CanSat 2015 Team Gamma Dokumentation

Alexander Brennecke Till Schlechtweg Marc Huisinga Robin Bley
Steffen Wißmann Alexander Feldmann Kevin Neumeyer

22. Mai 2015

Inhaltsverzeichnis

1 Einleitung

1.1 Teamorganisation und Aufgabenverteilung

Das gesamte Team besteht aus sieben Schülern und zwei betreuenden Lehrern. Die sieben Schüler sind jedoch intern in mehrere kleinere Teams aufgeteilt. Innerhalb der Teams ist jedoch kein Teammitglied vollkommen an seine Aufgaben gebunden, da uns ein guter Austausch und eine hervorragende Zusammenarbeit zwischen den einzeln Teammitgliedern und Teams wichtig ist. Die Arbeit der Gruppen und der einzelnen Personen werden im folgenden erläutert:

- Das Hardware-Team besteht aus drei Personen, welche sich um den Bau des Satelliten selber, dem Design und dem Bau der Dose sowie der Programmierung des Mikrocontrollers kümmern. Zu diesem Team zählen folgende Personen:

  Alexander Brennecke ist verantwortlich für das Design der Dose. Dazu zählt die Konstruktion der eigentlichen Dose und die Anordnung der Sensoren im inneren der Dose.

  Till Schlechtweg ist verantwortlich für die Funktionalität des Mikrocontrollers und den ausgewählten Sensoren.

  Steffen Wißmann ist verantwortlich für die Übertragung der Daten zur Bodenstation und dem Programmcode des Mikrocontrollers.

- Das Software-Team besteht aus vier Personen, welche sich um das Programmieren der Bodenstation und der Android Applikation kümmern. Allgemein gilt für alle Personen dieser Gruppe, dass die Grenzen der Zuständigkeitsbereiche der verschiedenen Personen verfließen, wobei jede Person allerdings noch ein gewisses Spezialgebiet besitzt. Dieses Team besteht aus folgenden Personen:

  Robin Bley ist verantwortlich für das Implementieren der Datenverarbeitung der Bodenstation und für das Testen von kritischen Bereichen innerhalb der Datenverarbeitung.

  Alexander Feldmann ist verantwortlich für die Entwicklung der Android-Applikation.

  Marc Huisinga ist ebenfalls verantwortlich für das Implementieren der Datenverarbeitung der Bodenstation, für die Entwicklung der Datenvisualisierungskomponenten und für die Architektur der Datenverarbeitung.

  Kevin Neumeyer ist verantwortlich für die zusammenführende Architektur der Bodenstation, die grafische Umgebung und die Administration der Software-Repositories.

- Zudem gib es ein Team, bestehend aus Alexander Brennecke und Till Schlechtweg, zur Organisation, Kommunikation mit Sponsoren und Öffentlichkeitsarbeit.

- Betreut wird das Projekt durch zwei Lehrer unserer Schule:

  Mathematiklehrer Harm Hörnlein , welcher als Ansprechpartner für die Software Gruppe zur Verfügung steht.

  Physiklehrer Frank Marshall , welcher als Ansprechpartner für die Hardware Gruppe und den CanSat Wettbewerb zur Verfügung steht.

Die Arbeit an dem Projekt findet zum größten Teil wöchentlich am Dienstag und Mittwoch Nachmittag in den Laboren unserer Schule statt. Die Labore sind mit diversen Werkzeugen ausgestattet, sodass sowohl die Software als auch die Hardware Gruppe dort problemlos arbeiten kann. Zusätzlich zu diesen vier bis acht Stunden pro Woche kommen fünf Projekttage, welche uns von der Schule gestellt wurden. Aber natürlich arbeitet jedes Teammitglied auch außerhalb dieser Treffen an seinem Fachgebiet, soweit dies möglich ist. Zusätzlich gibt es immer wieder Treffen mit externen Unterstützungen, oder Zeit in der Schule, wenn Vertretungs- oder Mitbetreuungsunterricht stattfindet.

1.1.1 Stärken des Teams

Die große Stärke des Teams ist es, dass es auch schon vor diesem Projekt existiert hat und sich somit sehr gut kennt. Die Teilnahme am europäischen CanSat Wettbewerb 2014 hat dazu geführt, das Ebenfalls von Vorteil ist, dass jedes Teammitglied durch unsere schulische Ausbildung genügend Wissen hat, um auch außerhalb seines Fachgebietes unterstützend tätig zu sein. Zudem ist die Arbeit- und Leistungsbereitschaft der meisten Teammitglieder überdurchschnittlich gut.

1.1.2 Verbesserungsbereiche des Teams

Der größte Verbesserungsbereiche des Teams liegt ganz klar im Zeitmanagement. Für viele Aufgaben wird zu wenig Zeit eingeplant. Oft kommt es auch vor, dass der Schwerpunkt der Arbeit auf Dingen liegt, die nicht höchste Priorität haben und somit Zeit beanspruchen, die eigentlich dringend woanders gebraucht wird. Ebenfalls problematisch ist, dass die meisten Teammitglieder gerne neue Technologien oder Praktiken ausprobieren wollen. Dieses Interesse ist zwar löblich und für die Einzelperson sehr lehrreich, jedoch kommt es bei neuen Technologien und Praktiken oft zu Problemen, die man bei bereits bekannten deutlich schneller lösen könnte.

1.2 Das Missionsziel

Die Idee hinter dem gesamten Projekts bezieht sich auf die extremen Umweltbelastung und ihre Folgen für den Menschlichen Körper. Ausschlaggebend für diese Idee ist ein Zeitungsartikel der Zeit, welcher über eine drohende Klage der EU-Kommission in Brüssel berichtet. (vgl. Die Zeit, 24.10.2014). Die EU-Kommission droht mit einer Klage gegen Deutschland, da die deutsche Bundesregierung bisher zu wenig Aufwand betreibt, um die Feinstaubkonzentration in der Luft zu reduzieren. Wir möchten diesen Aspekt aufgreifen und Messungen durchführen um die tatsächlichen Werte zu bestimmen. Der CanSat Wettbewerb eignet sich optimal dazu, da er uns die Möglichkeit bietet die Messungen nicht nur auf dem Boden sondern in verschiedenen Schichten der Atmosphäre durchzuführen. Feinstäube stehen in Verdacht, Krankheiten wie Asthma, Herz-Kreislauf Beschwerden und Krebs zu begünstigen.

Da der menschliche Körper nicht nur durch Feinstaub belastet wird haben wir uns entschlossen auch die Intensität der UV-Strahlung, welche die Hauptursache für Hautkrebserkrankungen ist, zu messen. Zusätzlich soll auch der Ozonwert bestimmt werden, da Ozon bereits in geringen Konzentrationen gesundheitsschädlich ist und zu Reizungen der Atemwege führen kann.

Für sich genommen ist jede dieser drei Größen schädlich für den Menschen. Im Zuge des Projektes wollen wir jedoch versuchen herauszufinden, ob es einen Zusammenhang zwischen ihnen gibt. Beispielsweise ist herauszufinden, ob ein höherer Ozon Gehalt gleichzeitig einen niedrigeren Feinstaubgehalt mit sich bringt.

Zusätzlich zum Bau des Messsystems im CanSat ist es unser Ziel eine einwandfreie Verarbeitung, Analyse und Präsentation der gemessenen Werte zu erzielen. Um dies zu garantieren programmieren wir ein eigenes Analysetool. Dieses Tool ermöglicht es uns die gemessenen Werte, während des Fluges des Satelliten, auszuwerten. Die Werte sollen dabei anschaulich und in Abhängigkeit zueinander dargestellt werden.

Um die Daten auch mobil verfügbar zu haben wollen wir eine Android Applikation bereitstellen. Diese Applikation soll vorerst nur für unser Projekt optimiert sein, bei Erfolg jedoch auch die Werte andere Teams anzeigen können.

1.3 Praktischer Nutzen für den Auftragsgeber

Die Ausrichter des Wettbewerbes, welche in unserem Projekt als Auftraggeber angesehen werden, können sich aus unserem Projekt wahrscheinlich relativ wenig praktischen Nutzen ziehen. Der Wettbewerb im Allgemeinen bietet den Veranstaltern jedoch mehrere Möglichkeiten. Zum einen können sie dadurch mehr Jugendliche für die Bereiche Raumfahrt, Elektrotechnik, Informatik und Physik begeistern. Zum anderen kann es auch möglich sein, dass Mitglieder der Jury, welche

meist in einem der gerade genannten Fachbereiche tätig sind, Lösungsansätze für Probleme der Wissenschaft in einem der Projekte wiederfinden.

2 Beschreibung des CANSAT

2.1 Missionsüberblick

Wir haben uns für den Satelliten überlegt, dass dieser so weit wie möglich individuell sein sollte. Daher greifen wir nicht auf das, vom Wettbewerb bereitgestellte T-Minus CanSat Kit zurück. Stattdessen haben wir uns im Detail überlegt, welche Sensoren unseren Erwartungen entsprechen und wie wir diese bestmöglich innerhalb der Dose platzieren können. Zusätzlich möchten wir nicht auf eine Cola-Dose als Hülle zurück greifen, sondern möchten auch hier unser eigenes Design erschaffen.

2.2 Mechanisches und Strukturdesign

Wir haben den CanSat in drei Komponenten aufgeteilt: Die Hülle, die Innenwand und die Sensorik Platine. Diese drei Komponenten bilden den Hauptbestandteil des CanSats und haben maßgeblich zu dem mechanischen und strukturellem Design beigetragen. Im nachfolgenden wird kurz auf jeden dieser Komponenten eingegangen und die exakte Funktion im Zusammenhang erklärt.

2.2.1 Fachliche Grundlage

Um die 3D gedruckte Wand zu erzeugen wurde die 3D Moddelierungssoftware Sketchup von Google verwendet. Sketchup bietet die Möglichkeit vergleichsweiße einfach 3D Modelle zu zeichnen. Um dies zu tun muss klar sein, welche Objekte gezeichnet werden sollen. Diese Objekte müssen vermessen und innerhalb von Sketchup gezeichnet werden. Dies erfordert die Kentniss über gewisse mathematische Methoden zur Berechnung von Kreisen, Flächen und Körpern. Die meisten 3D-Drucker benötigen Datein des Types .stl, welche in Sketchup mit einem Plugin erzeugt werden können. Zum fertigen von GFK Komponenten wird ein Körper benötigt, auf welchen das GFK laminiert werden kann. In unserem Fall ist dieser Körper zylindrisch, mit einem Durchmesser von 31,5 mm, und aus Aluminium gefräst.
Um die Platine zu erstellen wurde die Design Software Eagle PCB verwendet. Eagle bietet die Möglichkeit sowohl Schaltpläne als auch das entsprechende Layout zu erstellen. Im Anschluss wurde die Platine, mit Hilfe und Mitteln des Hackerspace Bremen e.V. geätzt.

2.2.2 Die Hülle

Wir haben uns dazu entschieden, die äußere Hülle aus GFK (Glasfaser verstärkter Kunststoff) zu fertigen. Dieses hat die Eigenschaften, dass er bei einem sehr geringen Gewicht, und bei einer geringen Wandstärke trotzdem eine gewisse Stabilität aufweißt. Aus dem GFK haben wir eine Röhre mit einem Innendurchmesser von 31,5 mm und einem Außendurchmesser von 33,5 mm laminiert. Diese Röhre wurde auf eine Länge von 111 mm gekürzt und gefeilt. Um die Röhre oben und unten zu verschließen haben wir uns bei Thyssen Krupp System Engineering zwei Aluminium Deckel fräsen lassen. Diese haben uns ebenfalls durch ihr geringes Gewicht und ihre hohe Stabilität überzeugt.

2.2.3 Innenwand

Um die Elektronik innerhalb der Hülle zu platzieren und zu befestigen haben wir uns dazu entschieden eine Wand anzufertigen. Diese Wand teilt die Hülle mittig und bietet so auf beiden Seiten Platz um unser Mikrocontroller Board und unsere Sensorik Platine zu befestigen. Beide Bauteile werden mittels vier Gewindestangen an der Wand befestigt. Durch die Technik des 3D-Druckens ist es möglich der Wand ein sehr geringes Gewicht bei einer verhältnismäßig hohen Stabilität zu verleihen. Zusätzlich gibt es uns die Möglichkeit die Wand millimetergenau zu gestalten.
Am unteren Ende der Wand befindet sich eine Aushöhlung, sowie ein Fuß. Diese ist zum einen dafür da um den Sharp Feinstaub Sensor zu befestigen. Zum anderen gibt der Fuß der Wand und somit dem gesamten Satelliten eine gewisse Stabilität. Der Fuß besitzt auf der einen Seite der

Wand Bohrungen. Diese Bohrungen werden verwendet um die Aluminiumdeckel an der Wand zu befestigen. An der oberen Seite der Wand befinden sich ebenfalls solche Bohrungen um den oberen Deckel der Hülle zu befestigen. Da der Feinstaubsensor einen Luftzug benötigt befindet sich ein Durchlass innerhalb der Wand. Um das Mikrocontroller Board mit der Sensorik Platine zu verbinden existiert ein Fenster in der Mitte der Wand. Um die Sensorik Platine und das Mikrocontroller Board an der Wand zu befestigen existieren vier Bohrungen.

