieder zum Hauptmodus zurück zu gelangen, muss wieder die Taste F betätigt werden. Damit man von einem Hauptmodus zum nächsten kann und wieder zurück, gibt es die Knöpfe A und B: A für zurück und B für vorwärts.

Der nächste Hauptmodus ist die DWD Formel. Von der Ausgabe und der Auswahl an Funktionen ist sie praktisch identisch zum Modus der internationalen Höhenformel. Der einzige Unterschied sind die Variablen, die für die Anpassung der Höhe verändert werden können. In diesem Modus kann nämlich nur der Luftdruck auf Meereshöhe manipuliert werden. Sonst sind alle Funktionen wie der Sensor-Fusion Modus ebenfalls vorhanden.

Als dritter Hauptmodus kommt das GPS. Hier wird lediglich die Position und die vom GPS gemessene Höhe angezeigt. Die Lage wird in Grad angegeben, kann aber durch Betätigen der Taste C zu Grad und Minute umgewandelt werden. Weitere spezielle Funktionen wie Sensor-Fusion sind nicht vorhanden.

Beim allerletzten Hauptmodus werden alle Varianten der Höhenmessung aufgelistet: internationale Höhenformel, DWD, GPS und die modellierte Höhe (siehe Kap. Höhenmessung mit Map-Matching). Zudem bestehen zwei Varianten der Datensammlung: Bei einer wird dem Programm durch kurzes Drücken der Taste C gesagt, wann eine Messung abgespeichert wird; wenn aber der Knopf länger gedrückt wird, so speichert das Programm fortlaufend Höhendaten in einem Messintervall von einer Sekunde bis es wieder durch längeres drücken der Taste C gestoppt wird.

In Abb. wird das Bedienkonzept nochmals grafisch dargestellt:

# State Machine

«Eine State Machine (auch als endlicher Zustandsautomat bezeichnet) ist ein Verfahren zur Steuerung von «Maschinen» welche ihre Aktionen abhängig von den jeweils vorherrschenden Zuständen oder Zustandsübergängen ausführen.» [10] Damit dies funktioniert, benötigt sie also eine Schaltlogik und einen Zustandsspeicher ebd. Laut Heimo Gaicher gäbe es bei einer State Machine vier verschiedene Aktionstypen:

* Eine Eingangsaktion, die beim Eintritt in einen Zustand ausgeführt wird
* Eine Ausgangsaktion, die beim Verlassen eines Zustandes aktiviert wird
* Eine Eingabeaktion, die durch Betätigen einer Taste im aktuellen Zustand gestartet wird
* und eine Übergangsaktion, die beim Übergang von einem Zustand in den anderen ausgeführt wird.

Nun stellt sich die Frage, wie eine State Machine möglichst effizient programmiert werden kann. Eine mögliche Herangehensweise wäre die Verwendung einer Software wie «Quantum Leaps». Mit dieser lässt sich eine State Machine über eine graphische Programmiersprache programmieren, dabei generiert sie auch gleich den Code in C/C++. Der Vorteil bei dieser Software ist die Übersichtlichkeit der Programmiersprache – was ziemlich praktisch ist, denn bei der Implementation von State Machines kann es schnell zur Verwirrung kommen. Der Nachteil jedoch ist die Verständlichkeit der Software; man benötigt nämlich wie bei jedem Programm eine gewisse Zeit bis man alle Funktionen verstanden hat. Ausserdem lohnt es sich erst wirklich, wenn die State Machines komplizierter sind – d.h. wenn deutlich mehr Zustände vorhanden sind als beim Bedienkonzept des Autors.

Deshalb hat sich der Autor entschlossen, selbst seine eigene State Machine zu schreiben. Hilfreich bei diesem Prozess war das Beispiel von Martin Gomez (siehe [17], Seite 44). Er hat nämlich gezeigt, dass eine State Machine auch ohne ein ***Switch-Statement*** implementiert werden kann, nämlich mit Hilfe eines Arrays, welches aus Funktionszeigern besteht. Die Zustände können über die Variablen ***curr\_state*** und ***prev\_state*** gespeichert werden, welche als Aufzählungstyp definiert werden. Umgesetzt sieht das Ganze wie folgt aus:

//Aufzählungstyp

typedef enum {international=0, dwd, gnss, map\_matching, comparing} State\_type;

