

# **Entwurf und Implementation einer Daten-Schnittstelle zum Betrieb des Laserscanners VLP-16 an einem Raspberry Pi**

**Bachelorthesis**

vorgelegt von:  
Florian Timm

Mittwoch, den 13. Dezember 2017

**Verfasser**

Florian Timm

Matrikelnummer: 6028121

Gaiserstraße 2, 21073 Hamburg

E-Mail: florian.timm@hcu-hamburg.de

**Erstprüfer**

Prof. Dr. rer. nat. Thomas Schramm

HafenCity Universität Hamburg

Überseeallee 16, 20457 Hamburg

E-Mail: thomas.schramm@hcu-hamburg.de

**Zweitprüfer**

Dipl.-Ing. Carlos Acevedo Pardo

HafenCity Universität Hamburg

Überseeallee 16, 20457 Hamburg

E-Mail: carlos.acevedo@hcu-hamburg.de

## Kurzzusammenfassung

Die vorliegende Arbeit ist Teil eines Projektes, dass die Entwicklung eines Systems zum Ziel hat, welches den modular austauschbaren Betrieb verschiedenster Sensorsysteme an einem Multikopter erlauben soll. Im Speziellen soll hier die Datenschnittstelle von einem Kompakt-Laserscanner Velodyne Lidar Puck VLP-16 zu einem Einplatinencomputer Raspberry Pi entwickelt und implementiert werden. Der Scanner selbst liefert hierbei die Daten in einem proprietären, binären Format, welche in ein einfache lesbares Format, hier eine ASCII-Datei, umgewandelt und gespeichert werden sollen. Außerdem sollen die Daten mit einem eindeutigen Zeitstempel versehen werden, um diese später mit anderen Sensorsystemen verknüpfen zu können. Diese Datentransformation sollte möglichst simultan zur Aufnahme erfolgen.

Auch Teil der Arbeit ist die Schaffung einer Steuerung der Aufnahme des Laser-scanners. Hierfür wurde ein Bedienmodul entwickelt, welches am Raspberry Pi direkt angeschlossen werden kann, sowie eine Steuerungsweboberfläche eingebunden, die die Steuerung während des Fluges ermöglichen soll.

## Abstract

The present work is part of a project aimed the development of a system that allows the modular interchangeable operation of various sensor systems on a multicopter. In particular, the data interface for compact laser scanner Velodyne Lidar Puck VLP-16 to a single-board computer Raspberry Pi will be developed and implemented. The scanner itself provides the data in a proprietary, binary format, which should be converted and stored into an easy-to-read ASCII file. In addition, the data should be provided with a unique timestamp in order to be able to link it later with other sensor systems. This data transformation should be carried out as simultaneously as possible while recording.

Also part of the work is the creation of a control of the recording of the laser scanner. For this purpose, an operating module was developed, which can be connected directly to the Raspberry Pi, as well as a web control surface integrated in the software, which should enable the control during the flight.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Problemstellung . . . . .	1
1.2 Zielsetzung . . . . .	1
1.3 Struktur . . . . .	2
<b>2 Grundlagen des Airborne Laserscanings</b>	<b>3</b>
2.1 Laserscanner . . . . .	3
2.1.1 Entfernungsmessung . . . . .	3
2.1.2 Ablenkeinheit . . . . .	5
2.1.3 Oberflächeneffekte . . . . .	6
2.2 Positionsbestimmung mittels globalen Navigationssatellitensystemen . .	7
2.3 Inertiale Messeinheit . . . . .	7
2.4 Kombination des Messsystems . . . . .	9
2.5 Bisherige Systeme zur dreidimensionalen Erfassung mittels Multikoptern	9
<b>3 Technische Realisierung</b>	<b>10</b>
3.1 Verwendete Gerätschaften . . . . .	10
3.1.1 Velodyne VLP-16 . . . . .	10
3.1.2 Inertiale Messeinheit und GNSS-Empfänger iMAR iNAT-M200-FLAT . . . . .	11
3.1.3 Raspberry Pi 3 Typ B . . . . .	12
3.1.4 Multikopter Copterproject CineStar 6HL . . . . .	13
3.1.5 Gimbal Freefly MöVI M5 . . . . .	15
3.2 Auswahl des Datenverarbeitungssystems . . . . .	15
3.3 Stromversorgung . . . . .	16
3.4 Anbindung des Raspberry Pi an den Laserscanner . . . . .	16
3.5 Verbindung des GNSS-Moduls zum Laserscanner . . . . .	17
3.6 Steuerung im Betrieb . . . . .	18
3.7 Platinenentwurf und -realisierung . . . . .	20
<b>4 Theoretische Datenverarbeitung</b>	<b>23</b>
4.1 Verwendung von Python . . . . .	23

4.2	Datenlieferung vom Laserscanner . . . . .	23
4.3	Geplantes Datenmodell . . . . .	24
4.4	Weiterverarbeitung der Daten zu Koordinaten . . . . .	25
4.5	Anforderungen an das Skript . . . . .	26
<b>5</b>	<b>Entwicklung des Skriptes</b>	<b>29</b>
5.1	Klassenentwurf . . . . .	29
5.2	Evaluation einzelner Methoden . . . . .	29
5.3	Multikern-Verarbeitung der Daten . . . . .	31
5.4	Klassen . . . . .	32
5.4.1	VdAutoStart . . . . .	32
5.4.2	VdInterface . . . . .	34
5.4.3	VdHardware . . . . .	34
5.4.4	VdPoint . . . . .	35
5.4.5	VdDataset . . . . .	35
5.4.6	VdFile . . . . .	35
5.4.7	VdBuffer . . . . .	35
5.4.8	VdTransformer . . . . .	35
5.4.9	VdConfig . . . . .	35
5.5	Beispiel-Quelltext-Zitat . . . . .	35
<b>6</b>	<b>Konfiguration des Raspberry Pi</b>	<b>36</b>
6.1	Installation von Raspbian . . . . .	36
6.2	Befehle mit Root-Rechten . . . . .	37
6.3	IP-Adressen-Konfiguration . . . . .	37
6.4	Konfiguration als WLAN-Access-Point . . . . .	38
6.5	Autostart des Skriptes . . . . .	39
<b>7</b>	<b>Systemüberprüfungen</b>	<b>40</b>
7.1	Untersuchung der Gleichzeitigkeit von PPS-Signalen von verschiedenen GNSS-Empfängern . . . . .	40
7.2	Messgenauigkeit des Laserscanners im Vergleich . . . . .	40
<b>8</b>	<b>Ausblick</b>	<b>41</b>
<b>Literaturverzeichnis</b>		<b>42</b>
<b>Abbildungsverzeichnis</b>		<b>44</b>
<b>Tabellenverzeichnis</b>		<b>45</b>
<b>Anhang</b>		<b>46</b>

<b>A Python-Skripte</b>	<b>47</b>
A.1 vdAutoStart.py . . . . .	47
A.2 vdBuffer.py . . . . .	55
A.3 vdTransformer.py . . . . .	57
A.4 vdInterface.py . . . . .	59
A.5 vdGNSSTime.py . . . . .	61
A.6 vdHardware.py . . . . .	63
A.7 vdFile.py . . . . .	66
A.8 vdDataset.py . . . . .	68
A.9 vdPoint.py . . . . .	72
A.10 config.ini . . . . .	74
A.11 convTxt2Obj.py . . . . .	75
<b>B Beispieldateien</b>	<b>77</b>
B.1 Rohdaten vom Scanner . . . . .	77
B.2 Dateiformat für Datenspeicherung als Text . . . . .	77
B.3 Dateiformat für Datenspeicherung als OBJ . . . . .	78

# 1 Einleitung

## 1.1 Problemstellung

Daten aus Airborne Laserscanning, dem Abtasten von Oberflächen mit einem Laser-scanner aus der Luft, lassen sich für viele verschiedene Zwecke benutzen. Oft werden sie zur Erfassung von digitalen Geländemodellen verwendet, aber auch für die Erstellung von Stadtmodellen oder Vegetationsanalysen sind die Daten nutzbar. Aktuell werden als Trägersysteme des Laserscanners Helikopter oder Flugzeuge verwendet, die mit entsprechender Sensorik ausgerüstet sind. Diese Messmethode lohnt sich allerdings nicht für die Vermessung kleinerer Gebiete und ist auch aufgrund der Größe und die Gefahren des Fluggerätes nicht für die Aufnahme feiner Strukturen wie Fassaden geeignet, bei denen zwischen Häuserschluchten geflogen werden müsste. Außerdem sind die Betriebs- und Anschaffungskosten sehr hoch, so dass sich eine solche Messung oft nur für sehr große Gebiete lohnt. Alternativ bietet sich die terrestrische Messung mittels Tachymeter oder auch per Laserscanner um kleinere Gebiete abzubilden an – hier benötigt die Aufnahme jedoch viel Zeit und Personal. Hinzukommt, dass die Genauigkeit für viele Anwendungsfälle der 3D-Modelle zu hoch ist. Beide Möglichkeiten, die Messung aus der Luft oder vom Boden, sind sehr kostenintensiv. Ein Lösungsansatz hierfür wäre es, anstatt eines Helikopters als Trägersystem, einen Multikopter zu nutzen. Jedoch ist die Tragfähigkeit für die meisten Laserscanning-Systeme nicht ausreichend. Daher basieren 3D-Erfassungssysteme, die Multikopter nutzen, heutzutage meist auf photogrammetrischen Prinzipien, welche Luftbilder zur Erfassung nutzen. Hierzu muss jedoch ausreichend Beleuchtung vorhanden sein, welches wiederum die Einsetzbarkeit des Systems in Städten beschränkt, in denen nur nachts für ausreichende Sicherheitszonen zum Betrieb von Multikoptern gesorgt werden kann.