2.2.4 Die Sensorik Platine

Die Sensorik Platine ist eine von uns geätzte Platine, welche mit unseren Sensoren bestückt ist. Es gibt mehrere positive Aspekte, die eine eigene Platine mit sich bringt. Zum einen bietet sie eine stabile Plattform für die Befestigung der Sensoren. Zum anderen sparen wir uns dadurch eine Menge Kabel, welche deutlich störanfälliger sind als eine Platine. Die Platine hat an den entsprechenden Stellen Bohrungen um sie mit der Zwischenwand und dem Mikrokontrollboard zu verbinden. Die Platine bietet Platz für folgende Module:

- BMP108 Drucksensor: Misst den Luftdruck und gibt diesen, sowie die daraus berechnete Höhe zurück
- Sparkfun UV Sensor: Misst die Intensität des Spektrums 270-380 nm, welches dem UVA und UVB Spektrum entspricht
- TMP006 Infrarot Temperatursensor: Misst die Temperatur eines dünnen Aluminiumstückes in der Außenwand
- Adafruit Ultimate GPS: Bestimmt die aktuelle Position sowie die Höhe
- APC220 Transceiver Modul: Sendet die Daten als JSON String zur Bodenstation
- Steckplatz zum Anschluss des Sharp Feinstaub Sensors: Misst den Anteil der Partikel, welche kleiner als 10 µm sind
- Steckplatz zum Anschluss an das Mikrokontrollerdboard: Bildet die Schnittstelle zwischen BeagleBone und Senorik Platine

2.3 Elektrische Konstruktion

Unser CanSat besteht aus mehreren Sensoren und einem zentralen Verarbeitungssystem sowie einem Sender, diese Kommunizieren alle über verschiedene Protokolle. Im Anhang unter der Einleitung befindet sich das Blockdiagramm unseres Satelliten. Im Blockdiagramm fehlen allerdings die verschiedenen Protokolle, in unserem Fall kommuniziert der BeagleBone Black, die MCU, mit allen Sensoren und hollt deren Daten ab.

Bauteil	**Kommunikationsprotokoll**
UV ML8511	ADC
Sharp Feinstaubsensor	ADC
APC220	UART
Ultimate GPS	UART
TMP006	I^2C
BMP180	I^2C

Tabelle 1: Kommunikationsprotokolle

2.3.1 Fachliche Grundlagen

2.3.1.1 Embedded System Ein Embedded System ist in unserem Fall der BeagleBone mithilfe vom ARM Cortex-A6 mit 1Ghz, ist ein leich modifizierte linux kernel mit Frontend installiert, dass liebevoll Angstrom genannt wurde. Um ein paar andere Beispiele für ein eingebettetes System sind etwa ein Smart TV oder ein Router, beide haben eine Art eingebettetes System, dass immer öfter auf dem Linux Kernel basiert und je nach Anwendung angepasst wurde. In unserem Fall unterstützt das BeagleBone verschiedene Technologien zum empfangen von Daten verschiedener Bauteile, wie etwa UART, I-2-C, SPI, Analog, Digital, PWM, Timer und PRU. Viele dieser Technologien sind in unserem Projekt nicht in Verwendung alle anderen Grundlagen sind unten beschrieben.

2.3.1.2 Transistor-Transistor-Logik 5V werden immer als logische 1 bezeichnet, damit ist gemeint wenn der Sensor den höchsten Messwert erreicht gibt er eine Spannung von 5V. Ist dies nicht der Fall hat der Sensor eine andere Kennkurve die zum Beispiel bei 3.3V aufhört. Allgemein wird aber Transistor-Transistor-Logik genutzt, welche 5V als logische 1 und geerdet als logische 0 ansieht, es gibt natürlich Toleranzen, diese sind aber bei verschiedenen integrierte Schaltkreisen und Mikrokontrollern unterschiedlich.

2.3.1.3 Analog-to-Digital-Converter Andere Sensoren wie der UV-Sensor die nur über einen internen Wiederstand verfügen der sich, je nach Konzentration, an einer mathematischen Kurve orientierend, im Wert leicht verändert und dadurch die ankommende Spannung am jeweiligen Analog Pin ändert. Mithilfe eines Analog-to-Digital-Converter konvertieren wir das analoge Signal, zum Beispiel 5V, in das äquivalente digitale Signal mit der Auflösung von 12 Bits.

$$2^{12} \quad = 4096$$

4096 verschiedene Stufen können wir also darstellen, wobei ein analoges Signal, theoretisch, unendlich Stufen bietet. Jede Veränderung könnte noch so klein sein. Ein einzelner Schritt ist trozdem im digitalen (12 Bits) sehr klein.

$$\frac{5V}{4096} = 0.001220703125V$$

Das bedeutet jeder 0.001220703125V kann dargestellt werden, wobei der Arduino Mega 2560 nur 10 Bits zur Verfügung stellt.

$$2^{10} \quad = 1024$$

$$\frac{5V}{1024} = 0.0048828125V$$

Das BeagleBone kann den Wert der am analogen Pin ankommt viel genauer darstellen, als der Ardunio.

2.3.1.4 Universal-Asynchronous-Receiver-Transmitter UART ist eine digitale serielle Schnittstelle zum realisieren von einfachen Kommunikationen zwischen zwei Endpunkten, die Funktionsweise ist denkbar einfach. Wir nutzen in unserem Satelliten meist eine Baudrate von 9600bps, Baud ist die Schrittgeschwindigkeit oder Symbolrate, also 9600 bits per second. Für UART gibt es wie beim RJ45 Stecker TX und RX, die beim Aufbau einer Kommunikation gekreuzt

werden. Transciever und Reciever. Nun wird zwischen vielen verschiedenen Arten von UART unterschieden in unserem Fall die TTL-UART Variante welche die beim Analog-to-Digital-Converter genannten 5V als logische 1 bezeichnen.

2.3.1.5 Inter-Integrated-Circuit I-2-C ist ein serieller Datenbus der über zwei Kabel mit einer 10-Bit-Adressierung, 1024 IC's steuern kann mit einer maximalen Geschwindigkeit 5 Mbit/s. Der Sinn des Bussystems ist es mithilfe von einer Adressen einen Datensatz oder Befehl nur an den gewünschten Empfänger zu senden, obwohl nur eine Datenleitung genutzt wird, eine Art Master/Slave System. Der Master sagt wer wann zu sprechen hat und welche Befehle von wem zu empfangen sind.

2.3.2 Sensorik

2.3.2.1 ML8511 - UV Sensor Der UV Sensor bietet im inneren nicht viel, eine Fotodiode welche auf UV-A und UV-B reagiert und einen Verstärker damit die minimale Veränderung auch gemessen werden kann. Die Fotodiode ändert je nach Einstrahlung von UV-A und -B ihren Wiederstand, die dabei enstehende Veränderung in der Spannung ist messbar.

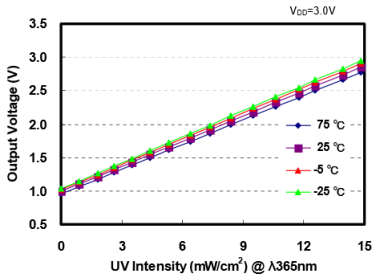

Abbildung 1: Spannungsausgabe vs. UV Intensität

2.3.2.2 Sharp Feinstaubsensor Der Sharp Feisensor arbeite, wie der ML8511, sehr simpel. Im inneren befindet sich eine Infarot Diode welche die Partikel anstrahlt, auf der anderen Seite befindet sich ein Fototransistor welcher dann feststellt wieviel von diesem Licht von Partikeln reflektiert wird, diese Veränderung ist messbar.

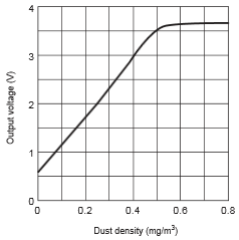

Abbildung 2: Spannungsausgabe vs. Staub

Um den Fototransistor nicht immer ganz zu bestrahlen, ist die Infarot Diode nicht die gesamte Zeit angeschaltet sonder nur einen Bruchteil einer Sekunde, die Messung muss inerhalb dieser Zeitspanne passieren.

2.3.2.3 APC220 Der APC220, ist ein Transciever welcher entweder senden oder empfangen kann. Der BeagleBone Black schickt den formatierten JSON String per UART an den APC220,

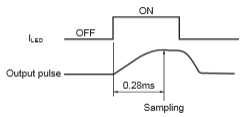

Abbildung 3: Zeitspanne einer Messung

dieser schickt ihn über die vorher am Computer festgelegte Frequenz in den Raum weiter. Am Boden befindet sich ebenfalls ein APC220, welcher die Daten empfängt.

2.3.2.4 Ultimate GPS Der Ultimate GPS von Adafruit verbindet sich mit den allgemeinen GPS-Satelliten und sendet Serial seine Daten im NMEA Format aus, aus diesen können wir dann verschiedene Daten auslesen:

Nummer	Daten
1	Breitengrad
2	Längengrad
3	Zeit
4	GPS-Qualität
5	Anzahl benutzter Satelliten
6	Höhe

Tabelle 2: Auslesebare Daten

Außerdem können wir den Ultimate GPS über die serielle UART Schnittstelle konfigurieren um ihn zum Beispiel nur ein bestimmtes NMEA Format ausgeben zu lassen oder um die Bitrate der seriellen Übertragung zu ändern.

2.3.2.5 TMP006 Der TMP006 ist ein Infarot Temparatur Sensor, welcher die Temperatur von einem Objekt misst ohne in direktem Kontakt zu stehen. Der Sensor misst die Temperatur eines Objektes anhand der ausgestrahlten Energy auf Wellenlänge von 4 micrometers bis zu 16 micrometers. Durch die veränderte Spannung am Sensor ist eine Messung der Temperatur möglich. Je größer das Objekt umso weiter entfernt muss es sich befinden um vom Sensor Field of View erfasst zu werden.

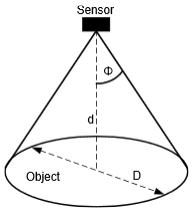

Abbildung 4: Sensor Field of View

Die Messung kann sehr ungenau werden da je nach Aussentemperatur und Temperatur der Sensorfläche selber Fehler beim Messen enstehen können.

2.3.2.6 BMP180 Der BMP180 ist ein Drucksensor welcher mithilfe einer Membran den Druck misst und per On-Board Controller diesen direkt in die Höhe umrechnet. Der Sensor sendet die Daten dann per I^2C-Bus an uns weiter.

2.3.3 Energieverbrauch

Bauteil	Stromaufnahme	Spannung	Leistungsaufnahme
Beaglebone Black	500mA	5V	2500mW
ML8511	<1mA	3.3V	<3.3mW
TMP006	<1mA	3.3V	<3.3mW
Sharp	20mA	3.3V	66mW
BMP180	<1mA	3.3V	<3.3mW
Ultimate GPS Modul	25mA	3.3V	82.5mW
APC220	35mA	5V	175mW
			2833.4mW

Tabelle 3: Energieverbrauch

2.4 Softwaredesign

2.4.1 Python als Programmiersprache

Als Programmiersprache zur Programmierung des Beaglebone Black's, haben wir uns für die Programmiersprache Python entschieden. Es wäre zwar ebenfalls möglich gewesen den Mikrocontroller mit den Sprachen JavaScript, Java, C, C++, C# und vielen weiteren Sprachen zu programmieren, da es sich bei dem Beaglebone um ein embedded-System handelt, welches praktisch alle Programmiersprachen unterstützt, sofern entsprechende Librarys existieren. Allerdings haben wir uns aufgrund der Tatsache, dass Python im Gegensatz zu Java nicht objektorientiert geschrieben werden muss, und wir auf der Hardwareseite möglichst auf objektorientierte Programmierung verzichten wollen, da sie nicht nötig ist, für Python entschieden. Ein weiteres wichtiges Argument war die gute Python-Library, welche von einer großen Community permanent gewartet und aktualisiert wird.