//Funktionen, die den jeweiligen Zustand ausführen, falls sie aufgerufen werden

void internationaleFormel(), DWD(), gps(), MAP\_MATCHING(), COMPARE();

//Array aus Funktionszeigern

void (\*state\_table[])() = {internationaleFormel, DWD, gps, MAP\_MATCHING, COMPARE};

//Definierung der Variablen curr\_state und prev\_state

State\_type curr\_state;

State\_type prev\_state;

Sensor Fusion

In diesem Kapitel werden die Methoden Map-Matching, Kalman-Filtering und die Alternative des Kalman-Filters vorgestellt und ausgewertet. Doch davor wird zuerst der Begriff Sensor Fusion definiert.

# Definition Sensor Fusion

Sensor Fusion ist ein Prinzip, das heute praktisch nicht mehr wegzudenken ist. Warum? Weil das was Forscher und andere messen, nicht nur ein Parameter ist, sondern eine ganze Palette. Es reicht nicht, nur die Temperatur zu bestimmen, wenn jemand Aussagen machen möchte über das Wetter von heute und morgen. Dazu braucht es noch einen Barometer, einen Windstärke-Messgerät und vieles mehr – sogar Satellitenbilder können auch behilflich sein. Also der Grundsatz für diese Kapitel lautet: Je mehr Informationen für ein System bekannt oder vorhanden sind, desto mehr kann darüber ausgesagt werden.

Aber bei Sensor Fusion geht es nicht nur um Aussagen, sondern auch darum, dass bessere Resultate erzielt werden können. Dies lässt sich durch Kombinieren der Sensoren bewerkstelligen. Die Idee dahinter sei, dass die Kombination bessere Messungen liefere im Vergleich zu einem einzelnen Sensor [18]. Also, statt nur den Barometer für die Höhenmessung zu verwenden, warum nicht gleich auch noch das GPS hinzunehmen!

Wie gut dann die Ergebnisse sein werden, hängt einzig und allein davon ab wie die Sensoren aneinandergefügt (im übertragenden Sinn) wurden, also welcher Algorithmus bzw. Filter verwendet wurde.

# Map-Matching

Map-Matching ist – wie bereits der Name schon sagt – ein Kartenabgleich. D.h. besitzt man einen Datensatz Z bestehend aus z.B. GPS-Messungen, so wird dieser mit einem Datensatz M, welcher Ortsinformationen von einer Karte besitzt, abgeglichen. Dabei beinhaltet M Datenpunkte, die als Referenzen verwendet werden – also invariant sind.

Diese Methode wird häufig im Bereich der Navigation angewendet. Dort tritt oft das Problem auf, dass die gemessenen GPS Koordinaten mit einer Ungenauigkeit (siehe Kap. Effizient der vertikalen Positionsbestimmung) verbunden sind und deshalb von den Ortsinformationen der Karte abweichen. Durch das Abgleichen der Messungen mit der Karte kann eine Kalibrierung der Sensoren bewerkstelligt werden [19].

Nun stellt sich die Frage wie man so ein Kartenabgleich implementiert. Eine Variante dies zu tun, ist via die Point-to-Point-Methode. Dabei handelt sich um eine durch den Autor abgeänderte Variante.

## Point-to-Point-Methode

Die Grund Idee der Point-to-Point-Methode ist eine Kalibrierung der Sensoren mit Hilfe von Fixpunkten – also Standorte bei denen sowohl Längen- und Breitengrad als auch die entsprechende Höhe angegeben sind. Dabei werden die Koordinaten vom GPS verglichen mit denjenigen vom Datensatz. Entspricht die Messung einem Datenpunkt, findet eine Kalibrierung statt – dazu später mehr.

Die Fixpunkte weisen - wie bereits erwähnt - eine relativ geringe Abweichung auf. Der Autor verwendet für sein Embedded-System die Daten von Swisstopo, die laut ihren Angaben eine Standardabweichung von +/- 0.5m aufweisen – dies gilt für alle drei Dimensionen [20]. Bei der Auswahl dieser Fixpunkte oder auch Referenzpunkte empfiehlt es sich, Standorte auszuwählen, die auf oder in der Nähe der Route sind. Ein Bespiel zeigt Abb. 1.