## 1.2 Zielsetzung

Gesamtziel ist es, ein Laserscanning-System zu entwickeln, dass von einem Multikopter getragen werden kann. Hierbei soll vor allem auf ein geringes Gewicht geachtet, aber auch die Kosten niedrig gehalten werden. Im Speziellen soll hier als erster Schritt die Datenverarbeitung des Laserscanners in einem solchen System realisiert werden. Hierfür soll ein Ein-Platinen-Computer Typ Raspberry Pi 3 die Speicherung und Aufbereitung

der von einem Laserscanner Velodyne Puck VLP-16 aufgezeichneten Laserpunktdata übernehmen. Hierfür müssen entsprechende Schnittstellen zum Verbinden der Geräte in Hard- und Software entwickelt werden.

## 1.3 Struktur

Im Kapitel 2 sollen die Grundlagen des luftgestützten Laserscannings erläutert werden. Außerdem wird die benötigte Hardware zur Durchführung eines solchen Laserscannings besprochen. Im Folgenden wird näher auf die Realisierung des Projektes eingegangen: Welche Hardware wurde verwendet und wie wurde Sie angeschlossen (Kapitel 3), wie sollen die Daten verarbeitet werden (Kapitel 4) und wie wird die Verarbeitung schließlich durchgeführt (Kapitel 5) und das System konfiguriert (Kapitel 6). Einzelne Komponenten werden in Kapitel 7 auf ihre Genauigkeit und Zuverlässigkeit geprüft. Zum Abschluss soll in Kapitel 8 noch ein Einblick in die Zukunft des Systems geworfen werden.

# 2 Grundlagen des Airborne Laserscannings

Airborne Laserscanning bezeichnet das Verfahren, bei dem ein Laserscanner, welcher an einem Fluggerät befestigt ist, Oberflächen kontaktlos dreidimensional erfasst (Beraldin et al., 2010, S. 1). Der Laserscanner liefert hierbei Daten in Form der Abstrahlrichtung des Strahles und der Entfernung, relativ zu seiner eigenen Ausrichtung und Position. Um diese lokalen Daten in ein globales System zu überführen, werden zusätzlich die Ausrichtung und die Position des Laserscanners zum Zeitpunkt der Messung benötigt (Beraldin et al., 2010, S. 22f). Diese Daten liefern im Normalfall eine inertiale Messeinheit (siehe Abschnitt 2.3) und ein Navigationssatellitenempfänger (siehe Abschnitt 2.2). Auf diese Bestandteile wird im Folgenden eingegangen. Anschließend werden einige bisherige Lösungsansätze zur dreidimensionalen Erfassung auf Basis von Multikopterplattformen vorgestellt.

## 2.1 Laserscanner

Ein Laserscanner besteht grundlegend aus einer Laser-Entfernungsmeßeinheit und einer Ablenkeinheit. Für beide Teile gibt es verschiedenste Bauformen, auf die im Folgenden eingegangen wird.

### 2.1.1 Entfernungsmeßung

Für die Messung von Entfernungen mittels Laserscanners gibt es zwei meistgenutzte Verfahren:

**Impulsmessverfahren** Das bei Laserscannern am häufigsten eingesetzte Verfahren ist das Impulsmessverfahren, englisch time-of-flight genannt. Hierbei werden einzelne Laserimpulse ausgesandt. Mit dem Aussenden startet ein hochgenauer Timer seine Messung. Beim Eintreffen des an einer Oberfläche reflektierten Strahles beim Laserscanner wird der Timer gestoppt. Aus dieser gemessenen Laufzeit lässt sich die zurückgelegte Strecke des Lichtstrahles und somit die doppelte Entfernung zu der Oberfläche bestimmen. Hierzu wird der Brechungsindex  $n$  des vom Laser durchlaufenen Mediums

benötigt. Bei der Messung in der Luft lässt sich dieser aus den Daten von Temperatur-, Druck- und Luftfeuchtemessung ausreichend genau berechnen. Aus der bekannten Lichtgeschwindigkeit  $c_0$  und der benötigten Zeit  $t$  lässt sich dann die Entfernung  $s$  mit der Gleichung 2.1 berechnen.

$$s = \frac{c_0}{n} \cdot \frac{t}{2} \quad | \text{ Streckenberechnung} \quad (2.1)$$

**Phasenvergleichsverfahren** Eine andere, für Laserscanner selten verwendete Methode, ist das Phasenvergleichsverfahren. Hierbei wird nicht direkt die Zeit gemessen sondern die Phasenverschiebung eines kontinuierlichen Lichtstrahles, der mit einer Sinuswelle amplitudenmoduliert wurde (Intensitäts- bzw. Helligkeitsschwankungen). Hierdurch können weniger frequente Wellen (Modulationswelle) verwendet werden, wodurch sich bei guten Ausbreitungseigenschaften der hochfrequenten Trägerwellen die leichtere Verarbeitbarkeit von längeren Wellen ausnutzen lässt. Durch Messung des Phasenunterschiedes des Messstrahles, kann auf die Reststrecke der nicht-vollständigen Phasen des Messstrahles geschlossen werden. Als Trägerwelle wird normalerweise Infrarotlicht verwendet, da dieses gute Ausbreitungseigenschaften hat. Die hierfür benötigte Modulationswelle wird durch einen Quarzoszillator erzeugt. Ein hier verbautes Quarzplättchen wird durch Anlegen einer Spannung in eine Schwingung versetzt, die Schwingung verstärkt und an den Infrarot-Laser geleitet, so dass dieser das modulierte Infrarotlicht aussendet. Die maximal eindeutig messbare Entfernung ist direkt von der längsten verwendeten Wellenlänge, dem Grobmaßstab, abhängig: Da nur die Phasenunterschiede und nicht die Anzahl der Schwingungen gemessen werden können, ist die maximale eindeutige Streckenmessung genau halb so groß wie die maximale Wellenlänge (Messung von Hin- und Rückweg). Wenn längere Strecken als die halbe Wellenlänge gemessen werden, ist nicht bekannt, wie viele ganze Wellen das Licht schon zurückgelegt hat. Die Messung wäre mehrdeutig. Zur Messung der Phase werden die ausgesendete und die eingehende Messwelle mit einer Überlagerungsfrequenz vermischt, die aus diesen beiden hochfrequenten Wellen eine niederfrequente Welle erzeugt. Da die Genauigkeit der Phasenverschiebungsmessung begrenzt ist, wird durch Nutzung verschiedener Wellenlängen eine Genauigkeitssteigerung durchgeführt. Nach der groben Messung mit einer langen Wellenlänge, wird die Genauigkeit durch die Verwendung immer kürzerer Modulationswellen gesteigert. Eine grobe Messung ist jedoch vorher notwendig, da ansonsten die Anzahl der ganzen Schwingungen des Messstrahles unbekannt ist. (Witte & Schmidt, 2006, S. 311ff)

**Zeitmessung** Bei beiden Verfahren ist die genaue Zeitmessung ein Problem. Eine Möglichkeit dieser Messung ist die Nutzung eines Frequenzgenerators, welcher Zählimpulse erzeugt. Diese werden dann zwischen zwei Flanken der zu messenden Ausgangs- und

Eingangswellen mehrfach gezählt und gemittelt und ergeben so zum Beispiel die Phasenverschiebung. Dieses Verfahren wird als digitale Messung bezeichnet. Eine andere Methode ist die analoge Messung. Hierbei öffnet die eine Flanke den Stromfluss zu einem Kondensator, die Flanke der anderen Welle schließt sie wieder. Aus der Ladung des Kondensators kann dann auf den Phasenwinkel und die Phasenverschiebung geschlossen werden. (Witte & Schmidt, 2006, S. 314f)

## 2.1.2 Ablenkeinheit

Bei den meisten Laserscannern ist nur eine Laserentfernungsmesseinheit verbaut. Um hiermit verschiedene Punkte messen zu können, muss der Laserstrahl durch geeignete Verfahren abgelenkt werden. Auch hierfür gibt es im Airborne Laserscanning verschiedene Ansätze: (Pack et al., 2012, S. 23ff; Beraldin et al., 2010, S. 16ff)

**Schwenkspiegel** Der Laserstrahl wird auf einen schwingenden, flachen Spiegel gerichtet. Durch die Schwingung wird der Laserstrahl in einer Ebene nach links und rechts abgelenkt. Durch die Bewegung des Fluggerätes wird der Laser in Richtung der Schwingachse bewegt. Es entsteht eine Zick-Zack-Linie auf der Oberfläche als Messmuster.

**Rotierender Polygon-Spiegel** Beim drehenden Polygon-Spiegel dreht sich ein Prisma mit einem gleichseitigen Polygon in der Achse der Flugbewegung. Seine rechteckigen Seiten sind verspiegelt und der Laser auf diese gerichtet. Von oben gesehen wird der Strahl somit immer nur in eine Richtung abgelenkt und springt dann wieder zurück auf die andere Seite. Es entsteht ein Streifenmuster.

**Palmer Scanner** Beim Palmerscanner rotiert ein Flachspiegel um eine Achse, die fast senkrecht zur Spiegeloberfläche steht. Da der Spiegel nicht genau senkrecht auf dieser Achse montiert ist, beschreibt der auf den Spiegel gerichtete Laser einen Kreis. Durch einen im 45 Grad Winkel zur Achse stehenden Spiegel und einem sich in der Drehachse befindenden Scanner können die Strahlen auch rechtwinklig abgelenkt werden und somit eine Ebene scannen. Dies wird häufig bei terrestrischen Laserscannern im Zusammenhang mit einer zweiten Drehachse angewandt.