2.4.2 Datenverarbeitung auf dem Beaglebone

Die Datenverarbeitung auf dem Beaglebone verläuft relativ simpel. Zunächst werden alle von den Sensoren aufgezeichneten Daten, bei Sensoren mit I2C Anbindung mithilfe von Librarys und bei den anderen mithilfe von Umrechnungsalgorithmen, gesammelt. Anschließend werden alle gesammelten Daten in einen JSON-String geparsed, welcher mithilfe unserer Antenne an die Bodenstation übermittelt wird. Diese übernimmt die weitere Verarbeitung und Darstellung der Messdaten.

2.5 Bergungssystem

Für unseren Ladensystem haben wir uns entschieden unseren eigenen Fallschirm zu bauen. Die Hauptaufgabe ist es, was einen Fallschirm ausmacht, eine weiche Landung auf dem Boden. Was es für uns schwierig gemacht hat, war die gestellte Fallgeschwindigkeit von 15 Metern/Sekunde. Unsere Testfallschirme, die wir auch schon im Vorjahr genutzt haben, waren für eine deutlich geringer Fallgeschwindigkeit ausgelegt. Dieses Jahr wollen wir in CanSat 2015 eine neue Art eines Fallschirms testen. Wir haben die normalen 8 Schnüre zum Befestigen des Fallschirms mit der Dose mit sehr Luft durchlässigen Stoff ersetzt. Wir erhoffen uns dadurch eine stabilere Lage in der Luft und ein fortschrittlicheres Designe.

2.5.1 Bau

Beim Bau der Fallschirme haben wir Regenschirme benutzt. Diese sind bereits in einer Art Halbkugelform mit acht aneinandergefügten Panels. Dieser Stoff hat genug widerstand um den Belastung eines CanSat stand zu halten und ist sogar regendicht. Zuerst muss das Gestellt vorsichtig vom

Stoff herunter geschnitten werden. Danach muss in der Mitte des Stoff, wo alle 8 Kanten aufeinander Treffen ein ungefähr 5 cm großes Loch geschnitten werden. Nun kann der Fallschirm-Rohling in die entsprechende Größe zugeschnitten werden. Am Rand des Fallschirms sollte ein zusätzlicher ca. 1 cm Rand gelassen werden, der in den Fallschirm umgeklappt wird. Dieser muss dann mit einer Nähmaschine drauf genäht werden. Dies dient dazu die Struktur des Randes, durch Falten in den Innenbereichs besser zu schützen vor stärkeren Belastung. Am Rand befindet sich die höchste Wahrscheinlichkeit das durch eine kleine Kerbe ein kompletter Riss entstehen könnte. Nun kann auf dieser Grundlage die sehr Luftdurchlässigen Stoff drauf genäht werden. Durch zu hohe Belastungen am Angelpunkt mit dem der Dose sollte dort ein Stoff flicken drauf genäht werden, um die Belastung auf alle Seiten gleichmäßig zu verteilen.

2.5.2 Messungen

Um den Fallschirm auf die richtige Größe zu bekommen, haben wir eine Reihe von Test gemacht. Für uns waren die Berechnungen der Fallschirm Größe zu einfach und ungenau, deshalb haben wir nach alternativen Möglichkeiten gesucht. Unsere neue Idee war, die Größe mit Newton-Messer zu bestimmen. Der Fallschirm muss für die 15 Meter/Sekunde mit dem Gewicht von ca 250 Gramm eine Newton Wert von 12 haben. Die Messungen haben wir mit einigen Prototypen-Fallschirmen aus einem fahrenden Auto gemacht. Bei einer Geschwindigkeit von 50Km/h sollte die Prototypen diesen Wert so nah wie möglich kommen. Durch mehrere Fallschirmtest kann dadurch die Größe auf praktisch einen Meter genau bestimmt werden.

2.6 Aufgetretene Probleme

Während des Bau des CanSats sind selbstverständlich einige Probleme aufgetreten. Manche dieser Probleme waren relativ leicht zu lösen, andere erforderten eine detaillierte Recherche um sie zu beheben. Im nachfolgenden werden einige dieser Probleme und unsere Lösungsansätze erläutert.

- **Befestigung der Dosendeckel**: Es war vorerst geplant durchgängige Gewindestangen zu verwenden, welche durch die Wand führen und so die Deckel und die Wand miteinander verbinden. Diese Stangen kollidierten jedoch mit den Löchern für die Befestigung des Mikrokontorollers und der Sensorik Platine. Nach verhältnismäßig langem Überlegen und Ausprobieren haben wir uns dazu entschlossen die Gewindestangen nicht durchgängig zu machen. Stattdessen werden sie lediglich durch den Fuß und den Kopf der Wand gesteckt und dort verschraubt. Dies hat den Vorteil, dass wir das Gewicht deutlich verringern und wir innerhalb des CanSats wesentlich mehr Freiräume haben um Objekte zu platzieren. Nachteilig ist jedoch, dass dadurch die Stabilität verringert wird.

- **Spektrum des UV Sensors**: Unser UV Sensor misst die Intensität der Strahlung, welche im Spektrum 280-390 nm liegt. Dies hat die Folge, dass der Output des Sensors deutlich über dem zu erwartenden Wert liegt und es relativ schwer fällt einen Vergleich zwischen diversen Messungen aufzustellen. Dies liegt daran, dass man nicht verifizieren kann, welche Wellenlänge mit welchem Anteil an dem Gesamtoutput beteiligt ist.

- **Intensität des Sharp Feinstaub Sensors**: Zunächst war der Sharp Sensor dazu gedacht, die Feinstaubkonzentration in unserer Athmosphäre zu messen. Jedoch mussten wir feststellen, dass der Sensor keineswegs Feinstäube, sondern gröberen Staub wie zum Beispiel Hausstaub misst. Dies stellte deshalb ein Problem dar, weil wir bis dato das komplette Dosendesign auf den Sharp Sensor abgestimmt hatten. Ein neuer Sensor war zwar verfügbar, konnte jedoch aufgrund des Dosendesigns nicht integriert werden, da dieser zu groß ist.

- **Beaglebone Black**: Zu Beginn unserer Arbeit am Beaglebone, hatten wir einige Probleme mit diesem. Beispielsweise ließ sich zunächst die Python Library für das Beaglebone nicht vernünftig verwenden. Einige Pins ließen sich nicht vernünftig ansteuern was zu Fehlern führte. Letztlich fanden wir heraus, dass Linux Debian auf dem System installiert war. Dieses ist

jedoch noch in der Testphase und kann Bugs hervorrufen. Gelöst wurde das Problem indem der Speicher mit dem Defaultsystem Linux Angstrom geflasht wurde. Danach funktionierte das Ausführen von Pythoncode ohne jegliche Probleme.

- **UART und I2C-Bus**: Bei der Übertragung mithilfe von UART und dem I^2C-Bus sind wir auf Probleme mit BeagleBone gestossen, dieser hatte verschiedene Ports und Protokolle standardmäßig nicht aktiviert. Wir müssten im Linux-System Angstrom einige Startroutine hinzufügen, sodass bei jedem Start auch alle Ports und Protokolle aktiviert werden. Am Ende kostet dies blos viel Zeit und herumprobierei, da die Dokumentation von BeagleBone in diesem Punkt nicht detailgenau war und für verschiedene Versionen, verschiedene Lösungen des Problems exestieren.
- **Haltbarkeit des Gazestoffes**: Beim Bau des Fallschirms ist aufgefallen, das der Stoff der die Dose mit dem Fallschirm befestigt, eventuell nicht Stark genug ist. Durch die sehr löchrigen Struktur kann es auch schon durch einen Riss einer kleinen Stelle, der gesamte Stoff durch zu starke Belastung am Rissende zerreißen.
- **Durchmesser des CanSats**: Wir sind zu Beginn des Projektes davon ausgegangen, dass der Durchmesser des CanSats denen einer standardisierten Getränkedose entspricht (67mm). Zu spät ist uns aufgefallen, dass dies ein Irrtum war, und der Durchmesser 66mm betragen muss. Dies stellt jedoch kein Problem dar, da wir die Dicke der Außenwand problemlos um 0,5mm verringern können.

2.6.1 Testkonzept

Um sicherzustellen, dass der CanSat problemlos funktioniert wurden diverse Tests durchgeführt. Dazu zählt natürlich das Prüfen auf Funktionstüchtigkeit der Sensoren. Für dieses wurde jeder Sensor separat an verschiedene Mikrocontroller (Beaglebon Black, Arduino Mega) angeschlossen. Dadurch konnte verifiziert werden, dass jeder Sensor unter jedem Board den gleichen Output liefert. Zusätzlich wurde überprüft, ob die Sensoren auf eine Veränderung der zu messenden Eigenschaft reagieren. Um zu erkennen, ob die gemessenen Werte den tatsächlichen Werten entsprechen werden diese im Umweltlabor von Atlas Elektronik getestet und kalibriert. Dort soll ebenfalls überprüft werden wie stabil der CanSat ist um vorherzusagen, ob er beim Aufschlag beschädigt wird. Das Testkonzept für Sensoren verlies sich auf Trial and Error.

3 Beschreibung der Bodenstation

3.1 Desktop Applikation

3.1.1 Überblick

In diesem Teil der Dokumentation werden wir die Bodenstation vorstellen, welche als Datenempfänger und als Datenverarbeitungsplattform fungiert.
Die Bodenstation wurde von Robin Bley, Marc Huisinga und Kevin Neumeyer entwickelt.

Die zentrale Aufgabe der Bodenstation ist es, die Daten, welche vom Satelliten gesammelt werden, zusätzlich sicher am Boden zu speichern, sollte der Satellit und damit auch die lokal gespeicherten Daten verloren gehen.
Zusätzlich zur Datensicherung erfüllt die Bodenstation die Aufgabe, die empfangenen Daten auf verschiedene Arten zu visualisieren und somit dem Nutzer direkt während der Datenübertragung die Möglichkeit zu verschaffen, die Daten zu beobachten und diese zu analysieren.
Die Bodenstation ermöglicht es außerdem, dass gesicherte Daten auch nach der Datenübertragung noch betrachtet und analysiert werden können.
Unser Ziel bei der Entwicklung der Bodenstation war es, eine modulare und anpassbare Plattform zu entwickeln, welche nicht nur mit unserem Satelliten, sondern mit vielen verschiedenen Satelliten genutzt werden kann, ohne dass ein großer Konfigurationsaufwand besteht.
Um dies zu ermöglichen, haben wir die Bodenstation in mehrere Dimensionen skalierbar entwickelt, was es im Endeffekt sehr einfach macht, neue Satelliten und verschiedene Übertragungsprotokolle zur Bodenstation hinzuzufügen.

3.1.2 Verwendete Komponenten

Zum Erreichen unserer Ziele haben wir verschiedene Komponenten verwendet, welche einerseits der Datenvisualisierung und -analyse dienen, andererseits aber auch der Entkopplung und skalierbaren für die Entwicklung dienen.

Für die Bodenstation haben wir folgende Komponenten verwendet:

Java ist eine objektorientierte Programmiersprache. Diese wurde verwendet, da jedes unserer Gruppenmitglieder damit vertraut ist. Die Version Java 8 wurde verwendet um mächtige funktionale Features zu nutzen

Netbeans Platform das die Möglichkeit bietet, einfach eine integrierte, modulare und entkoppelte GUI-Applikation auf Basis von Java Swing zu entwickeln

JUnit ist ein Framework, welches zum Erstellen von automatisierten Softwaretests dient.

JSerialComm (zum Start des Projektes noch serial-comm) ist eine Bibliothek, welche das Auslesen serieller Schnittstellen ermöglicht

NASA World Wind ist eine Software, welche Satelliten- und Luftbilder auf einem virtuellen Erdball darstellt. Daten der Bodenstation werden mittels dieser Software in Relation zur Höhe in Echtzeit visualisiert.

JChart2D ist eine Grafik-Bibliothek, welche zur grafischen Visualisierung von Daten dient. Mithilfe dieser Bibliothek werden zweidimensionale Graphen erzeugt, welche empfange Daten des Satelliten, in Relation zur Zeit oder anderen Daten, in einem Graphen darstellt

JSON (JavaScript Object Notation) ist ein Datenformat, welches zum austausch von Daten zwischen Anwendungen angewand wird. JSON ermöglicht es Daten in verschiedenen Format in Textform zu speichern und sie wieder zurück in ihre uhrsprüngliche Form zu interpretieren. Dieses Datenformat wird in der Bodenstationssoftware genutzt um Daten mit dem Sateliten auszutauschen, zu loggen, zu exportieren und zu importieren.

Git ist eine freie Versionsverwaltungssoftware. Diese Software verwendeten wir um die Versionen unserer Software auszutauschen und zu verwalten.

3.1.3 Funktionen

3.1.3.1 Nutzerfreundlichkeit Die Bodenstation wurde so entwickelt, dass der Nutzer der Bodenstation sich nicht um Implementationsdetails scheren muss und die Bodenstation als zentralen Empfänger für Daten von seinem Satelliten nutzen kann, ohne dabei etwas anderes als die grafische Benutzeroberfläche zu verwenden.