Abb. 14 Foto von Autor: Referenzstrecke (rot) und mögliche Fixpunkte (grün); Durch einen Rechtsklick bei der gewünschten Position wird ein kleines Fenster mit den benötigten Informationen geöffnet.

Eine mögliche Implementierung eines Datensatzes aus Fixpunkten in C könnte wie folgt aussehen (aus Platzgründen wird nur ein Datensatz aus zwei Fixpunkten gezeigt):

#include <avr/pgmspace.h>

//1

typedef struct {

double lat;

double lon;

double h;

} X\_Y\_Height;

//2

const PROGMEM X\_Y\_Height data[] = {

46.90982,7.35846,585.7,

46.90934,7.35846,586.6

};

//3

double latitude = pgm\_read\_float\_far(&data[0].lat);

Als erstes wird ein neuer Datentyp für eine Struktur definiert, die drei Elemente besitzt: Latitude, Longitude und Höhe. Dann wird ein Array erstellt vom Typ X\_Y\_Height, welches in der ersten Kolonne die Latitude, in der zweiten die Longitude und in der dritten die Höhe beinhaltet. Doch damit diese Daten nicht den Arbeitsspeicher (den sog. SRAM) des Arduino Mega 2560 überfüllen, werden diese mit Hilfe von PROGMEM, das aus der «avr/pgmspace.h»-Bibliothek stammt, in den Flash-Speicher abgelegt (siehe Kap. Arduino). Punkt drei im Code zeigt lediglich wie man auf den Datensatz zugreifen kann.

Jetzt stellt sich natürlich die Frage wie und wann die Kalibrierung der Sensoren – d.h. in diesem Fall der Barometer und das GPS-Modul – mit Hilfe des Datensatzes stattfinden sollte.

## Kalibrierung des GPS Modul

Wie bereits erwähnt, werden die GPS-Koordinaten mit denjenigen vom Datensatz verglichen. Dabei wird nämlich für jeden Fixpunkt die Distanz zum Nutzer unter der Verwendung vom «Satz des Pythagoras» berechnet.

Damit das Programm die Werte besser abspeichern kann (als Vergleich: ein Meter entspricht in Grad 0.000008995), werden die Resultate in Meter umgewandelt, und es wird nicht die Wurzel gezogen, wenn der «Satz des Pythagoras» angewendet wird. Der Faktor wird wie folgt berechnet:

Faktor:

Während dem Prozess, bei welchem die unterschiedlichen Distanzen bestimmt werden, wird immer der am nächsten zum Nutzer liegenden Fixpunkt abgespeichert. Dann wird die Kalibrierung ausgeführt, falls der Abstand «Nutzer – Fixpunkt» kleiner ist als 16m. Ist dies der Fall, berechnet das Programm einen additiven Kompensationswert, der dann den jeweiligen GPS-Koordinaten hinzugefügt wird. Dieser kann durch Subtrahieren des Fixpunktes von der Position des Nutzers berechnet werden.

## Kalibrierung des Barometers

Gleichzeitig kann auch der Barometer justiert werden. Wenn also der Nutzer sich in der gewünschten Nähe zu einem Fixpunkt befindet, so kann die Höhe beim Fixpunkt für die Berechnung des Druckes auf Meereshöhe genutzt werden (siehe Kapitel Höhenformel). Diese wird dann anstelle der 1013.15 hPa als neue Druckreferenz genommen.

Als nächstes wird gezeigt, wie man mit Map-Matching auch die Höhe berechnen könnte.

## Höhenmessung mit Map-Matching

Wie bereits bei der vorherigen Map-Matching Anwendung gezeigt wurde, benötigt eine Map-Matching-Implementation immer einen Datensatz; so ist auch hier dies der Fall. Der hier verwendete Datensatz ist ein von Swisstopo gratis zur Verfügung gestelltes Höhenmodell mit einer Maschenweite von 200m [4]. Dieses besteht ebenfalls aus Fixpunkten mit drei Koordinaten (LV95/Höhe), wobei die mittlere Standardabweichung in allen drei Dimensionen +/- 1.5m beträgt [4]. Damit es mit den GPS Daten vergleichbar ist, muss es vorerst ins WGS-84-Format[[9]](#footnote-9) (lat/lon) umgewandelt werden. Glücklicherweise besitzt Swisstopo einen Formatwandler [21].