**Glasfaser-Scanner** Der Glasfaserscanner nutzt zur Ablenkung zusätzlich Glasfasern, welche fest verklebt sind. Hierdurch sind die Winkel zur Seite festgegeben. Zum Beispiel ein Polygonspiegel wie er zuvor beschrieben wurde, reflektiert den Messstrahl in die jeweiligen Faserbündel. Die Ablenkungswinkel sind fest vom Hersteller vorgeben.

**Zusätzliche Achsen** Zusätzlich zu den Spiegelmechanismen verfügen die terrestrischen Laserscanner über eine weitere Achse. Während beim Airborne Laserscanning die weitere Bewegung des Lasers durch die Fortbewegung des Fluggerätes durchgeführt wird, muss dies bei der terrestrischen Messung ein Motor übernehmen. Panorama-Laserscanner haben hierfür einen Drehmechanismus um ihre Hochachse. Bei Kamerascannern, sie messen nur eine quadratische Fläche wie eine Kamera, kann die zusätzliche Bewegung auch einfach durch einen zweiten Ablenkungsspiegel erfolgen. (Beraldin et al., 2010, S. 37)

Zusätzlich haben natürlich alle Ablenk- und Drehsysteme eine Messeinheit, die den Stand des Spiegels misst. Hierdurch lässt sich dann die Abstrahlrichtung des Lasers berechnen, beziehungsweise beim Faserlaser bestimmen, welches Faserbündel genutzt wurde. Die Richtung wird dann wiederum zur Berechnung von Koordinaten benötigt.

Bild  
malen

### 2.1.3 Oberflächeneffekte

Der vom Laser ausgesendete Impuls wird nur bei rechtwinklig zur Strahlenachse verlaufenden, ebenen Oberflächen als identischer, abgeschwächter Impuls zurückgestrahlt. Der Laser trifft bei der Messung nicht, wie idealisiert angenommenen, punktförmig auf die Oberfläche, sondern stellt einen Kreis beziehungsweise bei schrägem Auftreffen eine Ellipse mit einer bestimmten Größe, dem sogenannten Footprint dar. Hierdurch ergeben sich je nach Oberfläche verschiedene Reflexionen: Bei zum Laserstrahl schrägen Oberflächen wie einem Dach wird das Signal geweitet, die Impulsdauer des reflektierten Strahles (Echo) wird verlängert, da er auf der Oberfläche zeitversetzt auftrifft. Ein anderes Phänomen sind mehrfache Echos. Dies tritt auf, wenn zwei unterschiedlich weite entfernte Oberflächen von einem Strahl getroffen werden – zum Beispiel bei der Messung von Gebäudekanten oder Bäumen. zeigt die Echos in grafischer Form. (Beraldin et al., 2010, S. 28)

Abbildung

Laserscannern mit Impulsmessverfahren ermöglichen typischerweise bis zu vier einzelne Echos aufzuzeichnen. Alternativ gibt es Scanner, die das komplette Signal mit einer Abtastrate von bis zu 0,5 Nanosekunden digitalisieren. Hier ist es dann möglich, spezielle Auswertung aufgrund der Wellenform im Postprocessing durchzuführen. (Beraldin et al., 2010, S. 29)

Abbildung

## **2.2 Positionsbestimmung mittels globalen Navigationssatellitensystemen**

Zur Bestimmung der Position des Fluggerätes wird ein Empfänger für globale Navigationssatellitensysteme (global navigation satellite system, GNSS) verwendet. Ein solcher Empfänger kann durch die Laufzeitbestimmung des Signales von verschiedenen Satelliten zum Beispiel des US-amerikanischen Navstar GPS seine aktuelle Position bestimmen. Hierzu ist eine freie Sicht zum Himmel notwendig. Je nach Auswertung und Weiterverarbeitung des Signales sind Genauigkeiten zwischen 10 Metern ohne zusätzliche Daten und wenigen Millimetern bei statischen Dauermessungen und dem Einsatz von Daten von Referenzstationen im Postprocessing möglich. Es befinden sich pro System etwa 30 Satelliten in einer bekannten Umlaufbahn. Durch an Bord befindliche Atomuhren können die Satelliten hochgenaue Zeitstempel und sich wiederholende Codemuster aussenden. Im Fall von Navstar GPS erfolgt die Aussendung aktuell auf drei verschiedenen Frequenzen L1, L2 und L5. Für die öffentliche Nutzung ist nur L1 freigegeben. L2 und L5 sind der militärischen Nutzung vorbehalten. Durch reine Auswertung des ausgesendeten L1-Codes können Genauigkeiten bis 5 Meter erreicht werden. Für geodätische Anwendungsfälle wird zusätzlich die Phasenmessung benutzt. Hierbei wird nicht nur das dem Funksignal aufmodellierte Codemuster ausgewertet, sondern auch die Phase des Signals. Hierdurch ist es auch möglich, dass verschlüsselte L2-Signal mitzunutzen. Durch die Nutzung von Referenzstationsnetzen wie SPOS können Genauigkeiten von 1-2cm in Echtzeit und von unter Zentimetergenauigkeit im Postprocessing erreicht werden (Witte & Schmidt, 2006, S. 375).

## **2.3 Inertiale Messeinheit**

Bei der inertialen Messeinheit (inertial measurement unit, IMU) handelt es sich um einen Sensor, der die Neigung sowie Drehbewegungen der Sensoreinheit misst. Sie wird benötigt, um beim Airborne Laserscanning die genaue Ausrichtung des Laserscanners zu bestimmen. Daher muss diese auch verwindungssteif mit dem Laserscanner verbunden sein. In Kombination mit den Positionsdaten des GNSS-Modules ermöglicht sie die Rekonstruktion der Flugbewegungen (Trajektorie). Ein weiterer Vorteil der inertialen Messeinheit ist ihre Messfrequenz: Im Gegensatz zum GNSS, dass im Normalfall nur eine Messung pro Sekunde durchgeführt, kann die IMU bis zu 500 Messungen pro Sekunde ausführen. Sie stützt daher nicht nur die GNSS-Messung sondern hilft auch, die Trajektorie zu interpolieren und somit auch für den Bereich zwischen den GNSS-Messungen genaue Positionen zu bestimmen. (Beraldin et al., 2010, S. 23ff)

Die Messung erfolgt mit mehreren Einzelsensoren: Für die Messung der Beschleunigung in drei Dimensionen sind drei jeweils rechtwinklig zueinander stehende Beschleu-

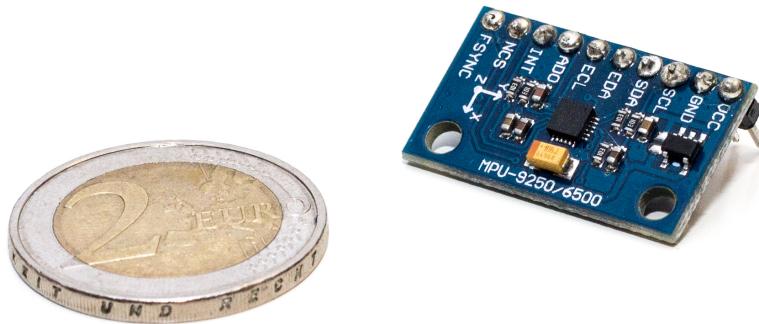


Abbildung 2.1: MPU-9250 - Low-Cost-MEMS-IMU-Modul wie es in vielen Consumer-Geräten und Multikoptern verwendet wird (schwarzes Bauteil mittig auf der Platine, eigene Aufnahme)

nigungsmesser verbaut. In der klassischen Bauform ist hierfür jeweils eine Probemasse zwischen zwei Federn gelagert. Durch eine auf die Probemasse wirkende Beschleunigung wird diese zwischen den Federn ausgelenkt. Die Messung der Drehrate erfolgt mittels drei einzelnen Kreiselinstrumenten (Gyroskop) in drei Achsen. Sie basieren auf Kreiseln, welche drehbar gelagert sind. Sie streben dazu, die Ausrichtung ihrer Drehachsen im Raum beizubehalten. Durch Messung der Kräfte kann die Drehrate berechnet werden. Einige inertiale Messeinheiten enthalten auch ein dreidimensionales Magnetometer, mit dem sich die magnetische Nordrichtung dreidimensional feststellen lässt.

Die Genauigkeit von inertialen Messeinheiten wird in ihre Messgenauigkeit und in ihre zeitliche Abweichung unterteilt. Die zeitliche Stabilität ist vor allem bei der Inertialnavigation, die ausschließlich auf deren Messungen basiert, wichtig.

Hauptsächlich unterschieden werden die inertialen Messeinheiten in klassische, mechanische Systeme, wie sie bereits hier beschrieben wurden, und mikroelektromechanische Systeme, sogenannte MEMS (microelectromechanical systems). Bei den zweitgenannten handelt es sich um stark miniaturisierte Bauteile (beispielsweise Abbildung 2.1), welche zum Beispiel in aktuellen Smartphones eingesetzt werden. Sie werden für Zwecke eingesetzt, in denen keine hohen Genauigkeitsanforderungen gestellt werden und der Preis gering gehalten werden soll. Auch zur Stabilisierung der Fluglage von Multikoptern oder Gimbalen werden diese Sensoren eingesetzt. Für geodätische Anwendungsfälle werden jedoch meist noch mechanische Systeme genutzt, da deren Genauigkeit und ihre zeitliche Abweichung geringer sind.