Ein wichtiger Faktor im Bereich der Nutzerfreundlichkeit ist die dynamische Benutzeroberfläche, welche es dem Nutzer erlaubt, verschiedene GUI-Komponenten in Teilpanels innerhalb der Applikation anzuzeigen und umzustellen. Diese Dynamik ermöglicht es dem Nutzer, die Benutzeroberfläche, welche für die Analyse seiner Daten am Besten ist, mithilfe der verschiedenen Panels einzustellen.

Ein weiterer, wichtiger Faktor ist, dass der Nutzer die Bodenstation so anpassen kann, wie es für die Datenübertragung seines Satelliten am Besten ist. Alle Teilkomponenten der Datenübertragung sind über die grafische Benutzeroberfläche austauschbar, was es sehr einfach macht, die Bodenstation auf die Datenübertragung des jeweiligen Satelliten anzupassen.

Alle Visualisierungskomponenten sind zudem intuitiv aufgebaut, sodass es nicht kompliziert ist, sich seine Daten mithilfe der Visualisierungskomponenten anzusehen. Die Anzeige über den 3D-Globus erlaubt es beispielsweise, den Globus beliebig zu bewegen und die Position auf dem Globus zu verändern, während der Graph es erlaubt, dass die Axen des Graphen beliebig ausgetauscht werden können.

3.1.3.2 Erweiterbarkeit Bei der Entwicklung der Bodenstation haben wir darauf geachtet, dass die Bodenstation auf einer skalierbaren Architektur aufgebaut ist. Dies ermöglicht es, leicht neue Module und Funktionalitäten zur Bodenstation hinzuzufügen, ohne dabei besonders viel Code abzuändern. Auf die folgenden Weisen ist die Bodenstation skalierbar:

- Neue GUI-Komponenten können sehr leicht hinzugefügt werden. Das Erstellen und Einbinden eines GUI-Komponenten umfasst lediglich die Erstellung eines neuen Netbeans-TopComponents.
- Unterschiedliche Satelliten können ohne eine erneute Kompilierung hinzugefügt und verändert werden, indem man die Konfigurationsdateien der Bodenstation anpasst, welche zur Laufzeit von der Bodenstation geladen werden.
- Neue Konfigurationsformate können leicht hinzugefügt werden, indem man die neue Konfigurationsimplementation unter einem Interface in der Applikation hinzufügt.
- Es ist ohne viel Aufwand möglich, die verschiedenen Datenquellen, aus denen Daten bezogen werden, auszutauschen. Möchte man also die Daten von einer anderen Quelle als einem USB-Port beziehen, so ist dies leicht zu implementieren, indem man ledigliche eine neue DataSource implementiert und zur Applikation hinzufügt.
- Verschiedene Datenübertragungsformate können ebenfalls ohne viel Aufwand hinzugefügt werden, indem neue Formate unter dem DataFormat-Interface implementiert werden. Die Applikation ist also nicht auf JSON als Übertragungsformat limitiert.
- Die verschiedenen Datenempfänger können auch leicht angepasst werden, indem man neue Datenempfänger unter dem Receiver-Interface implementiert, wodurch die verschiedenen Logging-Formate erweitert werden können.
- Daten können aus beliebigen Dateiformaten importiert werden, was ebenfalls leicht erweiterbar ist, indem man neue Import-Dateiformate über das Importer-Interface implementiert.
- Export-Formate können leicht erweitert werden, indem man neue Export-Formate unter dem Exporter-Interface implementiert.

3.1.3.3 Features Da die Software der Bodenstation auf dem Framework Netbeans Plantform basiert, lassen sich einzelne graphische Module kombinieren, welche sich per drag and drop verschieben lassen. Die Größe und Position dieser Module und des gesamten Frames lassen sich beliebig verändern. Des weiteren bietet die Software die Möglichkeit, Daten von verschiedenen Satelliten zu empfangen. Empfangene Daten lassen sich mittels der graphischen Oberfläche in Graphen anzeigen, welche sich verschieden kombinieren lassen. Außerdem lassen sich die empfangenen Daten zwischenspeichern und anschließend in verschiedene Dateiformate exportieren oder live in einer Datei loggen. Diese exportierten oder geloggten Daten lassen sich anschließend wieder einlesen und anzeigen. CSV, TXT, JSON, KML und PNG sind Dateiformate, welche exportierbar sind. Davon lassen sich exportierte CSV- und JSON Dateien wieder einlesen und visualisieren. TXT-Dateien werden formatiert und somit gut leserlich für den Nutzer exportiert, während exportierte KML-Dateien per Google Earth geöffnet und graphisch visualisiert werden können. Ein weiteres Feature der Bodenstationsoftware ist die Datenvisualisierung per Nasa World Wind als Modul in der graphischen Oberfläche. Diese Visualisierung zeigt den Flug des Satelliten auf einem virtuellen Globus, mittels Satelliten- und Luftbilder, und zeigt auf jeder gemessenen GPS-Koordinate die gemessenen Werte der Sensoren des Satelliten. Unter anderem bietet dieses Modul der Software die Möglichkeit, an die virtuelle Erdkugel heranzuzoomen und einzelne Elemente dreidimensional darzustellen. Darstellungen mittels dieses virtuellen Erdballs sind sowohl in Echtzeit mittels eines Streams vom Satelliten als auch als Import aus einer Datei möglich.

3.1.4 Architektur

Insgesamt ist die Architektur der Applikation um das GUI-Framework Netbeans Platform aufgebaut, da es die Nutzung von bestimmten Architekturen einfacher macht, mit Netbeans Platform zu arbeiten und alle Features von Netbeans Platform zu nutzen.

Insgesamt ist die Applikation in Module und Pakete aufgeteilt. Während normale Java-Projekte normalerweise lediglich in Pakete aufgeteilt sind, werden in Netbeans Platform die einzelnen Komponenten normalerweise in Module aufgeteilt. Jedes Modul verhält sich hierbei wie ein einzelnes Projekt, welches dann vom Hauptprojekt eingebunden wird. Dies fördert generell die Wiedernutzbarkeit der einzelnen Module, da sich Entwickler darüber Gedanken machen müssen, wie die einzelnen Module in verschiedenen Umgebungen verwendet werden können.
Den Aufbau der Netbeans Platform Modularchitektur ist auch im Anhang unter Abbildung 13 zu finden.

Anfänglich haben wir jeden einzelnen Teilkomponenten in ein Modul ausgelagert, was jedoch zu einer Menge Merge-Konflikten geführt hat, da Netbeans Platform für jedes einzelne Modul eigene Konfigurationsdateien generiert, über welche man nur schwer den Überblick behalten kann. Diese anfänglich Architektur wurde schlussendlich in eine Architektur mit nur drei Modulen umgewandelt: API, Core und GUI.
Das Core-Modul enthält die Programmlogik, welche sich hauptsächlich mit der Verarbeitung der Daten innerhalb der Input-Pipeline beschäftigt.
Das GUI-Modul enthält die verschiedenen GUI-Komponenten, welche sowohl Teil der allgemeinen Benutzeroberfläche sind, als auch Visualisierungskomponenten darstellen.
Innerhalb des API-Moduls befinden sich Interfaces und Utility-Klassen, welche sowohl vom Core-Modul als auch vom GUI-Modul genutzt werden. Das Core-Modul implementiert hierbei die Interfaces aus dem API-Modul, während das GUI-Modul die Programmlogik im Core-Modul lediglich entkoppelt über die Interfaces des API-Moduls anspricht.
Diese Architektur zeigt bereits, dass das GUI-Modul vom Core-Modul entkoppelt ist. Diese Entkopplung trägt stark zu der Erweiterbarkeit der Applikation bei. Die Architektur ähnelt hierbei der standardmäßigen “Model, View, Controller”-Architektur, jedoch scheint innerhalb der Architektur kein Controller vorhanden zu sein. Auf den ersten Blick gesehen scheint es so, als spreche das GUI-Modul das Core-Modul direkt über das API-Modul an, jedoch sorgt Netbeans-Platform dafür, dass dem nicht so ist. Die von Netbeans Platform bereitgestellten Lookups, welche eine weitere Ebene der Entkoppelung darstellen, erfüllen in der Applikation die Aufgabe des Controllers. Durch die Lookups wird innerhalb des GUI-Moduls eine passende Klasse innerhalb des

Core-Moduls über das Interface der Klasse im API-Modul geladen. Dank den Lookups und den Interfaces im API-Modul besteht absolut keinerlei Kopplung zwischen den einzelnen Klassen und Modulen: Die GUI kennt das Model nicht und das Model kennt die GUI nicht.
Die Modularchitektur der Bodenstation ist ebenfalls im Anhang unter Abbildung 14 zu finden.

Weitergehend relevant ist die Architektur der Input-Pipeline, welche in der Bodenstation dafür zuständig ist, die empfangenen Daten zu verarbeiten, bevor sie an die jeweiligen Komponenten zur Endbehandlung der Daten weitergeleitet werden. Insgesamt wandelt die Input-Pipeline die aus einem Datenstream empfangenen Daten in einen Map-Datentypen um, welcher genau einem Datensatz entspricht, und leitet die Daten an alle registrierten Empfänger weiter. Die Schlüssel der Map entsprechen den Schlüsseln der Datenübertragung und die Werte der Map entsprechen den Daten, welche zu dem jeweiligen Schlüssel gehören. Zusätzlich zu dieser Datentransformation sorgt die Input-Pipeline ebenfalls dafür, dass fehlerhafte und fehlende Daten herausgefiltert und über die zuletzt erhaltenen Daten abgeflacht werden.
Die Input-Pipeline ist Push-basiert, was bedeutet, dass jeder Komponent immer die verarbeiteten Daten an den nächsten Komponenten weitergibt, damit zwischen den einzelnen Komponenten nicht gepollt werden muss. Die Input-Pipeline besteht insgesamt aus vier Komponenten. Am Anfang der Input-Pipeline steht eine DataSource, welche die Daten von einer beliebigen Datenquelle via Polling bezieht und die ausgelesenen Daten an ein DataFormat weitergibt. DataFormat sorgt dafür, dass der von einer DataSource empfangene String mit dem jeweiligen Übertragungsformat, zum Beispiel JSON, geparst wird, um so einen Datensatz als Map zu erzeugen. DataFormat leitet die geparsten Daten dann an einen DataProvider weiter, welcher die fehlenden Daten herausfiltert, abflacht, und an alle bei ihm registrierten Empfänger weiterleitet. Bei der Abflachung der Daten merkt sich der DataProvider immer die zuletzt empfangenen Daten und ergänzt fehlende Daten mit den zuletzt empfangenen Daten. Sind noch keine Daten empfangen worden, so werden die Daten mit Default-Werten ergänzt.
Insgesamt wird die Input-Pipeline von einer DataPipeline umschlossen, welche die Input-Pipeline zusammenbaut, die verschiedenen Komponenten austauscht, die verschiedenen Empfänger beim DataProvider registriert und allgemein als Schnittstelle zu den verschiedenen Empfängern und dem GUI-Modul fungiert.
Der Aufbau der Input-Pipeline ist ebenfalls im Anhang unter Abbildung 15 zu finden.

3.1.5 Tests

3.1.5.1 Automatisierte Tests Einzelne Komponenten der Software werden mittels automatisierten Tests per JUnit getestet. Dabei werden bei den jeweiligen Export-Komponenten jeweils Testdaten in das jeweilige Format exportiert. während dessen kann überprüft werden ob sich die Komponente, bei der Übergabe verschiedener Parameter, wie geplant verhält. Außerdem kann geprüft werden, ob die erzeugten oder veränderten Dateien wie geplant aussehen. Darüber hinaus wurde für jeden Import-Komponenten ein automatisierten Test geschrieben, welche den Komponenten auf das verhalten bei verschiedenen Parametern überprüft. Des weiteren werden Daten erzeugt, welche zunächst mit dem jeweiligen Komponenten exportiert werden und anschließend mit dem passenden Komponenten importiert werden. Dabei wird geprüft, ob sich die importierten Daten von den ursprünglichen Daten unterscheiden.

3.1.5.2 Manuelle Tests

3.1.6 Nutzeranleitung

3.1.6.1 Datenempfang Um Daten in Echtzeit zu empfangen muss eine Verbindung zu einem Satelliten aufgebaut werden. Um diese Verbindung aufzubauen wählt man unter dem Menüpunkt File den Unterpunkt Satellites aus. Dort ist es möglich unter add einen Satelliten hinzuzufügen. Wenn nun Daten empfangen werden sollen wählt man im selben Unterpunkt manage aus. Dort wählt man den Satelliten aus, von welchem man Daten empfangen will. Anschließend startet der Datenempfang.