Da der Autor für seine Zwecke nicht das komplette Gitternetz der Schweiz benötigt, wird für dieses Projekt nur ein kleiner Ausschnitt bestehend aus 862 Fixpunkten entnommen. Nämlich folgende Region:

Abb. 15 Foto von Autor: Ausgewählte Region mit 862 Fixpunkten; 2000m x 4000m

Möchte man jetzt die Höhe berechnen mit diesem Datensatz, dann lässt sich dies mit Hilfe der Vektorgeometrie bewerkstelligen. Die Idee dahinter ist eine Gleichung für eine Ebene aufzustellen. Dafür benötigt man drei Fixpunkte, aber nicht irgendwelche, sondern die drei, die am nächsten zum Nutzer sind. Da es sich hier um mehr als nur einen Punkt handelt, muss das Programm dementsprechend erweitert werden.

Die Lösung für dieses Problem ist relativ einfach: Das Programm bestimmt als erstes den Fixpunkt, der unmittelbar in der Nähe vom Nutzer ist – dies basiert auf dem gleichen Prinzip wie bereits beim Kapitel Point-to-Point Methode erklärt wurde. Wurde dieser gefunden, wird seine Array-Position gespeichert. Anschliessend beginnt die Sucherei wieder von vorne mit dem Unterschied, dass dieser Punkt übersprungen wird – also konkret wird seine Distanz zum Nutzer nicht gespeichert. Sobald der zweit nächste Punkt bestimmt wurde, merkt sich auch hier das Programm seine Position, so dass bei der dritten Suche auch dieser zusätzlich übersprungen wird.

Als nächstes wird aus den drei Fixpunkten die Gleichung der Ebene definiert. Dafür werden, wie in Abb. 3 gezeigt, zuerst zwei Richtungsvektoren, durch einfache Vektorsubtraktion berechnet.

Abb. 16 Gitternetzsystem

Da die beiden parallel zur Ebene sind, kann die Ebenen-Gleichung mit Hilfe eines Normalvektors, der senkrecht zu jedem Vektor oder Punkt auf der Ebene ist, bestimmt werden. Berechnen lässt er sich aus dem Kreuzprodukt des ersten und zweiten Richtungsvektors. So lautet nun die Gleichung:

Wobei der Normalvektor ist und (Ortsvektor) der am nächsten liegenden Punkt darstellt. Löst man die Gleichung nach z auf, erhält man die Formel, die zur Berechnung der Höhe benötigt wird, wobei für x und y die GPS Koordinaten eingesetzt werden.

Die daraus berechnete Höhe könnte man wiederum zur Kalibrierung des Barometers verwenden (siehe Kapitel Kalibrierung des Barometers). Doch die Genauigkeit dieser Formel hängt davon ab, wie hoch die momentane Qualität der GPS Daten (siehe Kapitel GPS) und wie gross die Höhendifferenz der drei Fixpunkte ist. Je grösser diese ist, desto ungenauer wird das Resultat, und wenn zusätzlich noch das GPS Modul Abweichung liefert, steigt die Ungenauigkeit desto mehr. Zudem kann es zu einer Situation kommen, in der ***c***plötzlich 0 sein kann. Eine Null Division kann nicht berechnet werden. Damit das Programm keine Null Division berechnen muss, wird einfach ein if-statement hinzugefügt, welches im Fall von c = 0 die Höhe des am nächsten liegenden Fixpunkt ausgibt.

Um dem Höhendifferenz-Problem zu entkommen, schlägt der Autor vor, eine weitere Bedingung einzufügen, die die Fixpunkte erfüllen müssen. Nämlich soll die mittlere Höhendifferenz, der drei Fixpunkten, nicht grösser sein als ein bestimmter Wert. Dieser sollte zwischen der kleinsten und der grössten Höhendifferenz im Datensatz sein. Doch eine solche Einschränkung könnte dazu führen, dass mehrere Fixpunkte nie verwendet werden. Deshalb wird diese Map-Matching Anwendung nicht als eine weitere Variante für die Kalibrierung des Barometers angesehen, sondern als eine andere Art die Höhe zu berechnen.