Quelle

## 2.4 Kombination des Messsystems

Um die drei eigenständigen Messsysteme kombiniert nutzen zu können, muss die relative Position der Systeme genau bekannt sein und darf sich während des Fluges nicht verändern. Beim klassischen Airborne Laserscanning vom Helikopter werden hierfür zum Beispiel eigenständige Module entwickelt, die alle benötigten Systeme verdrehsicher enthalten und an den Kufen des Helikopters montiert werden können (Beraldin et al., 2010, S. 23f)

Außerdem müssen alle Systeme synchronisiert werden, damit die Daten später miteinander verarbeitet werden können. Hierfür wird üblicherweise das Sekundensignal des Navigationssatellitensystems (pulse per second, PPS) genutzt. Das GNSS-System sendet dazu zu jeder vollen Sekunde der GNSS-Zeit ein Impuls aus, mit welchen sich die anderen Sensorsysteme synchronisieren können.

## 2.5 Bisherige Systeme zur dreidimensionalen Erfassung mittels Multikoptern

Die meisten aktuellen Verfahren zur Erzeugung von 3D-Modellen unter Nutzung von kompakten Multikoptern mit einer Tragkraft von bis zu 5kg, basieren auf photogrammetrischen Verfahren. Sie erzeugen Bilder, meist unter direkter Georeferenzierung, welche im Postprocessing zu Bildverbänden verknüpft werden. Mittels Bilderkennung werden hieraus 3D-Punktwolken berechnet. Nachteil dieses Verfahrens ist es, dass ausreichende Beleuchtung vorhanden sein muss. Es kann somit nur tagsüber geflogen werden, aber auch starke Schatten können das Ergebnis verschlechtern. Für die automatische Erstellung von Punktwolken muss außerdem das Gelände ausreichende Strukturen aufweisen, damit automatische Verknüpfungen der Pixel erfolgen können.

Laserscanning als aktiver Sensor hat hier den Vorteil, dass keine zusätzliche Beleuchtung benötigt wird – der Sensor bringt sein Licht selber mit. Problematisch ist hierbei jedoch die Größe der Systeme. Aus diesem Grund wurden bisher hauptsächlich Systeme mit großen UAVs erprobt und verwendet (Ehring et al., 2016, S. 19). Durch die immer weiter fortschreitende Miniaturisierung und die Weiterentwicklung von Laserscannern zum Beispiel für die Entwicklung von autonomen Fahrzeugen werden die Scanner auch inzwischen kleiner und leistungsfähiger.

Quelle,  
füllen

# 3 Technische Realisierung

Im Folgenden wird zunächst auf die verwendeten Geräte und ihre technischen Eigenarten eingegangen, bevor danach auf die technischen Verbindungen eingegangen wird.

## 3.1 Verwendete Gerätschaften

### 3.1.1 Velodyne VLP-16

Um Gewicht zu sparen, wird für die Messung ein miniaturisierter Laserscanner eingesetzt. Einer dieser Kompakt-Laserscanner ist der Velodyne Puck VLP-16 (siehe Abbildung 3.1). Er hat einen Durchmesser von etwa 10 cm und eine Höhe von 7 cm bei einem Gewicht von etwa 830 g ohne Kabel und Schnittstellenbox. Es handelt sich beim VLP-16 wahrscheinlich um einen Faserscanner (siehe Abschnitt 2.1.2) mit 16 Messstrahlen, der sich zusätzlich um seine Hochachse dreht. Genaue Angaben macht der Hersteller hierzu keine. Seine Messgenauigkeit beträgt laut Datenblatt 3 *cm*. Gemessen wird im Impulsmessverfahren (siehe Abschnitt 2.1.1) mit einem Infrarotlaser mit einer Wellenlänge von 903nm. (Velodyne Lidar, 2017b)

Der Scanner sendet die Messstrahlen mit einem Zeitabstand von  $2,3\mu s$  hintereinander aus, gefolgt von einer Nachladezeit von  $18,4\mu s$ , so dass jeder Messstrahl alle  $55,3 \mu s$  ausgesendet werden kann (Velodyne Lidar, 2016, S. 16). Es ergibt sich somit eine durchschnittliche Messfrequenz von  $289.357 \text{ Hz}$  (siehe Gleichung 3.1). Während der Messungen dreht sich der Laserscanner je nach Einstellung über das Webinterface des Scanners mit 5 bis 20 Umdrehungen pro Sekunde (Velodyne Lidar, 2017b). Pro ausgesendeten Strahl können jeweils die erste und die stärkste Reflexion zurück gegeben werden, so dass über eine halbe Million Punkte pro Sekunde gemessen werden können (siehe Gleichung 3.1). Die Daten werden anschließend über den Netzwerkanschluss gestreamt (siehe auch Abschnitt 4.2). Außerdem verfügt der Scanner über einen Anschluss für ein GNSS-Modul des Typs Garmin GPS 18x LVC. Auch andere GNSS-Module sind nutzbar, so dass im Weiteren der Versuch unternommen wurde, hier das GNSS-Modul der inertialen Messeinheit (siehe Unterabschnitt 3.1.2) oder eines uBlox-GNSS-Modules zu nutzen (siehe Abschnitt 3.5). Durch die Nutzung eines GNSS-Moduls am Scanner ist es möglich, die Daten mit einem hochgenauen Zeitstempel zu versehen.



Abbildung 3.1: Laserscanner Velodyne VLP-16 (eigene Aufnahme)

pel zu versehen und die Messungen des Scanners so in der Nachbearbeitung mit den Daten aus der inertialen Messeinheit zu verknüpfen.

$$f = \frac{1s}{55,295\mu s} \cdot 16 \frac{\text{Messstrahlen}}{\text{Messung}} = 289.357 \frac{\text{Messung}}{\text{Sekunde}}$$

$$n = 289.357 \frac{\text{Messung}}{\text{Sekunde}} \cdot 2 \frac{\text{Messwerte}}{\text{Messtrahl}} = 578.714 \frac{\text{Messwerte}}{\text{Sekunde}}$$
(3.1)

### 3.1.2 Inertiale Messeinheit und GNSS-Empfänger iMAR iNAT-M200-FLAT

Als inertiale Messeinheit wird das auf Abbildung 3.2 zu sehende Sensorsystem des Typs iMAR iNAT-M200-FLAT verwendet. Hierbei handelt es sich um ein hochgenaues MEMS-System. Durch die Verwendung von mikroelektromechanischen Bauteilen wiegt der Sensor inklusive Gehäuse nur 550 Gramm. Er kann bis zu 500 Messungen pro Sekunde durchführen. Die Abweichung der Richtungsmessungen pro Stunde liegt unter 0,5 Grad. (iMAR Navigation GmbH, 2015)

Außerdem verfügt die verwendete Einheit über zwei differentielle Satellitennavigationsempfänger (GNSS-Module, siehe Abbildung 3.3). Durch Nutzung einer zusätzlichen GNSS-Basisstation oder auch einem entsprechenden Korrekturdienst können diese eine Positionsgenauigkeit von etwa 2 Zentimeter in Echtzeit erreichen (iMAR Navigation GmbH, 2015). Durch Postprocessing lässt sich diese sogar noch steigern. Wilken (2017) Durch die Verwendung von zwei Empfängern, die an jeweils einem Ausleger befestigt sind (siehe Bild Abbildung 3.3), kann auch die Orientierung des Scanners bestimmt werden. Hierdurch wird die ungenaue Messung des magnetischen Nordpols

mehr  
Daten

alternativ:  
Matt-  
hias  
Wil-  
kens



Abbildung 3.2: iMAR iNAT-M200-Flat im Prototypen des modularen Gehäuses, Leitungen führen zu den GNSS-Antennen (eigene Aufnahme)

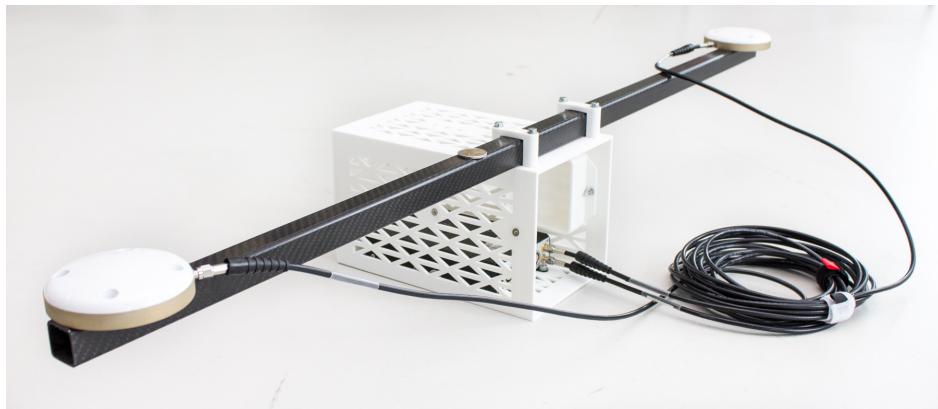


Abbildung 3.3: GNSS-Antennen des (links und rechts) iMAR iNAT-M200-Flat an Prototypen des modularen Gehäuses (eigene Aufnahme)

überflüssig. Außerdem kann die Positionssicherheit durch Mittlung der beiden Positionen erhöht werden.

Im Postprocessing kann aus den Daten der inertialen Messeinheit zusammen mit denen der GNSS-Module und GNSS-Korrekturdaten eine genaue Flugbahn des Multikopters berechnet werden. Die Daten der inertialen Messeinheit werden hierbei regelmäßig durch die Daten der GNSS-Module gestützt.