3.1.6.2 Datenimport Um Daten aus einer Datei zu importieren wird Im Menüpunkt File der Untermenüpunkt Import Ausgewählt. Anschließend wird eine Datei ausgewählt, welche importiert werden soll. Mit der Bestätigung werden die Daten dieser Datei eingelesen.

3.1.6.3 Datenexport Für das Exportieren der Daten gilt, dass alle aktuell geladenen Daten exportiert werden. Darunter fallen entweder zwischengespeicherte Daten einer Liveübertragung oder Daten, welche aus einer Datei importiert werden. Zum Exportieren der Daten wird im Menüpunkt File der Untermenüpunkt Export ausgewählt. Unter diesem Menüpunkt ist das Datenformat wählbar, in welches die gesammelten Daten gespeichert werden. Anschließend ist ein Pfad und eine Name wählbar unter dem die Datei gespeichert wird. Mit der Bestätigung werden die Daten exportiert.

3.1.6.4 Datenweiterleitung Per klick des auf das Icon des Servers in der Toolbar wird ein Server gestartet, empfangene Daten in Echtzeit an alle Clients versendet.

3.1.6.5 Oberflächenpersonalisierung Der Oberfläche können einzelne Komponenten hinzugefügt und entfernt werden. Diese Komponenten können unterschiedlich angeordnet werden. Um Komponenten wie z.B einen Graph hinzuzufügen wird entweder eine Graphvisualisierung oder eine Kartenvisualisierung hinzugefügt. Um einen der bestehenden Komponenten zu entfernen wird das Kreuz angeklickt, welches sich am Tab des Komponenten befindet. Per "drag and drop"können diese Komponenten neu angeordnet werden, dazu muss der Tab des Komponenten ausgewählt werden. In verschiedenen Bereichen können Komponenten angeheftet oder verschoben werden. Außerdem können diese übereinander verlagert werden um diese, in verschiedenen Tabs und in dem selben Bereich, zu verwalten.

3.1.6.6 Kartenvisualisierung Die Kartenvisualisierung startet über den Untermenüpunkt Map Vizualization im Menüpunkt Window. Geladene Werte werden dort angezeigt. Einzellen Messpunkt sind mit einem Punkt gekennzeichnet und mit einer Linie verbunden. Die mit Punkten gekennzeichneten Messwerte sind mit der linken Maustaste anklickbar. Mit einem Klick öffnen sich Details zu den angeklickt Messwerten.

3.1.6.7 Graphvisualisierung Um einen Graph zu erzeugen wählt man unter dem Menüpunkt Window Vizualization aus. Anschließend wird in der Oberfläche ein Graph erzeugt. Die Achsen des Graphs sind mit Sensorwerten belegbar. Um Belegung der Achsen zu verändern werden wählt man an den Achsen den jeweiligen Sensor aus und drückt den Button Ansicht aktualisieren.

3.1.6.8 Fenster zurücksetzten Die Anordnung der Komponenten der Oberfläche können im Menüpunkt window unter Reset Windows zurückgesetzt werden.

3.1.6.9 Beenden des Programms Um das Programm zu beenden gibt es zwei Möglichkeiten. Zum einen wird das Programm beendet, wenn das Kreuz am oberen rechten Rand der Oberfläche angeklickt wird. Zum andern kann das Programm über den Menüpunkt File geschlossen werden, in dem man dort Exit auswählt.

3.1.7 Realisierung der Desktop-Software

Während der Realisierung der Bodenstation kam es zu einigen Komplikationen. Zum einen wurde die Softwarearchitektur während der Implementationsphase geändert, so dass verschiedene Komponenten wie zum Beispiel Export- und Importkomponenten mehrfach realisiert wurden. Diese Architekturveränderung wurde vorgenommen und gewisse Komplikationen zu beseitigen, welche die Modulare Strukturierung von Netbeans Plattform mit sich bringt. Wir starteten mit einer Architektur, welche jede wichtige Komponente als Modul benennt. Diese Architektur brachte zum einen das Problem, dass Abhängigkeiten zwischen Modulen nur in eine Richtung stattfinden kann. Die

neue Architektur unterscheidet lediglich zwischen den Modulen API, GUI und Core. Die gesamte Architektur der Software zu ändern kostete viel uns viel Zeit. Des weiteren wurden verwendete Datentypen innerhalb der Software während der Implementationsphase geändert, so dass zusätzlich alle Komponenten, welche mit der Datenverarbeitung Zutun haben geändert werden mussten. Darunter vielen die gesamten Komponenten des Imports, Exports und der Live-Datenverarbeitung. Die Veränderung der benutzten Datentypen wurde von Long, String und Double zu einzig Double geändert, da alle Daten, welche von kompatiblem Satelliten in Double dargestellt werden können. Diese Änderung im Programmcode hat zwar einiges an Arbeit gekostet, doch der Umfang des Programmcodes wurde deutlich verringert und die Performance gesteigert. Am Anfang der Realisierung der Software, verlief die Versionsverwaltung mit Git nicht wie geplant. Nach fast jedem Versionsaustausch gab es Konflikte in unserem Projekt. Dies wurde hervorgerufen durch mangelnde Erfahrung mit Netbeans Plattform. Einzelne Dateien des Projektes werden ständig durch die verwendete IDE Netbeans, was uns nicht bekannt war. Dies wurde durch eine Vielzahl von Branches verkompliziert. Diese Konflikte zu lösen und das Problem endgültig zu beheben hat mehrere Stunden Zeit gekostet.

3.2 Android Applikation

3.2.1 App Übersicht

Auch dieses Jahr haben wir uns für die Erstellung einer App entschieden. Die Hauptaufgaben der App soll sein, die Datenpakete, die von der Bodenstation über einen Hotspot gesendete werden, zu empfangen und Live, in einem passenden Graphen anzeigen zu können. Dabei legen wird Wert drauf, das es möglich ist, alle Werte die gesendet werden einzeln oder in Gruppen darzustellen sind, um sie anhand der Daten vergleichen zu können. Zusätzlich haben wir uns überlegt, falls die Zeit reicht, auch die Werte in einem Balkendiagramm anzeigen zulassen. Dabei sollen die Werte nicht einfach, wie im Livegraphen live angezeigt werden, sondern es soll die Differenz ausgerechnet werden, von dem höchsten und niedrigsten Punkt im Flug. Diese Differenzen sollen dann Grün für positive Veränderung und Rot für negative Veränderung dargestellt werden. Neben her sollen ebenfalls einige Optionsmöglichkeiten vorhanden sein, um die Graphen nach den Individuellen wünschen des Nutzern zugestalten.

3.2.2 Plattform und Komponenten

Für die Entwicklung der App haben wir uns der Plattform von Androidstudio bedient. Durch eine übersichtlich Anordnung von Fenstern und Struktur ist dies für uns ein ideales Programmierumgebung. Androisstudio ist auf Java ausgelegt, das wiederum eine groß Zahl von Features mit sich bringt. Aber allein auf dieses Programm konnten wir uns nicht verlassen, es brauchte noch zwei weiter Komponenten, um den Graphen anzeigen zu lassen und das JSON zu entpacken. Dazu haben wir uns der AndoridPlot-Libary und einer der Liberies über JSON zur Hilfe genommen.

3.2.3 Funktionen

Für die App haben wir uns für folgende Funktionen vorgenommen:

- Anzeigen der Werte in einem Livegraphen
- Verwaltung der angezeigten Werte im Graphen während der Laufzeit
- Anzeigen von Differenzen von Werten im Balkendiagramm in Rot und Grün
- Manuelle Start/Stop Funktion für den Balkendiagramm
- Einstellung zur Geschwindigkeit des Graphen
- Einstellung wie viele Werte gleichzeitig angezeigt werden sollen
- Möglichkeit alle Funktionen mit einem Debugger zu testen

3.2.4 Nutzeranleitung

Um die Team-Gamma App nutzen zu können, brauche der Benutzer unser apk.-Datei. Diese werden wir unter den gegebenen unterständen, vor dem Raketen Start, Vorort verteilen. Um diese dann zu installieren zu können, braucht es ein geeigneten Dowloadmanager der es erlaubt die apk von dort aus zu installieren. Nach der erfolgreichen Installation der App kann man nun durch betätigen des Icons der App, diese dann, ganz wie gewohnt, starten. Davor sollte drauf geachtet werden, das um die Daten von der Bodenstation empfangen zu können, muss sich der Nutzer sich vorher in den zur Verfügung gestellten Hotspot einloggen. Dieser wird ohne Password in der nähe unserer Bodenstation erreichbar. Wir wollen aber noch mal ausdrücklich drauf Hinweisen, das dieser Hotspot keine Internet Verbindung bietet, sondern nur auf lokaler Basis arbeitet. Nach öffnen der App sollte nach kurzer Zeit das Menü erscheinen, das die App in 3 Sektionen unterteilt:

3.2.4.1 Livegraph Beim an tippen des Graphen-Icons erscheint ein graues Rechteck mit einigen Linien. Bei standardmäßigen Einstellungen, sollte fürs erste keine Werte angezeigten werden. Nur falls sich das Gerät in der nähe unseres zur Verfügung gestellten Hotspot befindet und genug Empfang hat werden nach einigen Sekunden automatisch Werte von der Bodenstation live angezeigt. Diese sollten aber während der Vorbereitungsphase nur gerade Linien darstellen. Nun kann, durchs drücken der Menütaste des Smartphones, eine Fenster erscheinen lassen, das dazu genutzt werden kann, explizite Werte anzeigen zu lassen. Dabei ist zu beachten, das auch mehrere Werte gleichzeitig im Graph darstellbar sind.

3.2.4.2 Balkengraph Im Balkengraph angekommen sollte dort ebenfalls ohne den Debugger nichts zu sehen sein. Im Normalfall sollte, wenn die Verbindung mit der Bodenstation steht, der Graph starten, wenn der CanSat über die Marke von 1000 Meter ist. Falls durch irgendwelche Defekte der Racket diese Marke nicht zu erreichen sei, kann man Manuelle durch betätigen des Menüknopf des Smartphones ein Fenster erscheinen lassen, wo man diesen manuell starten und stoppen kann. Der Graph wird dann den ersten Wert als ersten Punkt nehmen und den letzte gesendet Punkt als zweiten. Die Differenz wird dann positiv in Grün oder negativ in Rot dargestellt. Beim verlassen des Balkendiagramms wird dieser im Hintergrund weiter ausgeführt. Bei einer Marke von ungefähr 0 Metern, sollte der Graph sich selber beenden und die Werte solange anzeigen, bis sie nicht mehr benötigt werden. Beim Beenden des Programm sind diese aber unwiderruflich verloren!

3.2.4.3 Optionen Die Option in der App geben die Möglichkeit den Livegrafen schneller oder langsamer zu machen. Außerdem wird dort auch ermöglicht, wie viele Werte gleichzeitig angezeigt werden sollen. Für das Testen und Anzeigen von Werten gibt es dort zusätzlich auch einen Knopf um den Debugger einzuschalten.

4 Projektplanung

4.1 Zeitplan der CanSat Vorbereitung

Die Zeitplanung ist ausgerichtet für den Zeitpunkt der Abgabe unseres P5, da wir uns gewünscht haben, zu diesem Zeitpunkt mit dem Projekt fertig zu sein. Dieser Zeitplan wurde jedoch von Anfang an sehr kritisch gesehen. Daher ist es nicht verwunderlich, dass der Fortschritt des Projektes geringer ist, als er zum jetzigen Zeitpunkt eigentlich seien sollte. Dies ist jedoch nicht dramatisch, da bis zum Wettbewerb genügend Zeit ist die restlichen Arbeitspakete abzuarbeiten. Das gesamte Management der Arbeitspakete und des Zeitaufwandes wurde mit der Projektmanagementsoftware Redmine erledigt. Da diese auf unserem Server unter redmine.gamma-team.de erreichbar ist kann jedes Teammitglied zu jedem Zeitpunkt den Fortschritt der Arbeit verfolgen. Die Planung der beiden Halbgruppen ist größtenteils voneinander getrennt. Es gibt jedoch gemeinsame Meilensteine, welche von beiden Gruppen eingehalten werden sollen. Bevor die Arbeit der Halbgruppen begonnen hat gab es eine allgemeine Projektfindungsphase. In dieser Phase wurde ein grober Zeitplan festgelegt und es wurden alle relevanten Systeme (Webserver, Projektmanagementsoftware, GitLab etc.) aufgesetzt und eingerichtet um später einen reibungslosen Ablauf der Arbeitsphase zu garantieren. Die Idee und die Spezialisierung der Idee für das gesamte Projekt entstand ebenfalls in dieser Zeit. Anschließen wurde eine separate Zeitplanung in den beiden Halbgruppen erstellt, welche im Nachfolgenden erläutert wird.