Somit bleibt nur noch das GPS-Problem, das eigentlich auch bei der ersten Anwendung von grosser Bedeutung ist – schliesslich will ja der Autor, dass eine Kalibrierung stattfindet, wenn er bei einem Fixpunkt vorbeiläuft. Um dies zu garantieren schlägt der Autor den ***Kalman-Filter*** vor, der im nächsten Kapitel hauptsächlich auf theoretischer Ebene erläutert wird, da die Berechnungen – Stichwort grosse Matrizen – möglicherweise die Rechenleistung des Arduino Mega2560 übersteigen könnte.

# Kalman-Filter

## Definition Filter

Aus der Chemie kennt man die Filtration von z.B. Flüssigkeiten, die irgendwelche feste Bestandteile besitzen. Dabei wird ein spezielles Papier verwendet, dass nur Teilchen bis zu einer bestimmten Grösse durchlässt; der Rest bleibt haften.

Mit diesem Prinzip könnte das sog. «digital Filtering» erklärt werden, denn auch hier sind Bestandteile vorhanden, die rausgefiltert werden müssen, z.B. könnten dies das Rauschen von Sensoren sein. Doch damit diese «gesäubert» werden können, benötigt man eine mathematische Gleichung, die aus den «verunreinigten» Sensordaten einen Schätzungswert liefert. Dieser sollte natürlich mit dem Lauf der Zeit genauer werden als die Messung selbst. In Abb. 8 wird die Funktionsweise eines Filters vereinfachend dargestellt.

Abb. 17 Vereinfachte Darstellung eines digitalen Filters

Als erstes versorgen die Sensoren den «estimator» mit Daten. Dieser besteht grundsätzlich aus mehreren mathematischen Gleichungen. Ihre Aufgabe ist es, eine vernünftige Schätzung zu berechnen, unter der Verwendung der Sensordaten, einem Model und einer «objective function» – letzteres bestimmt die Ziele bzw. die Genauigkeit, die der «estimator» erreichen sollte. Beim Model handelt es sich um eine Formel, die die Messwerte annähernd vorhersagen kann, dabei werden auf andere Sensoren zu gegriffen, die nicht in der Eingabemenge vorhanden sind. Ein einfaches Beispiel ist das GPS-Modul: Hier schlägt der Autor vor, das Modell mit der Hilfe von einem IMU-Sensor und den Gleichungen der Kinematik zu kreieren.

Obwohl dadurch auf theoretischer Ebene genaue Resultate erzielt werden könnten, spricht das noch lange nicht für die Realität, denn auch die Sensordaten des IMU-Sensors sind nicht absolut perfekt. So stellt sich schliesslich die Frage, wie dieses Problem gelöst werden kann. Darauf wird uns Herr Kalman eine Antwort geben, während dann anschliessend der Autor seine persönliche Meinung dazu äussert und auf eigene Erfahrungen zugreift, die nicht dem, was Herr Kalman prophezeit, entsprechen.

## Prozess und Gleichungen

Der Kalman-Filter besteht grundsätzlich aus zwei Stufen: Die erste ist eine Vorhersage der Messwerte, die aufgrund der Dynamik des Zustandes bestimmt wird, und die zweite eine Korrektur dieser Vorhersage. Als Beispiel: Wenn der Roboter weiss, wo er gewesen ist (vorheriger Zustand) und wie schnell er im Moment fährt (Dynamik des Zustandes), so kann er daraus seine momentane Position schätzen. In der zweiten Phase vergleicht er dann die Schätzung mit den Messwerten und errechnet eine neue, genauere Position. Die Gleichungen, die diesen Prozess beschreiben, sehen wie folgt aus:

Als erstes wird der momentane Zustand zum Zeitpunkt t mittels der linearen Gleichung, die die Dynamik des Zustandes darstellt, berechnet:

wobei *F* und *G* Matrizen sind, die die Dynamik des Systems beschreiben, und *u* der Input-Vektor ist. In dieser Maturarbeit wurden dort die Sensordaten des Beschleunigungssensor eingeben.