### 3.1.3 Raspberry Pi 3 Typ B

Es wurde sich entschieden, die Datenverarbeitung mit einem Raspberry Pi 3 (siehe Abbildung 3.4) durchzuführen. Es handelt sich hierbei um einen von der Raspberry Pi Foundation entwickelten Einplatinencomputer. Die Stiftung gründete sich 2006, um einen erschwinglichen Computer zu entwickeln, an den Schüler direkt Hardware- und Elektronikprojekte entwickeln können. Die erste Version des Raspberry Pi kam im

Februar 2012 auf den Markt. Er verfügte über 256 MB Arbeitsspeicher und einen 700 MHz Ein-Kernprozessor. Das verwendete dritte Modell verfügt über einen Vier-Kern-Prozessor mit 1,2 Ghz und 1 GB Arbeitsspeicher. Bisher wurden alle Versionen zusammen über 11 Millionen mal verkauft. (Möcker, 2017)

Alle Modelle der Raspberry Pi Serie basieren auf Ein-Chip-Systemen des Halbleiterherstellers Broadcom. In diesem Chip sind die wichtigsten Bauteile des Systems integriert wie ein ARM-Prozessor, eine Grafikeinheit sowie verschiedene andere Komponenten. Die so gering gehaltene Anzahl an einzelnen Bauelementen beim Raspberry Pi ermöglichen den geringen Preis - ein Ziel der Raspberry Pi Foundation.

Der Vorteil des Raspberry Pi zur Datenverarbeitung sind vor allem seine verschiedenen Schnittstellen zur Daten Ein- und Ausgabe (RS Components Limited, 2015):

- 4 USB 2.0 Host-Anschlüsse
- Netzwerkschnittstelle (RJ45)
- Bluetooth- und WLAN
- 27 GPIO-Ports, nutzbar als (Schnabel, 2017)
  - Digitale Pins
  - Serielle Schnittstelle
  - I2C-Schnittstelle
  - SPI-Schnittstelle
- Stromversorgung 3,3V und 5V
- MicroUSB-Anschluss zur eigenen Stromversorgung (5V)
- MicroSD-Steckplatz
- verschiedene Video- und Audioausgänge

Außerdem vorteilhaft für die Nutzung am Multikopter ist seine geringe Größe und sein relativ geringer Stromverbrauch von maximal 12,5 Watt (RS Components Limited, 2015), welche aber im Betrieb ohne Peripherie nicht erreicht wird.

### **3.1.4 Multikopter Copterproject CineStar 6HL**

Bei einem Multikopter handelt es sich um ein Fluggerät mit drei oder mehr Rotoren. Es gibt entsprechend der Rotoranzahl verschiedene Modelle wie zum Beispiel den weit verbreiteten Quadrokopter oder den Hexakopter, welcher in dieser Arbeit betrachtet



Abbildung 3.4: Raspberry Pi 3 (eigene Aufnahme)

wird. Multikopter wurden ursprünglich für Militär- und Polizeizwecke eingesetzt, inzwischen sind sie aber auch vermehrt in kleineren Ausführungen im Privatbesitz für Videoaufnahmen zu finden (Heise Online, 2017). Angetrieben werden die handelsüblichen Modelle, welche eine Flugdauer von bis zu 30 Minuten und eine Tragkraft von bis zu fünf Kilogramm versprechen, mit Lithium-Polymer-Akkumulatoren (LiPo-Akkus). Die Anzahl und die maximale Umdrehung der Rotoren bestimmt die Schubkraft und somit auch die Tragkraft des Multikopters. Im Normalfall ist die Anzahl der Rotoren durch zwei teilbar, damit sich das auf das Traggestell wirkende Drehmoment aufhebt. Dies ist der große Vorteil gegenüber einem Hubschrauber, bei welchem mit einem Heckrotor dem Drehmoment um die Hochachse entgegengewirkt werden muss. Die einzelnen Motoren und Propeller werden kreuzweise angeordnet, so dass eine Drehzahländerung eines Propellerpaars zur Steuerung ausreicht. Vorteil eines Multikopters im Gegensatz zu einem Modellflugzeug ist es außerdem, dass er senkrecht starten kann und auch zum Beispiel für die Aufnahme von Bildern auf der Stelle stehen bleiben kann. Nachteil ist der höhere Energieverbrauch, so dass Flugzeuge bei gleicher Akkukapazität deutlich länger in der Luft bleiben können. (Bachfeld, 2013)

In dieser Arbeit soll der Multikopter den Laserscanner, die IMU, das Gimbal, die Stromversorgung, Datenverarbeitung und -speicherung im Betrieb tragen können. Bei der Systementwicklung des Multikopters muss daher darauf geachtet werden, dass das Gewicht möglichst gering bleibt und dennoch müssen die angehängten Messeinrichtungen auch für härtere Landungen ausgelegt sein. Der verwendete Hexakopter (siehe Abbildung 3.5) hat eine Tragkraft von maximal 5 Kilogramm und eine Flugdauer von bis zu 20 Minuten (Schulz, 2016).



Abbildung 3.5: Multikopter Copterproject CineStar 6HL mit Gimbal Freefly MöVI M5  
(eigene Aufnahme)

### 3.1.5 Gimbal Freefly MöVI M5

Um die Messgeräte während des Fluges des Multikopter zu stabilisieren und zu verhindern, dass sich jede Neigung der Flugsteuerung an den Laserscanner überträgt, wird ein sogenanntes Gimbal verwendet. Durch einen Regelkreis aus Motoren und einer inertialen Messeinheit (siehe auch Abschnitt 2.3), werden Neigungen und Drehungen in Echtzeit ausgeglichen. Außerdem ist es durch viele Gimbals möglich, die Messtechnik unabhängig vom Multikopter auszurichten - dies ist zum Beispiel bei der Luftbildaufnahme wichtig.

Für das Projekt wird ein Gimbal des Herstellers Freefly verwendet.

mehr...

## 3.2 Auswahl des Datenverarbeitungssystems

Ein Teil der Datenverarbeitung und die Speicherung soll direkt auf dem Sensorsystem durchgeführt werden. Da bei dem Betrieb des Multikopters jede weitere Masse die Laufzeit verkürzt, muss hierbei auf das Gewicht geachtet werden. Somit kommen für die Verarbeitung nur Ein-Chip-Computersysteme wie der Raspberry-Pi oder Mikrokontroller-Boards wie die der Arduino-Serie in Frage.

Vorteile eines Arduinos wären vor allem der geringere Stromverbrauch und die Echtzeitfähigkeit. Jedoch ist die Steuerung der Datenaufnahme über die Netzwerkschnittstelle und die Speicherung deutlich komplizierter und die Hardware nicht so leistungsfähig. Bei der Alternative, dem Raspberry-Pi übernimmt das Betriebssystem die grundlegenden Steuerungen, so dass nur noch die Daten selbst verarbeitet werden müssen. Außerdem bietet er mit der festverbauten Netzwerkschnittstelle und dem

Gerät	Laserscanner	IMU	Raspberry Pi
Spannung	9 - 18 V	10 - 36 V	5,0 V
max. Strom	0,9 A	0,75 A	2,5 A
typ. Leistung	8 W	7,5 W	12,5 W

Tabelle 3.1: Spannungs- und Strombedarf der einzelnen Module (Velodyne Lidar, 2017b; iMAR Navigation GmbH, 2015; RS Components Limited, 2015)

MicroSD-Karten- und der USB-Schnittstelle auch die komplette benötigte Hardware, die so nicht einzeln zusammengestellt und -gebaut werden muss.

### 3.3 Stromversorgung

Die Stromversorgung des Raspberry-Pi an der Drohne soll mittels Lithium-Ionen-Zellen erfolgen. Der Raspberry-Pi erfordert hierbei eine stabilisierte Spannungs- und Stromversorgung. Eine fehlerhafte Stromversorgung kann hierbei zu Systeminstabilitäten führen und so im schlimmsten Fall die Datenaufzeichnung komplett verhindern. Auf den genauen Aufbau einer solchen Versorgung wird hierbei verzichtet, sondern nur die Anforderungen an die Energiequelle erläutert.

Tabelle 3.1 listet die verschiedenen Module und die jeweils benötigte Energieversorgung auf. Der Multikopter mit der Gimbal verfügt über eine eigene Versorgung und muss daher nicht weiter beachtet werden. Außerdem hat hier eine eigene Akkukapazität auch Vorteile - auch bei einem zu hohen Verbrauch der Sensortechnik bleibt der Multikopter durch seine eigenständige Akku-Überwachung immer noch flugfähig um sicher landen zu können.