4.1.1 Zeitplan der Hardware Gruppe

Innerhalb der Hardwaregruppe wurden versucht die meisten Aufgaben zu parallelisieren. Jedes Teammitglied hat sein eigenes spezielles Aufgabengebiet. jedoch herrscht trotzdem ein stetiger Austausch zwischen den Teammitgliedern. Grund für die Parallelisierung war, dass in unseren Augen die meisten Aufgaben nur die Aufmerksamkeit einer Person benötigen. Es ist nur selten erforderlich, dass mehrere Teammitglieder an ein und dem selben Arbeitspaket arbeiten. Der gesamte Arbeitsprozess wurde in diverse Abschnitte gegliedert. Diese Abschnitte lassen sich auch im GANTT Diagramm im Anhang dieses Dokumentes wiederfinden. Bei den Abschnitten handelt es sich um folgende:

- Planung: Erstellung von Arbeitspaketen, sowie eine Verteilung dieser und eine Erstellung diverser Diagramme
- Fallschirm: Gestaltung und Bau des Bergungssystems.
- Sensorik: Dieser Abschnitt behandelt das Heraussuchen, Bestellen und Testen passender Sensoren für unser Projekt.
- Beagleboard: Festlegung der Programmiersprache, IDE und der Recherche zu den elektrotechnischen Eigenschaften des Boards
- Dose: Design und Bau der Hülle und der Deckel der Dose
- Dosenmanagement: Design und Bau des inneren der Dose, sowie die Integration der Sensoren in das Gesamtsystem

Die einzelnen Abschnitte sind in diverse Arbeitspakete unterteilt, Personen zugewiesen und mit einem Zeitraum versehen.

4.1.2 Zeitplan der Software Gruppe

4.2 Einschätzung der Mittel

4.2.1 Budget

Um das CanSat Projekt zu finanzieren konnten wir aktuell noch keine Sponsoren finden. Jedoch konnten wir uns mit unserem Schulverein verständigen, welcher uns finanziell unterstützen wird.

Da wir nicht auf das T-Minus Kitt zurückgreifen sondern stattdessen ein anderes Mikrocontroller Board verwenden können wir ungefähr 150 €sparen. Der 200 €Watterot Gutschein, welcher vom Wettbewerb gestellt wird, ist in unseren Rechnung noch nicht inbegriffen. Dies liegt daran, dass noch nichts bei Watterot bestellt wurde, bzw. die Bestellung lange vor der Annahme am Wettbewerb getätigt wurde. Im Nachfolgenden sind alle Ausgaben und Einnahmen aufgelistet.

Ausgabe	**Datum**	**Empfänger**	**Grund**
-12,16 €	08.01.2015	Watterott	BMP180 Breakout
-28,99 €	09.01.2015	eBay - rcskymodel	Ultimate GPS
-14,32 €	10.01.2015	Spark Fun Electronics	UV-Sensor
-51,99 €	10.01.2015	Amazon	Beagle Bone Black
-17,30 €	01.12.2014	eBay - hdt-preiswert	GFK-Set 1kg Polyesterharz + 20g Härter + $2m^2$ Glasfasermatte
-3,54 €	23.03.2015	toom baumarkt	6 x Schleifpapier
-3,79 €	23.03.2015	toom baumarkt	Filzrolle
-4,49 €	23.03.2015	toom baumarkt	Plüschwalzen
-2,19 €	23.03.2015	toom baumarkt	Mundschutz
-1,99 €	23.03.2015	toom baumarkt	Farbwanne
-4,99 €	23.03.2015	toom baumarkt	Einmalhandschuhe
- 145,75 €			

Tabelle 4: Ausgaben

Einnahmen	**Datum**	**Absender**	**Grund**
17,30 €	01.12.2014	Alexander Brennecke	GFK-Kauf
107,46 €	10.01.2015	Alexander Brennecke	Sensorenkauf
20,99 €	23.03.2015	Alexander Brennecke	toom Einkauf
145,75 €			

Tabelle 5: Einnahmen

4.2.2 Externe Unterstützung

Externe Unterstützung erhielten wir von vielen Lehrern unserer Schule, welche uns Fragen zur Elektrotechnik und Softwareprogrmmierung beantworten konnten. Zusätzlich haben wir finanzielle Unterstützung durch den Schulverein unserer Schule erhalten (siehe 4.2.1). Unterstützung außerhalb unserer Schule erhileten wir durch folgende Personen/Organisationen:

- Das Hackerspace Bremen e.V., welches uns ihren 3D-Drucker zur Verfügung gestellt hat. Zusätzlich konnten wir dort unsere Platine ätzen.
- Prof. Martin Schneider von von dem Hochfrequenzlabor der Universität Bremen, welcher uns geholfen hat unsere Antenne an die Frequenz und die Wellenimpedanz anzupassen.
- Das Umweltlabor der Atlas Elektronik GmbH hat uns geholfen den CanSat, hinsichtlich seiner Stabilität, zu testen und die Sensoren korrekt zu kalibrieren.

5 Öffentlichkeitsarbeit

5.1 Website

Unsere Website Team Gamma wurde bereits für den europäischen CanSat Wettbewerb 2015 verwendet. Diese haben wir weiter geführt und dort in unregelmäßigen Abständen aktuelle Informationen über das Projekt veröffentlicht. Da die Website durch den europäischen Wettbewerb bei anderen europäischen Teams bekannt ist wird die Website in Englisch geführt. Man findet dort zusätzlich einige Dokumente, Fotos und Videos. Die Informationen auf der Website sind meist relativ detailliert verfasst.

5.2 Schülerzeitung

In der Schülerzeitung unserer Schule sind bereits diverse Artikel über unser Projekt erschienen und sollen auch in Zukunft erscheinen. Diese Artikel handeln zumeist von dem Wettbewerb selber und gehen weniger auf die technischen Details ein.

5.3 Präsentationen

Da wir das CanSat Projekt bereits seit einiger betreiben kommt es immer wieder vor, dass wir es vor unserer Klasse präsentieren. Dies kommt zum Beispiel dann vor, wenn wir Teile des Projektes in Schulprojekte einfließen lassen. Zusätzlich haben wir, beispielsweise am Tag der offenen Tür unserer Schule, diversen Schulbesuchern das Projekt und den Wettbewerb näher gebracht.

5.4 Ausstellung am MINT-Projekttag unserer Schule

Im Schuljahr 2015/2016 findet an unserer Schule ein Tag der MINT Projekte statt. Dieser Tag wird von einer Schülergruppe unserer Parallelklasse organisiert und wir wollen an diesem Tag natürlich unser Projekt vorstellen.

5.5 Logo

Das Logo wurde ebenfalls aus Gründen der Wiedererkennbarkeit aus dem vorherigen Jahr übernommen. Das Aussehen des Logos wurde von drei Faktoren beeinflusst:

- Das Zeichen in der Mitte soll dem Gamma Logo ähneln, welches zu unserem Teamnamen passt
- Das Zeichen soll zusätzlich, wenn man es um 180° dreht, dem Lambda Logo ähneln. Da bei dem Entwurf unserer Antenne immer wieder auf Lambda gestoßen sind, sind daraus diverse interne Späße entstanden, die wir in das Logo einfließen lassen wollen.
- Das Logo des Computerspiel Halflife, welches von einigen Teammitgliedern gespielt wird

6 Anforderungen

Die folgenden Werte sind aktuelle Werte und keine finalen. Zusätzlich sind aktuell nicht alle Werte festgelegt, bzw. berechnet. Dies wird jedoch noch folgen.

Anforderung	**Messwert**
Masse des CanSat	230 g
Höhe des CanSat	115 mm
Durchmesser des CanSat	67 mm
Länge des Bergungssystems (vgl. Pkt 2 Anhang 1)	xy mm
Planmäßige Flugzeit	xy Sekunden
Berechnete Sinkgeschwindigkeit	15 m/S
Genutzte Funkfrequenz	434 mHz
Energieverbrauch	2833.4 mW
Gesamtkosten	145,75 €

Tabelle 6: Anforderungen an den CanSat

7 Reflexion des Projektverlaufes

7.1 Reflexion der Hardwaregruppe

Als wir angefangen haben das gesamte Projekt zu planen haben wir uns als Ziel gesetzt Ende Mai fertig zu sein. Dieses Datum haben wir Aufgrund der Abgabe unseres P5 gewählt, für welches wir das CanSat Projekt ebenfalls einreichen wollen. Uns war bewusst, dass dies ein sehr hoch gestecktes Ziel ist. Im Nachhinein haben wir relativ schnell gemerkt, dass wir dieses Ziel nicht erreichen können. Diese Verzögerung wurde durch mehrere Faktoren hervorgerufen. Dazu zählt der enorm hohe Anspruch den wir uns selber gesetzt haben. Dieser hatte immer wieder zur Folge, dass viele Dinge mehrfach oder gründlicher gemacht werden mussten, als es zu Anfang geplant war. Zum anderen haben wir verhältnismäßig lange gebraucht um uns auf eine finale Idee festzulegen und diese zu präzisieren. Da wir uns jedoch kontinuierlich zum arbeiten getroffen haben konnten wir dennoch gute Fortschritte erzielen. Wir lagen zwar die meiste Zeit über hinter unserem Zeitplan, konnten jedoch die Reihenfolge der zu bearbeitenden Aufgabenpakete größtenteils einhalten.

7.2 Reflexion der Softwaregruppe

7.2.1 Desktop Applikation

In der Planung zur Implementation der Desktop Applikation setzten wir uns verschiedene Meilensteine. Der Erste Meilenstein beschreibt eine Basisversion, welche eine funktionsfähige Software mit den wichtigsten Grundfunktion bildet. Anschließend sollten weitere Versionen angefertigt werden, welche weitere Features enthält, bis hin zur finalen Version. Die Termine zu den Meilensteinen für diese Versionen konnten wir nicht einhalten. Denn die Funktion der Oberfläche konnten leider erst in der Finalen Version fertiggestellt werden, obwohl sie bereits in der Basisversion fertiggestellt sein sollte. Die Unterschätzung der Oberflächenkomponenten sorgte dafür das die Planung zwar für die logischen Softwarekomponenten zur Implementation gepasst hat, jedoch die graphischen Softwarekomponenten erst mit dem Finalen Meilenstein alle anderen Meilensteine abschloss. Im Endeffekt lagen wir stets gut in der Zeit, da wir die Software zum Finalen Abgabetermin weites gehend fertigstellten. Die Arbeit an der Implementation häufte sich gegen Ende des Projektes nicht, da wir kontinuierlich unsere Aufgaben abarbeiteten.

7.2.2 Android Applikation

7.3 Reflexion der Zusammenarbeit zwischen den Teams

Da der inhaltliche Schwerpunkt der beiden Teams relativ wenig miteinander zu tun hat sollte es theoretisch relativ wenige Berührungspunkte geben. Dies war bei unserer Projektarbeit jedoch nicht so. Da die Arbeit der beiden Halbgruppen zur gleichen Zeit in der gleichen Räumlichkeit stattfand war es oft so, dass teamübergreifend diskutiert wurde. Dies hat den Vorteil, dass beide Teams nochmal einen anderen Blick auf eventuelle Problemstellungen bekommen und so einfache oder bessere Lösungen für Probleme finden können. Zusätzlich lief die Absprache über den Datenaustausch zwischen Bodenstation und CanSat sehr gut. Die beiden Teams haben also hervorragend kooperiert und gemeinsam versucht ein bestmögliches Gesamtprodukt zu erschaffen.