Da zu Beginn ein vorheriger Zustand noch nicht existiert, wird bei der ersten Iteration eine initiale Schätzung verwendet, das gleiche gilt auch für die Fehler-Matrix (gibt den Fehler der berechneten Zustandsvektoren und an), welche mit Hilfe der folgenden Gleichung ermittelt wird:

dabei gibt Q die Ungenauigkeit des Models an, welches durch die Gleichung 1 beschrieben wird. Sobald eine Vorhersage über den Zustand beim Zeitpunkt t besteht, wird als nächstes der sog. Kalman Gain berechnet:

wobei R die Fehler-Matrix der Messungen ist und die Matrix H die Zustandsmatrizen und so anpasst, damit sie mit den Beobachtungsmatrizen R und zt addiert und subtrahiert werden können. Der Kalman Gain bestimmt dann in der nächsten Gleichung um viel das Model bzw. die Beobachtungen gewichtet werden sollten und passt die Fehler-Matrix P an:

hierbei ist I die Identitätsmatrix und zt liefert die Messungen. Die aus Gl. 4 und 5 berechneten Zustandsvektoren werden in der nächsten Iteration als und wiedereingesetzt. Für weitere Erklärungen zum Kalman-Filter empfiehlt der Autor die Arbeit «A Kalman Filtering Tutorial For Undergraduate Students» [22].

## Erfahrungen bei der Implementation eines Kalman-Filters in Python

Der Autor hat sich entschlossen, zuerst einen Kalman-Filter für sein eigenes Mobiltelefon mit Hilfe der Pythonista Programmier-App zu programmieren, da der Umgang mit Matrizen in Python (Stichwort Numpy) leichter ist als bei C/C++ der Arduino IDE. Das Resultat, das dabei herauskam, war nicht sehr zufrieden stellend. Der Filter lieferte Schätzungen, die weitaus schlechter waren als die des GPS. Zudem brauchte er ziemlich lange, bis er einigermassen in der Nähe der aktuellen Position war, dabei wurde die Position nicht verändert.

Würde man darauf warten, bis der Filter (Mobiltelefon ist im regungslosen Zustand) Positionen angibt, die in der Nähe von der richtigen Lage sind; so könnte man dessen berechnete Fehler-Matrix P bei der nächsten Benützung des Kalman-Filters anfänglich als initiale Schätzung verwenden. Das würde dann bedeuten, dass der Filter schneller auf die korrekten Werte kommt, da eine Fehler-Matrix P gewählt wurde, die besser die Genauigkeit der Zustandsvektoren beschreibt als die vom Autor eingeschätzte Matrix. Realistisch gesehen, führte dieses Verfahren dazu, dass der Filter wirklich schneller auf die richtige Position konvergierte, doch sobald Bewegung ins Spiel kam, stieg die Ungenauigkeit umso mehr. Der Grund könnte an der Definierung der beiden Fehler-Matrizen Q und R liegen, da diese die Präzision des Models und der Messungen bestimmen.

Obwohl diese Implementation nicht so funktionierte wie sie sollte, konnte sich der Autor dennoch einen Einblick in die Funktionsweise eines der am häufigsten verwendeten Filter verschaffen. Dadurch erhielt er wertvolle Erkenntnisse, die ihm dabei geholfen haben, einen eigenen weniger komplizierten, aber dennoch effektiven Filter zu kreieren, welcher ausserdem Arduino-Mega-2560 freundlich ist.

# «Feingold-Filter»

Bei dieser Alternative handelt es sich um einen Filter, der alle unvernünftigen Messungen des GPS durch die des IMU-Sensors ersetzt. Implementiert wurde dies mit Hilfe eines im Voraus bestimmten Grenzwert, der nur GPS-Messungen durchlässt, die unter dem Grenzwert liegen. Und so funktioniert er:

Der Filter hat zwei Eingänge: Eine liefert vom GPS gemessene Positionsdaten und die andere entnimmt welche, die vom IMU-Sensor ermittelt worden sind. Aus den Informationen des GPS wird mittels Pythagoras die Distanz im Quadrat zwischen der jetzigen und der vorherigen Position berechnet. Dann vergleicht der Filter den Abstand mit denjenigen, der als Grenzwert definiert wurde: Falls er kleiner ist, werden die Messungen des GPS verwendet, sonst werden diejenigen des IMU-Sensors genommen.

Wie gross schlussendlich der Grenzwert ist, hängt einerseits davon ab, wie viel der Nutzer an ungenauen GPS-Messungen toleriert, und andererseits von der Geschwindigkeit, mit der sich dieser fortbewegt. Dabei sollte beachtet werden, dass der Grenzwert nicht kleiner ist als die Distanz, die der Nutzer in der Zeit zurücklegt, bei welcher keine Messung stattfindet.