Für eine geplante Flugdauer von 30 Minuten wird bei einem angenommenen Wirkungsgrad von 90% eine Akkukapazität von mindestens 16 Wh (siehe Gleichung 3.2) benötigt. Außerdem muss ein Teil in 12 Volt und ein Teil mit 5V stabilisierter Spannung abgeben werden können. Gegebenenfalls sind hierfür auch zwei komplett unabhängige Spannungsquellen zu nutzen.

$$E = \frac{P \cdot t}{\eta} = \frac{(8 \text{ W} + 12,5 \text{ W} + 7,5 \text{ W}) \cdot 0,5 \text{ h}}{90 \%} \approx 15,6 \text{ Wh} \quad (3.2)$$

### 3.4 Anbindung des Raspberry Pi an den Laserscanner

Durch seine vielseitigen Anschlussmöglichkeiten bildet der Raspberry Pi den Sternpunkt der Schnittstellen. Der Laserscanner wird mit einem RJ45-Kabel an der Netzwerkschnittstelle angeschlossen. Die inertiale Messeinheit zeichnet die Daten selbst-

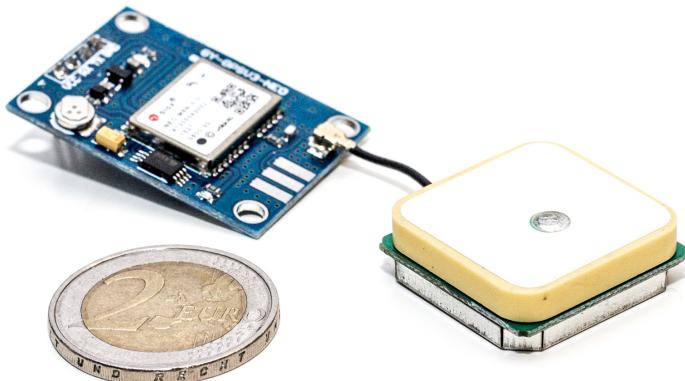


Abbildung 3.6: uBlox NEO-M8N, das Vorgängermodell NEO-6M mit PPS-Ausgang wurde verwendet (eigene Aufnahme)

ständig auf, kann aber auch mittels der als serieller Schnittstelle nutzbaren GPIO-Pins an den Raspberry Pi angeschlossen werden. Außerdem kann an diesem Port auch ein GNSS-Modul angeschlossen werden. Dieses GNSS-Modul kann im Folgenden dem Raspberry Pi zu einer genauen Uhrzeit verhelfen, die für die Verarbeitung der Daten benötigt wird. Alternativ kann auch ein an den Laserscanner angeschlossenes GNSS-Modul sein Zeitstempel per Netzwerk an den Raspberry Pi liefern. Diese Methode soll hier verwendet werden.

### 3.5 Verbindung des GNSS-Modules zum Laserscanner

Für die Versorgung des Laserscanners mit einem GNSS-Signal zur Synchronisierung wurde ein zusätzliches GNSS-Modul vom Typ uBlox NEO6M mit PPS-Signal ausgewählt (ähnlich dem auf Abbildung 3.6), da dieses kleiner und leichter ist, als die entsprechenden Adapterkabel der inertialen Messeinheit um dieses Signal zu nutzen.

Die Übertragung der Daten des GNSS-Modules zum Laserscanner erfolgt per serieller Schnittstelle über einen acht poligen Platinensteckverbinder. Bei dem vom Laserscanner benötigten Übertragungsprotokoll handelt es sich um das standardisierte NMEA-Protokoll, welches mit einer Datenrate von  $9600 \frac{\text{bit}}{\text{s}}$  und einer Signalspannung zwischen 3 und 15 Volt. Der direkte Anschluss eines uBlox GNSS-Modules vom Typ NEO-6M brachte zunächst keinen Erfolg. Messungen mit einem Arduino (siehe Abbildung 3.7) zeigten, dass das Signal des verwendeten GNSS-Moduls nicht dem im Datenblatt von Velodyne Lidar (2017a, S. 3) entsprach. Es zeigte sich, dass das Signal gedreht werden musste, da die Definition der Signalspannung verschieden war: Der Laserscanner benötigte ein Signal, bei dem Logisch 1 mit einer Spannung von über 3 Volt (Velodyne Lidar, 2017a, S. 3) codiert ist (HIGH), beim GNSS-Modul entspricht die höhere



Abbildung 3.7: Messung des Signals am uBlox NEO-6M (grün: Ausgangssignal; rot: Signal nach Nutzung eines Pegelwandler; 1000 Punkte entsprechen 5 Volt)

Spannung Logisch 0.

Um das Signal zu drehen wurde ein Integrierter Schaltkreis 74HC04 verwendet. Hierbei handelt es sich um ein Logikkonverter, der die HIGH- und LOW-Signale (Signal gegen Masse) tauscht. Der Laserscanner versorgt das GNSS-Modull nur mit 5 Volt Spannung, der GNSS-Chip benötigt jedoch eine Spannung von 3,3 Volt. Hierfür wurde ein Spannungsregler verwendet, der die Spannung auf 3,3 Volt stabilisiert. Zur weiteren Stabilisierung wurden Kondensatoren eingesetzt. In Kombination mit dem Logikkonverter dient dieser auch als Pegelwandler. Die genaue Schaltung ist Abbildung 3.8 zu entnehmen.

## 3.6 Steuerung im Betrieb

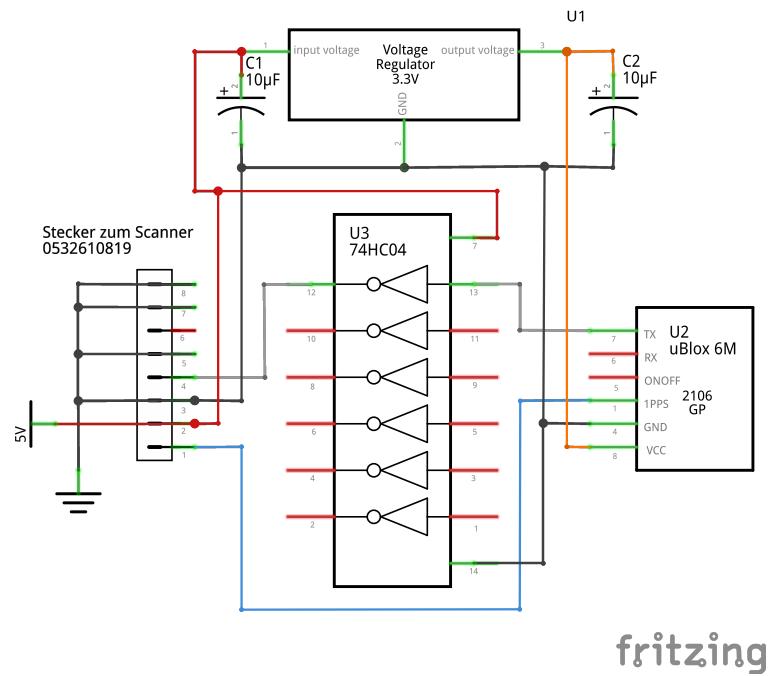
Der Betrieb des Raspberry Pi erfolgt im Betrieb ohne Tastatur und Bildschirm. Daher ist es notwendig, eine alternative Benutzerschnittstelle zu implementieren. Ein großer Steuerbedarf ist nicht gegeben, so dass wenige Tasten zum Stoppen der Datenaufzeichnung und zum Herunterfahren des Raspberry Pi ausreichend sind. Um auch eine Steuermöglichkeit zu implementieren, die im Flug genutzt werden kann, soll ein WLAN-Access-Point und ein simpler Webserver auf dem Raspberry Pi implementiert werden, der den Zugriff zum Beispiel über ein Smartphone oder Laptop ermöglicht.

Abbildung 3.9 zeigt den Schaltplan des entwickelten Steuermodules. Dieses bietet mit drei Leuchtdioden und 2 Tastern die Möglichkeit, im Skript später einfache Anzeigen und Eingaben zu realisieren. Hierfür wurde eine Erweiterung auf Basis des GPIO-Portes des Raspberry Pi aufgebaut. Die zwei Taster sind über die beiden GPIO-Pins 18 und 25 erreichbar. Ohne Betätigung werden die Eingänge über internen die Pull-Up-Widerstände () auf ein High-Level gezogen. Durch Drücken des Tasters wird die Spannung über die Widerstände auf ein Low-Level gezogen, welches durch das Python-Skript zur Laufzeit ausgelesen werden kann. Der Widerstand dient zur Strombegrenzung und als Sicherheit, falls die GPIO-Pins falsch geschaltet werden. Die drei Leuchtdioden wurden mit jeweils einem 150Ohm Vorwiderstand direkt zwischen einem GPIO-Pin

R?

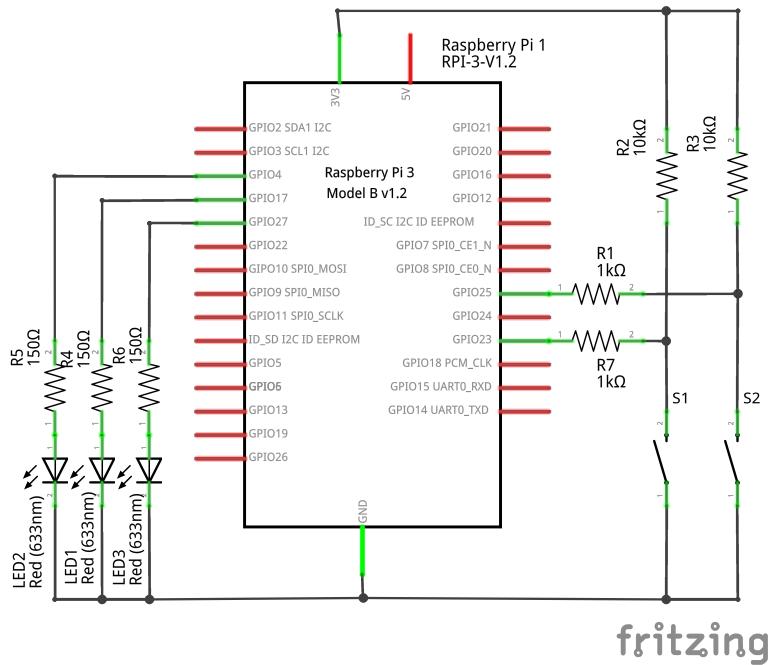
Widerstand

R?