8 Anhang

8.1 Einleitung

8.1.1 Blockdiagramm

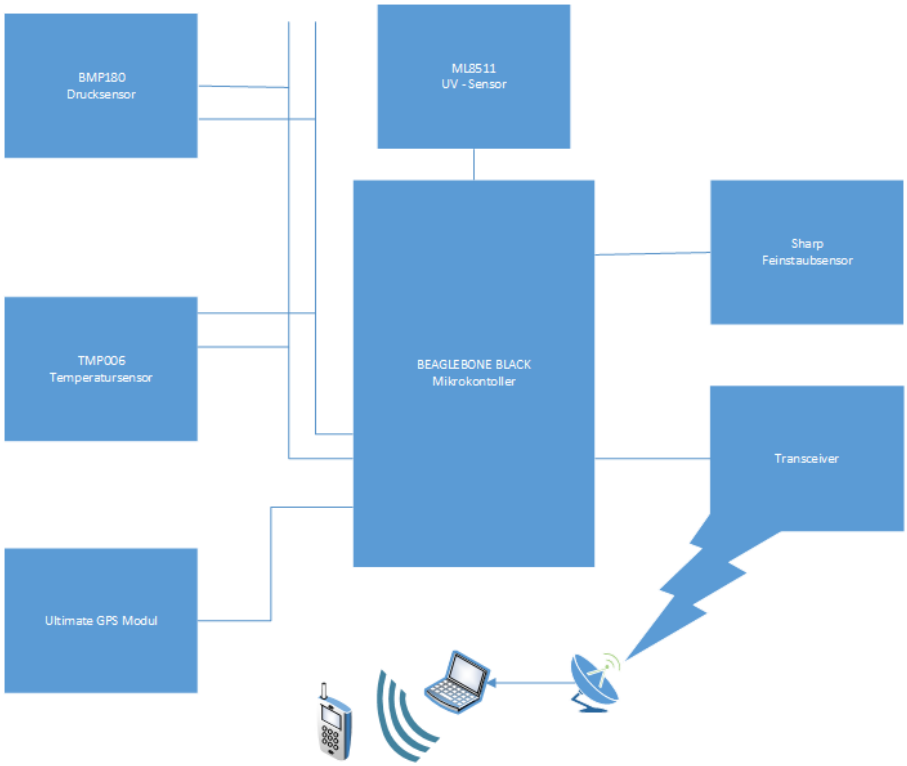

Abbildung 5: Das GANTT-Diagramm der Hardware Gruppe

8.2 GANTT-Diagramme

8.2.1 Hardware-GANTT

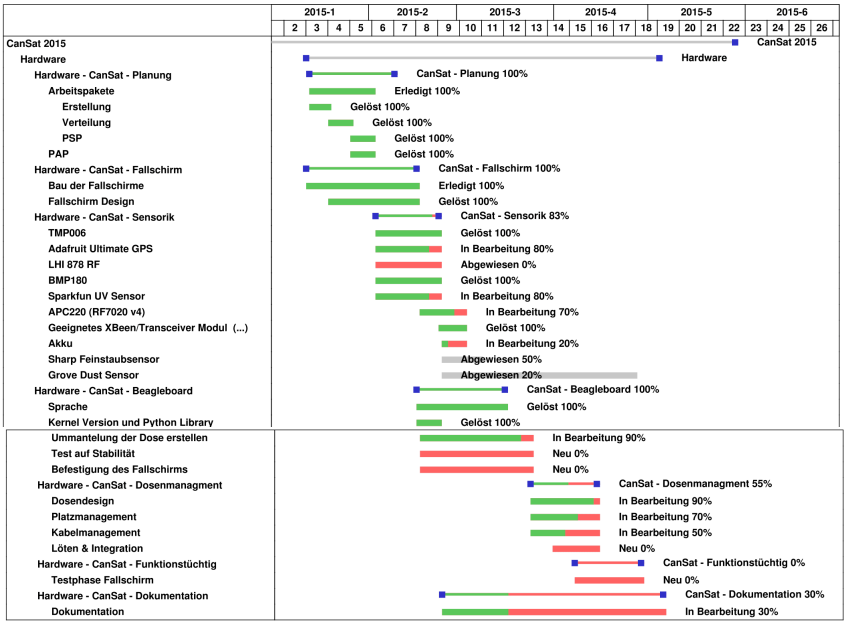

Abbildung 6: Das GANTT-Diagramm der Hardware Gruppe

8.2.2 Bodenstation-GANTT

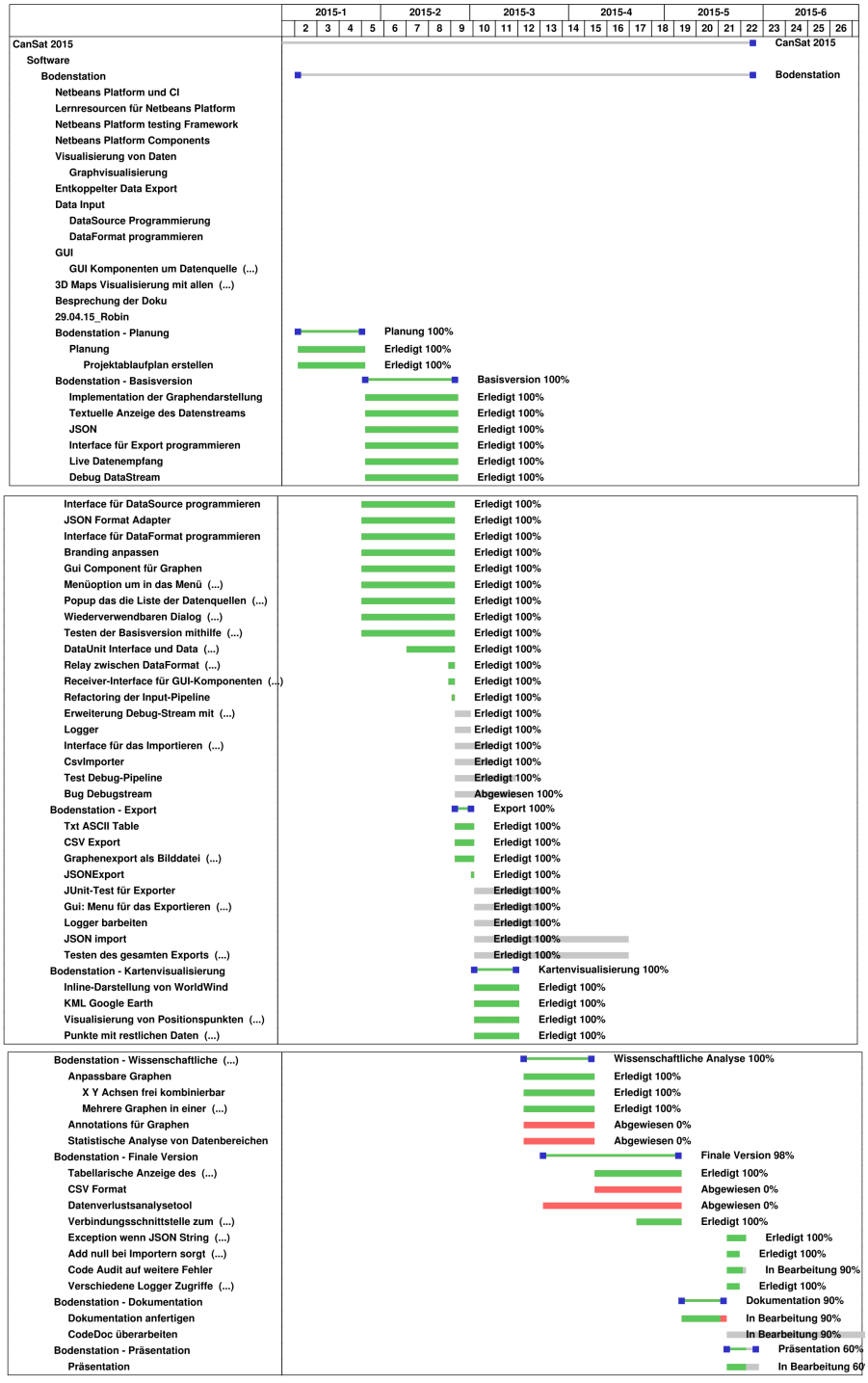

Abbildung 7: Das GANTT-Diagramm der Bodenstation

8.3 Der CanSat

Abbildung 8: Die Hülle und ein Dosendeckel

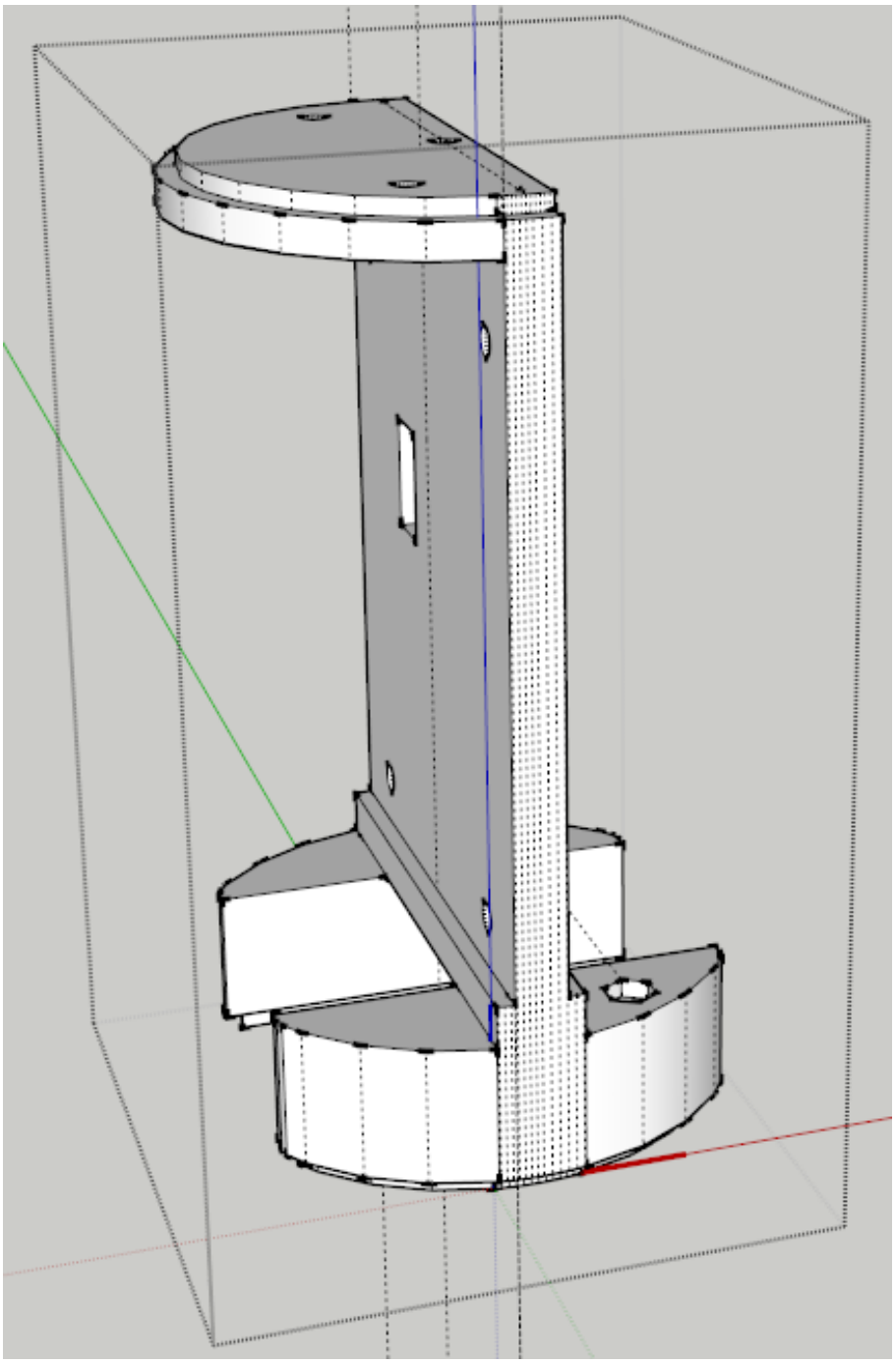

Abbildung 9: Screenshot der Zwischenwand aus Sketchup

Abbildung 10: Der Satellit (Diese Zeichnung ist möglicherweiße nicht sichtbar, da es eine 3D Zeichnung ist. Bitte verwenden sie den Adobe Acrobat Reader)