Als kleiner Exkurs wird noch kurz erklärt wie der Autor mit Hilfe des IMU-Sensors sowohl die Longitude als auch die Latitude bestimmen konnte.

## Positionsdaten vom IMU-Sensor

Um die Position mittels des IMU-Sensors zu berechnen, wurden die Formel der Kinematik verwendet:

wobei ***k = 0.000008995*** ein Umwandlungsfaktor (von Meter in Grad) ist. Die dort eingesetzten Beschleunigungsdaten stammen von einem Beschleunigungssensor, welcher immer in die Laufrichtung zeigt. Mit Hilfe des Magnetometers (siehe Kap. Magnetometer) kann die Orientierung relativ zu Norden in Grad bestimmt werden. Daraus lassen sich unter der Verwendung der Trigonometrie die Beschleunigungen für die Latitude und die Longitude berechnen:

Damit die Geschwindigkeit wieder auf null gesetzt wird, verwendet das Programm zusätzlich die Geschwindigkeitsmessungen des GPS. Mit dessen Hilfe kann das Programm herausfinden, ob der Nutzer sich fortbewegt oder stillsteht.

Im nächsten Kapitel werden die Resultate aus den Sensor-Fusion-basierenden Messungen präsentiert und mit denjenigen ohne Sensor-Fusion Algorithmen verglichen. Da wird sich dann auch herausstellen, ob das Model mit dem IMU-Sensor zuverlässig ist oder nicht.