fritzing

Abbildung 3.8: Entwurf des Schaltplanes zum Anschluss des GNSS-Modules an den Laserscanner, gezeichnet in Fritzing



fritzing

Abbildung 3.9: Entwurf des Schaltplanes für Steuerung des Raspberry, gezeichnet in Fritzing

und Ground eingebaut. Durch Ansteuerung der GPIO-Pins lassen sich diese An- und Abschalten. Außer zum Schutz und Betrieb der LEDs verhindern die Vorwiderstände auch eine zu hohe Stromaufnahme aus den GPIO-Pins. Die genaue Belastbarkeit der Pins ist nicht dokumentiert, jedoch wird meist von einem Wert um 10mA bei 3,3 Volt gesprochen (zum Beispiel Schnabel (2017)).

$$U_R = U_{GPIO} - U_{LED} = 3,3V - 2,0V = 1,3V \quad | \text{ Benötigter Spannungsabfall}$$

$$R = \frac{U_R}{I_{LED}} = \frac{1,3V}{0,01A} = 130\Omega \quad | \text{ min. Vorwiderstand} \quad (3.3)$$

## 3.7 Platinenentwurf und -realisierung

Nach dem Entwurf und Test der beiden Schaltungen aus Abbildung 3.8 und Abbildung 3.9 auf einem lötfreien Steckbrett, soll diese Schaltungen zum späteren Einsatz an Bord des Multikopters als Platine mit verlötzten Bauteilen erstellt werden. Vorteile der gelötzten Schaltung sind in diesem Projekt ihre höhere Widerstandsfähigkeit gegen Vibrationen und Korrosion. Durch die Vibrationen im Flug könnten sich so Bauteile lösen und im schlimmsten Fall zum Kurzschluss und somit zur Zerstörung führen. Auch können die Kontakte zwischen den Federklemmen und den Bauteilen durch den Betrieb außerhalb von Gebäuden durch Luftfeuchtigkeit korrodieren und somit der Kontaktwiderstand höher werden, was zu Störungen führen kann.

Für den Prototyp soll die Schaltung von Hand aufgebaut und verlözt werden. Erst in der zukünftigen Entwicklung, wenn die Schaltung ausreichend erprobt wurde, könnte es sinnvoll sein, eine Platine ätzen zu lassen. Als Platine kommen daher vorerst nur vorgefertigte Layouts in Frage:

- Lochrasterplatten (Platine mit einzelnen Lötpunkten)
- Streifenrasterplatine (Lötpunkte sind in Streifen verbunden)
- Punktstreifenrasterplatine (Streifenrasterplatine, bei denen die Streifen regelmäßig, zum Beispiel alle 4 Lötpunkte, unterbrochen sind)
- spezielle Aufsteckplatten für den Raspberry Pi

Da nur wenige Bauteile benötigt wurden, wurde eine Streifenrasterplatine gewählt. Bei einer solchen Platine sind alle Kontakte in einer Reihe mit einer Leiterbahn verbunden. Falls keine Verbindung gewünscht ist, kann diese Leiterbahn mit einem Messer oder ähnlichem unterbrochen werden. Da das Unterbrechen der Leiterbahn jedoch zeitaufwändig und fehlerträchtig ist, beispielsweise durch nicht vollständig getrennte Leiterbahnen, sollten diese beim Layouten der Platine möglichst vermieden werden. Auch

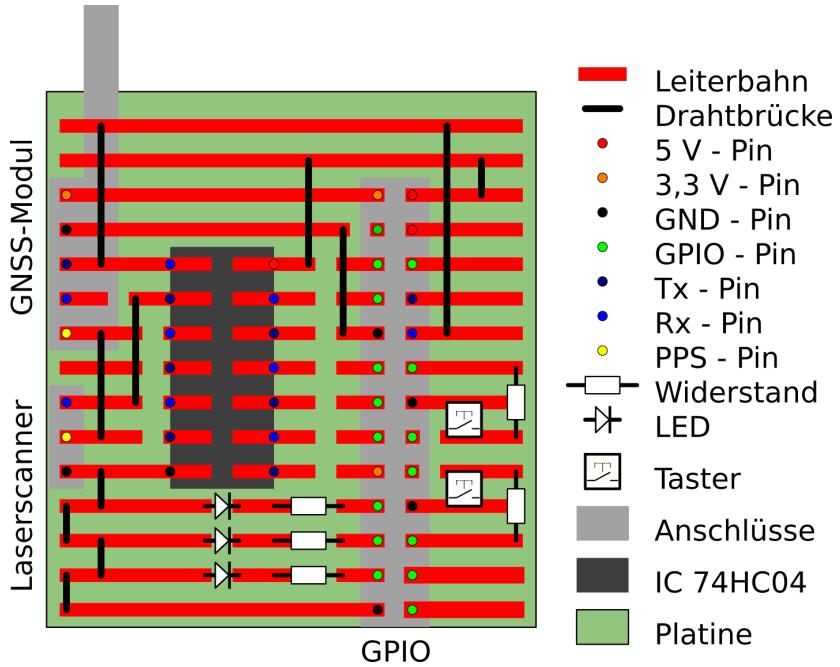


Abbildung 3.10: Layout der Lochstreifenplatine

sollte möglichst viele der benötigten Verbindungen durch diese Leiterbahnen erfolgen und möglichst wenig Drahtbrücken verwendet werden, die diese Leiterbahnreihen verbinden, da diese zusätzlichen Lötaufwand erfordern. Das endgültige Layout der Platine ist der Abbildung 3.10 zu entnehmen.

Beim Routing wurden noch einige Teile der Schaltung optimiert und versucht, einige Bauteile einzusparen, in dem zum Beispiel die Stromversorgung vom Raspberry Pi für den integrierten Schaltkreis und das GNSS-Modul verwendet wurden. Außerdem wurde die Auswahl der GPIO-Pins des Raspberry Pi platzsparender optimiert und nur die auch an dem ersten Typ des Raspberry Pi vorhandenen PINs genutzt. Hierdurch ist die Schaltung abwärtskompatibel zu allen Versionen des Raspberry Pi. Der endgültige Schaltplan ist Abbildung 3.11 zu entnehmen. Für die Taster wurden hier die internen Pull-Up-Widerstände genutzt, so dass hier zwei Widerstände eingespart werden konnten. Außerdem wurde der Datensendeport (Tx) vom GNSS-Modul an die serielle Schnittstelle des Raspberry Pi angeschlossen, so dass der Raspberry Pi nun auch ohne den Umweg über den Laserscanner die Daten vom GNSS-Modul empfangen kann.

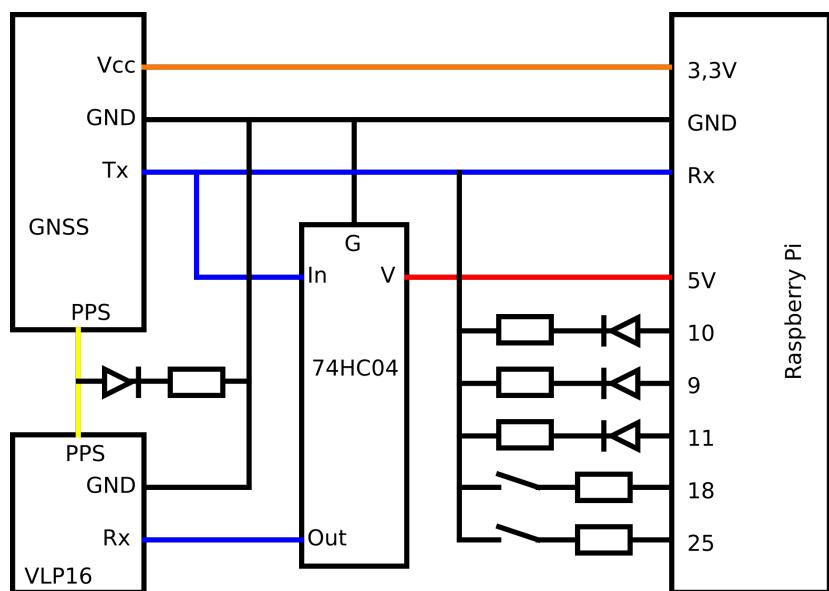


Abbildung 3.11: Vereinfachter, endgültiger Schaltplan

# 4 Theoretische Datenverarbeitung

## 4.1 Verwendung von Python

Zur Realisierung der Programmierung wurde die Skriptsprache Python ausgewählt. Python bietet den Vorteil vergleichsweise kurzen und gut lesbaren Programmierstil zu fördern. Hierfür werden unter anderem nicht Klammern zur Bildung von Blöcken genutzt, sondern Texteinrückungen verpflichtend hierfür eingesetzt (Theis, 2011, S. 13f). Die Struktur des Programmes ist so schnell erfassbar. Außerdem ist es nicht notwendig, den Quellcode zu kompilieren. Er wird vom Interpreter direkt ausgeführt. So sind kurze Entwicklungszyklen ohne (zeit-)aufwändiges Kompilieren möglich. Änderungen und Anpassungen können schnell durchgeführt werden.

Python wurde in seiner ersten Version 1991 von Guido van Rossum freigegeben. Sein Ziel war es, eine einfach zu erlernende Programmiersprache zu entwickeln, die der Nachfolger der Sprache ABC werden sollte. Außerdem sollte die Sprache leicht erweiterbar sein und schon von Haus aus eine umfangreiche Standardbibliothek bieten. Python bietet mehrere Programmierparadigmen an, so dass je nach zu lösendem Problem objektorientiert oder strukturiert programmiert werden kann (Theis, 2011, S. 14).