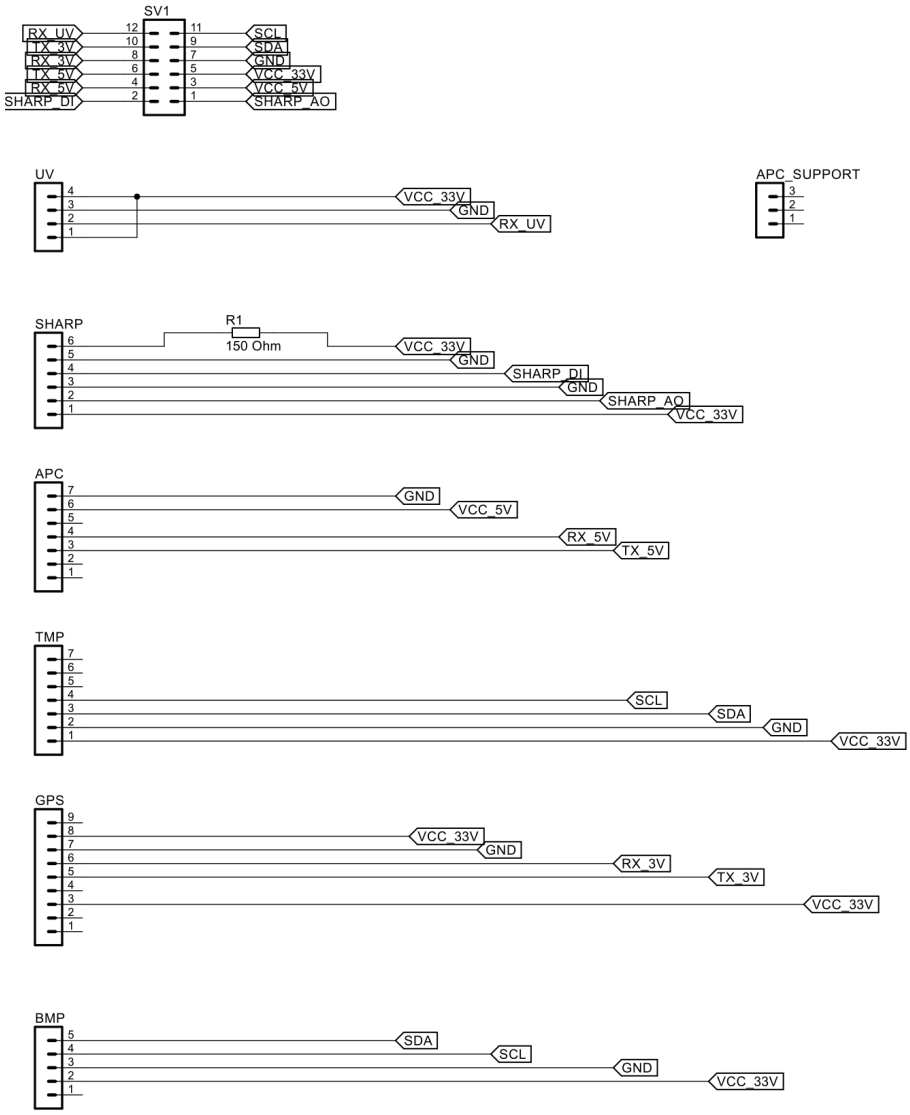

Abbildung 11: Der Schaltplan der Sensorik Platine

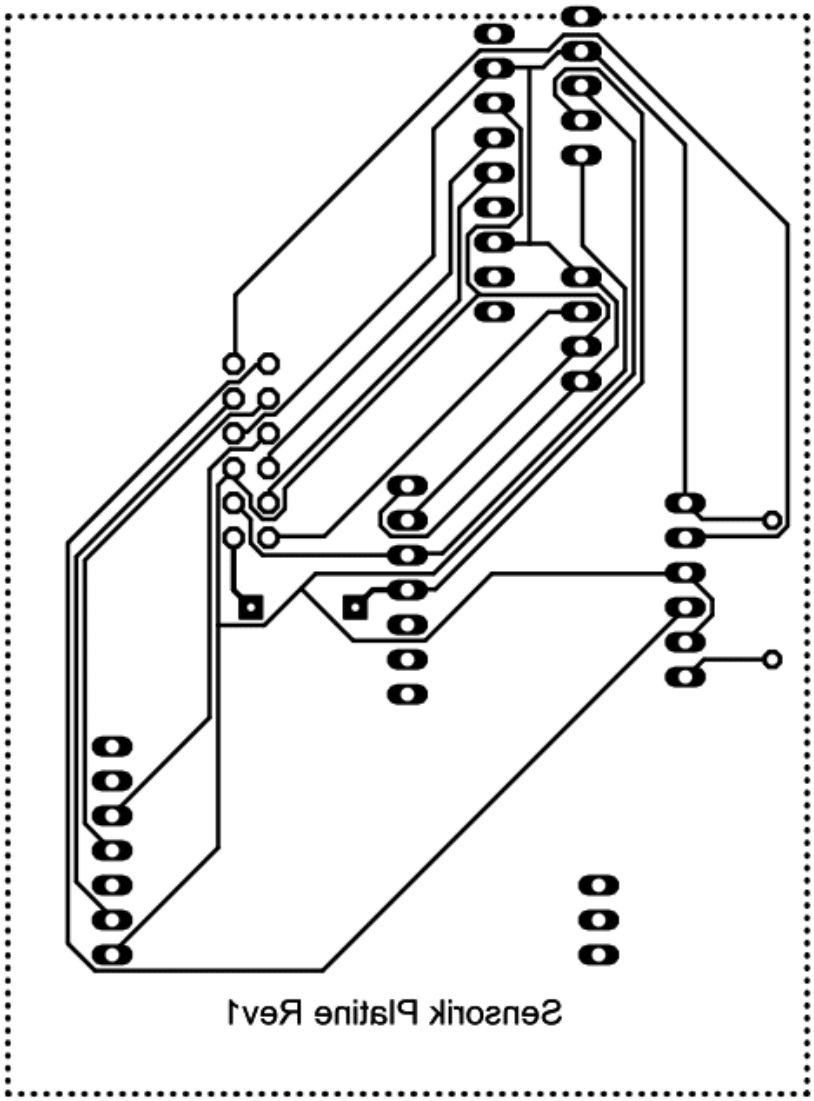

Abbildung 12: Das Layout der Sensorik Platine

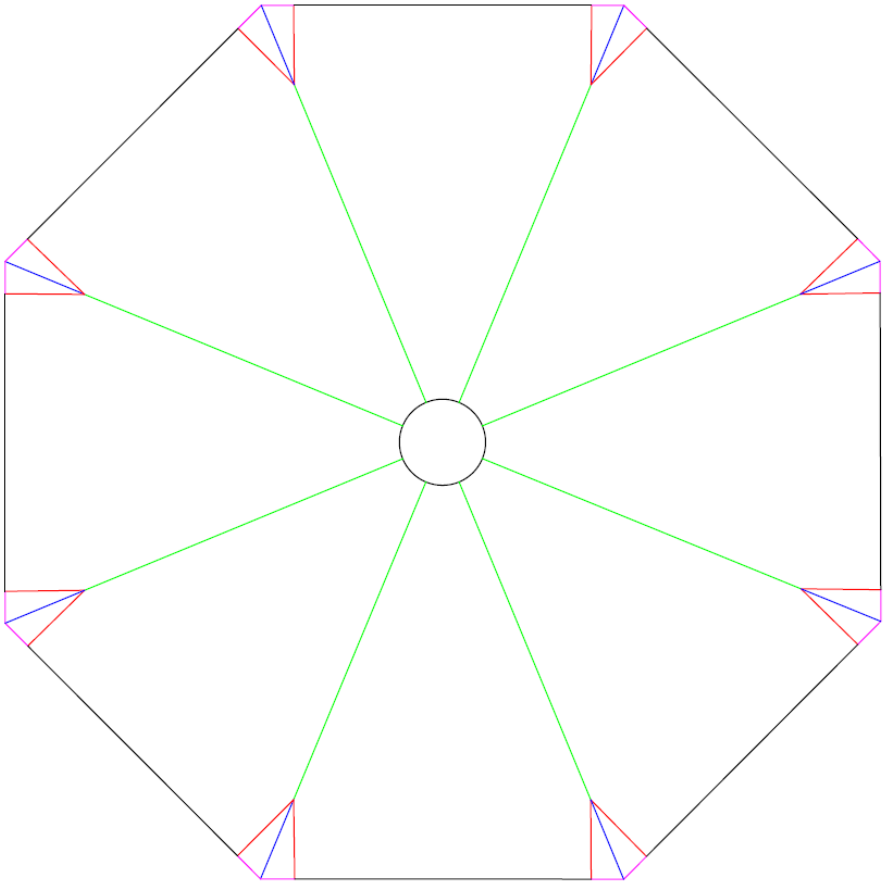

Abbildung 13: Skizze des Fallschirms

8.4 Bodenstationsarchitektur

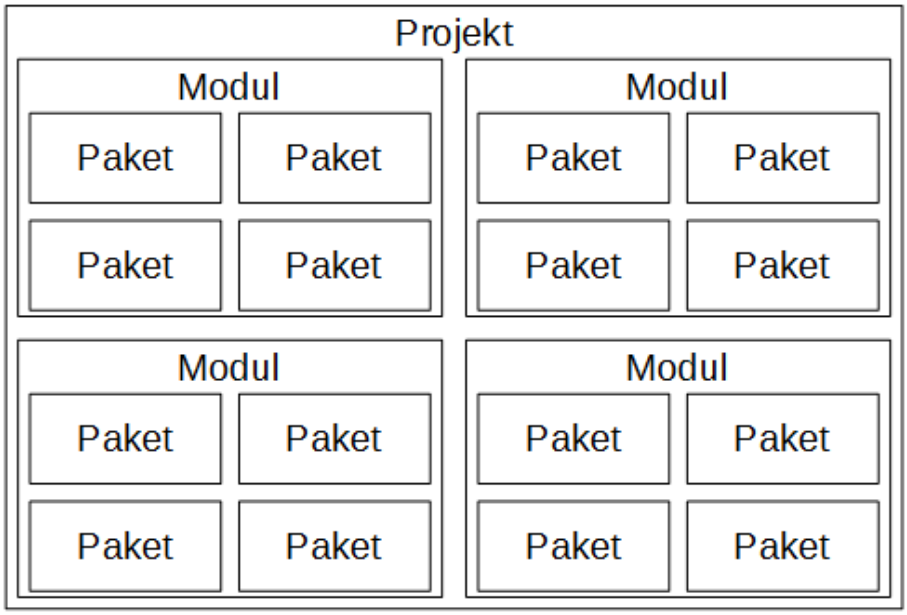

Abbildung 14: Modularchitektur von Netbeans Platform

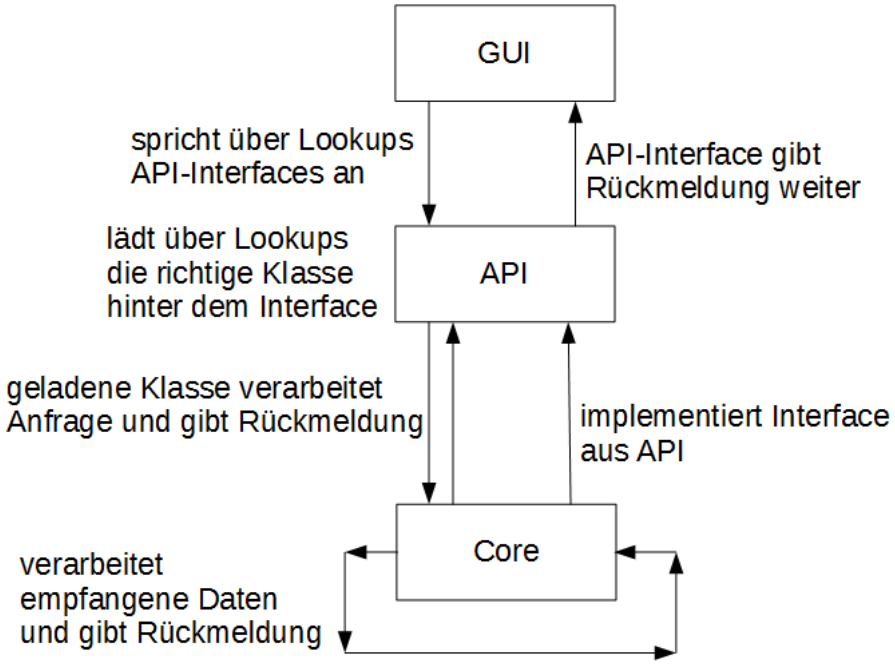

Abbildung 15: Modularchitektur der Bodenstation

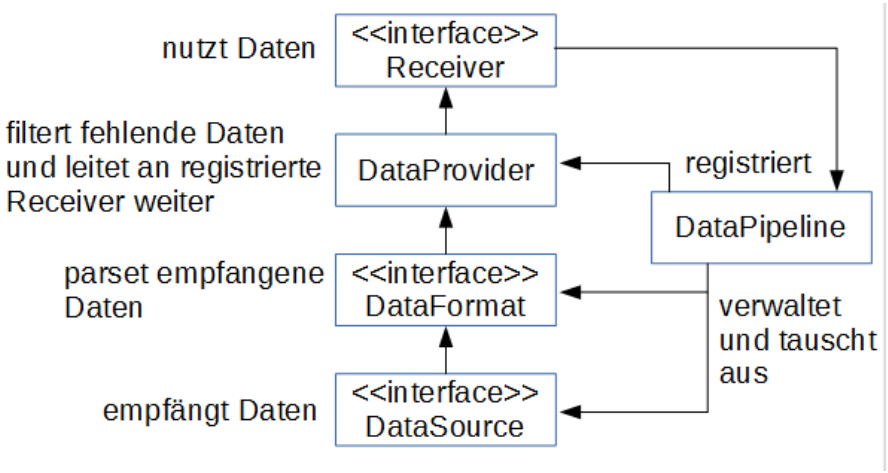

Abbildung 16: Architektur der Input-Pipeline