# Resultate: Sensor-Fusion vs. Anti-Fusion

# Fazit

# Literaturverzeichnis

|  |  |
| --- | --- |
| [1] | Wikipedia, „Barometrische Höhenformel,“ 1 Julie 2019. [Online]. Available: https://de.wikipedia.org/wiki/Barometrische\_H%C3%B6henformel. [Zugriff am 26 August 2019]. |
| [2] | S. Eidgenossenschaft, „Stockwerkgliederung der Atmosphäre,“ 2018. [Online]. Available: http://www.planat.ch/de/wissen/gewitter/stockwerkgliederung-der-atmosphaere/. [Zugriff am 4 September 2019]. |
| [3] | T. Horstmann, „BME280 von Bosch Sensortec kombiniert Messung von Druck, Luftfeuchtigkeit und Temperatur,“ Bosch , 1 Januar 2014. [Online]. Available: https://www.bosch-presse.de/pressportal/de/de/bme280-von-bosch-sensortec-kombiniert-messung-von-druck-luftfeuchtigkeit-und-temperatur-42408.html. [Zugriff am 9 September 2019]. |
| [4] | SwissTopo, „DHM25,“ 2019. [Online]. Available: https://www.swisstopo.admin.ch/content/swisstopo-internet/de/home/products/height/dhm25/\_jcr\_content/contentPar/tabs/items/dokumente/tabPar/downloadlist/downloadItems/868\_1464696772548.download/dhm25infode.pdf. [Zugriff am 28 März 2019]. |
| [5] | lernhelfer, „Barometer,“ lernhelfer, 2010. [Online]. Available: https://www.lernhelfer.de/schuelerlexikon/physik/artikel/barometer. [Zugriff am 22 September 2019]. |
| [6] | M. Eliasson, „semanticscholar,“ 2014. [Online]. Available: https://pdfs.semanticscholar.org/0b93/eb84ff2f48ea8d9e2770e4b45d30609095b1.pdf. [Zugriff am 23 September 2019]. |
| [7] | B. Tom, „Höhenmessung mit GPS oder Barometer,“ Bergfreund, 8 Dezember 2017. [Online]. Available: https://www.bergfreunde.de/basislager/hoehenmessung-mit-gps-oder-barometer/. [Zugriff am 23 September 2019]. |
| [8] | Wikipedia, „Inertiale Messeinheit,“ Wikipedia, 20 Oktober 2018. [Online]. Available: https://de.wikipedia.org/wiki/Inertiale\_Messeinheit. [Zugriff am 24 September 2019]. |
| [9] | T. Michaelsen, „Lagebestimmung druch Sensorfusion mittels Kalmanfilter,“ 27 Juni 2018. [Online]. Available: http://edoc.sub.uni-hamburg.de/haw/volltexte/2018/4392/pdf/Masterarbeit\_Tobias\_Michaelsen.pdf. [Zugriff am 20 Septemeber 2019]. |
| [10] | H. Gaicher, AVR-Mikrocontroller Programmierung in C, Hamburg: tredition GmbH, 2015. |
| [11] | G. P. Vilamil, „Bootload the Arduino Mini,“ 2019. [Online]. Available: https://www.arduino.cc/en/Hacking/MiniBootloader. [Zugriff am 12 Juni 2019]. |
| [12] | Arduino, „Memory,“ Arduino, 2019. [Online]. Available: https://www.arduino.cc/en/tutorial/memory. [Zugriff am 13 Juni 2019]. |
| [13] | Bosch, „BME280,“ September 2018. [Online]. Available: https://ae-bst.resource.bosch.com/media/\_tech/media/datasheets/BST-BME280-DS002.pdf. [Zugriff am 22 März 2019]. |
| [14] | L. Ada, „adafruit learning systems,“ 11 Juni 2019. [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps-logger-shield.pdf. [Zugriff am 23 März 2019]. |
| [15] | Bosch, „BNO055,“ November 2014. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/BST\_BNO055\_DS000\_12.pdf. [Zugriff am 25 März 2019]. |
| [16] | K. Townsend, „Adafruit BNO055 Absoulte Orientation Sensor,“ Adafruit, 22 April 2015. [Online]. Available: https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor. [Zugriff am 17 Juni 2019]. |
| [17] | M. Gomez, „Embedded State Machine Implementation,“ Dezember 2000. [Online]. Available: https://m.eet.com/media/1174905/f-gomez.pdf. [Zugriff am 28 März 2019 ]. |
| [18] | Wikipedia, „Sensordatenfusion,“ 2019. [Online]. Available: https://de.wikipedia.org/wiki/Sensordatenfusion. [Zugriff am 5 August 2019]. |
| [19] | M. e. a. Bevemeier , „Barometric Height Estimation Combined with Map-Matching in a Loosely-Coupled Kalman-Filter,“ 2010. [Online]. Available: https://www.researchgate.net/publication/224199908\_Barometric\_height\_estimation\_combined\_with\_map-matching\_in\_a\_loosely-coupled\_Kalman-filter. [Zugriff am 25 März 2019]. |
| [20] | Swisstopo, „Swisstopo Online Shop,“ 2019. [Online]. Available: https://shop.swisstopo.admin.ch/de/products/height\_models/alti3D. [Zugriff am 29 Mai 2019]. |
| [21] | SwissTopo, „Reframe,“ 2019. [Online]. Available: https://www.swisstopo.admin.ch/de/karten-daten-online/calculation-services/reframe.html. [Zugriff am 28 März 2019]. |
| [22] | M. B. Rhudy, R. A. Salguero und K. Holappa, „A Kalman Filtering Tutorial For Undergraduate Students,“ Februar 2017. [Online]. Available: http://aircconline.com/ijcses/V8N1/8117ijcses01.pdf. [Zugriff am 5 Mai 2019]. |
| [23] | Wikipedia, „World Geodetic System 1984,“ 2019. [Online]. Available: https://de.wikipedia.org/wiki/World\_Geodetic\_System\_1984. [Zugriff am 7 August 2019]. |

1. Entspricht 0° [↑](#footnote-ref-1)
2. https://de.wikipedia.org/wiki/Corioliskraft [↑](#footnote-ref-2)
3. Wird häufig als 0° angegeben. [↑](#footnote-ref-3)
4. https://store.arduino.cc/mega-2560-r3 [↑](#footnote-ref-4)
5. https://de.rs-online.com/web/p/entwicklungskits-prozessor-mikrocontroller/8968660/ [↑](#footnote-ref-5)
6. <https://github.com/RF4587/Maturaarbeit> [↑](#footnote-ref-6)
7. UART ist ein anderer Begriff für «Serial Port» [↑](#footnote-ref-7)
8. Im Sinne von extrahieren [↑](#footnote-ref-8)
9. World Geodetic System 1984 [23] [↑](#footnote-ref-9)