Die aktuelle Version von Python (Oktober 2017) ist die Version 3.6. Das Skript wurde unter Verwendung dieser Version entwickelt. Es wurde aber auch auf eine Kompatibilität mit Python 2.7, der neusten Version von Python 2, die noch sehr häufig im Einsatz ist, geachtet. Um Teile des Quellcodes als Python-Module auch in andere Skripte einfach einbinden zu können, aber auch den Quelltext übersichtlich zu halten, wurde der objektorientierte Programmierstil gewählt.

## 4.2 Datenlieferung vom Laserscanner

Der Laserscanner Velodyne VLP-16 liefert seine Daten als UDP-Netzwerkpakete in einem proprietären binären Datenformat. Diese Daten sind nicht direkt lesbar sondern müssen vor einer weiteren Nutzung aufbereitet und umgeformt werden. Dies soll mittels des in dieser Arbeit entwickelten Skriptes durchgeführt werden.

Ein Datenpaket (siehe Tabelle 4.1) besteht jeweils aus einem Header von 42 Bytes, gefolgt von 12 Datenblöcken mit jeweils 32 Messungen, abgeschlossen von 4 Bytes, die

Header			Netzwerk-Header	42 Bytes	
Block 1	0-1		Flag	2 Bytes	
	2-3		Horizontalrichtung	2 Bytes	
	Messung 1	4-5	Entfernung	2 Bytes	
		6	Reflektivität	1 Byte	
	Messung 2	7-8	Entfernung	2 Bytes	
		9	Reflektivität	1 Byte	
Messungen 3 - 32					
Block 2 - 12					
Time		1200-1204	Zeitstempel	4 Bytes	
Factory		1205-1206	Return-Modus	2 Bytes	

Tabelle 4.1: Aufbau der Daten des Netzwerkpaketes, nach Velodyne Lidar (2016)

den Zeitstempel angeben und 2 Bytes, die den eingestellten Scan-Modus zurückliefern. Jeder Datenblock enthält die aktuelle horizontale Ausrichtung des rotierenden Lasers und darauf folgend die Messwerte von zwei Messungen der 16 Laserstrahlen. Die genaue Horizontalrichtung zum Zeitpunkt der Messung muss aus den Horizontalrichtungen aus zwei aufeinander folgenden Messungen interpoliert werden.

Der Laserscanner sendet bei der Einstellung Dual Return, also der Rückgabe vom stärksten und letzten Echo pro Messung bis zu 1508 Pakete dieser Form pro Sekunde (Velodyne Lidar, 2016, S. 49). Die Ausgangsdaten werden, bei einer Paketgröße von 1248 Bytes mit einer Datenrate von 1,8 MB/s empfangen (siehe Gleichung 4.2). Hierbei werden fast 600.000 Messwerte pro Sekunde übertragen (siehe Gleichung 4.1).

$$1508 \frac{\text{Pakete}}{\text{Sekunde}} \cdot 12 \frac{\text{Datenblöcke}}{\text{Paket}} \cdot 32 \frac{\text{Messungen}}{\text{Datenblock}} = 579.072 \frac{\text{Datensätze}}{\text{Sekunde}} \quad (4.1)$$

$$1508 \frac{\text{Pakete}}{\text{Sekunde}} \cdot 1248 \frac{\text{Bytes}}{\text{Paket}} = 1,79 \text{ MB/s} \quad (4.2)$$

### 4.3 Geplantes Datenmodell

Die Daten des Laserscanners sollen in einer einfach lesbaren Textdatei abgelegt werden. In der Nachbereitung sollen die Daten aus dieser Textdatei mit den Daten der inertialen Messeinheit und des GNSS-Empfängers verknüpft werden, um so die Daten georeferenzieren zu können. Als Verknüpfung bietet sich hier der Zeitstempel an. Die inertialen

Messeinheit und der Laserscanner können hierbei die Zeitdaten aus dem GNSS-Signal verwenden. Hierdurch sind hochgenaue Zeitstempel möglich. Die Zeitinformation bildet also einen wichtigen Schlüssel in den Daten. Als einfaches Textformat wurden durch Tabulator getrennte Daten, jeweils eine Zeile je Messung, gewählt. Folgende Daten sind in dieser Reihenfolge enthalten:

- Zeitstempel in Mikrosekunden
- Richtung der Messung in der Rotationsebene in Grad
- Höhenwinkel zur Rotationsebene in Grad
- Gemessene Entfernung in Metern
- Reflektivität auf einer Skala von 0 bis 255

Problematisch ist bei diesem Datenmodell jedoch die benötigte Datenrate. Eine Datenzeile erfordert 29 Bytes und somit wird bei über einer halben Million Messungen pro Sekunde (siehe Gleichung 4.1) eine Datenschreibrate von mindestens 16 MB/s benötigt (siehe Gleichung 4.3). Da das Schreiben nicht dauerhaft erfolgt, sollte die Datenrate bevorzugt deutlich höher sein.

$$579.072 \frac{\text{Datensätze}}{\text{Sekunde}} \cdot 29 \frac{\text{Bytes}}{\text{Datenzeile}} = 16,02 \text{ MB/s} \quad (4.3)$$

Erste Tests ergaben, dass diese Verarbeitungsgeschwindigkeit nicht mit dem Raspberry Pi erreicht werden konnte. Außerdem benötigen die Daten sehr viel Speicher. Daher wurde sich später für eine Hybridlösung entschieden (siehe Kapitel 5).

## 4.4 Weiterverarbeitung der Daten zu Koordinaten

Die als Text gespeicherten Rohdaten sollen dann im Rahmen einer weiterführenden Arbeit zu Koordinaten umgewandelt werden. Zu dieser Umwandlung werden die Positionen des Laserscanners mittels dem GNSS-Empfänger in der IMU und die Neigungsdaten aus der IMU verwendet. Die Neigungen werden dazu direkt mit den Winkeldaten verrechnet.

Bei der Berechnung ist jedoch zu beachten, dass der Ursprungsort der Entfernungsmeßung zwar in der Drehachse des Laserscanners liegt, jedoch der Ursprungsort der ausgesendeten Strahlen etwa 40mm in Strahlrichtung verschoben ist (siehe Abbildung 4.1). Bei der Streckenberechnung ist diese Strecke mit enthalten, jedoch kann zur Berechnung der Z-Komponente der lokalen Koordinaten nicht einfach der Höhenwinkel und die gemessene Strecke verwendet werden. Die lokalen Koordinaten berechnen sich somit nach der Gleichung 4.4.

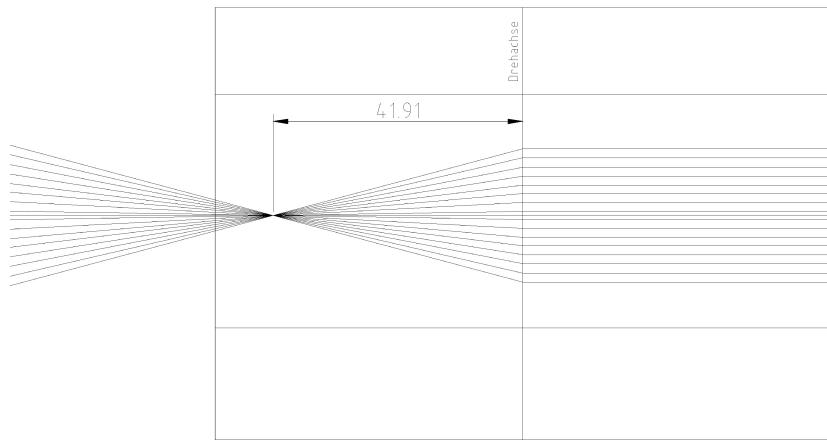


Abbildung 4.1: Strahlengang im Laserscanner VLP-16, Werte in Millimetern, nach Velodyne Lidar (2014)

$h$  : Höhenwinkel ( $-15^\circ - 15^\circ$ )

$r$  : Horizontalrichtung ( $0^\circ - 360^\circ$ )

$s$  : Gemessene Strecke

$$\begin{aligned} s_S &= s - 41,91 \text{ mm} && | \text{ Schrägstrecke nach dem Fokuspunkt} \\ s_H &= s_S * \cos(h) + 41,91 \text{ mm} && | \text{ Horizontalstrecke von der Drehachse} \end{aligned} \quad (4.4)$$

$$X = s_H \cdot \sin(r) \quad | \text{ Y-Achse in Nullrichtung}$$

$$Y = s_H \cdot \cos(r)$$

$$z = s_S \cdot \sin(h)$$

## 4.5 Anforderungen an das Skript

Aus den technischen Vorgaben ergeben sich dann folgende Funktionen, die das Skript aufweisen muss:

- Rohdaten vom Scanner abrufen
- Zeit vom GNSS-Modul abrufen
- Steuerungsmöglichkeit mittels Hard- und Software
- Umwandlung in eigenes Datenmodell

Der Ablauf der einzelnen Schritte ist oft abhängig vom Fortschritt anderer Schritte und Gegebenheiten. Daher wurden die benötigten, einzelnen Schritte vorerst als grober Ablaufplan skizziert. So hat der Raspberry Pi keinen eigenen Zeitgeber. Um die Dateien aber mit dem korrekten Zeitstempel zu versehen, ist daher eine aktuelle Uhrzeit notwendig - diese liefert das GNSS-Modul, welches am Laserscanner angeschlossen ist, sofern ein GNSS-Fix besteht. Es muss also vor dem Erzeugen der Dateien auf ein gültiges GNSS-Signal gewartet werden. Der endgültige, vereinfachte Ablaufplan ist der Abbildung 4.2 zu entnehmen.

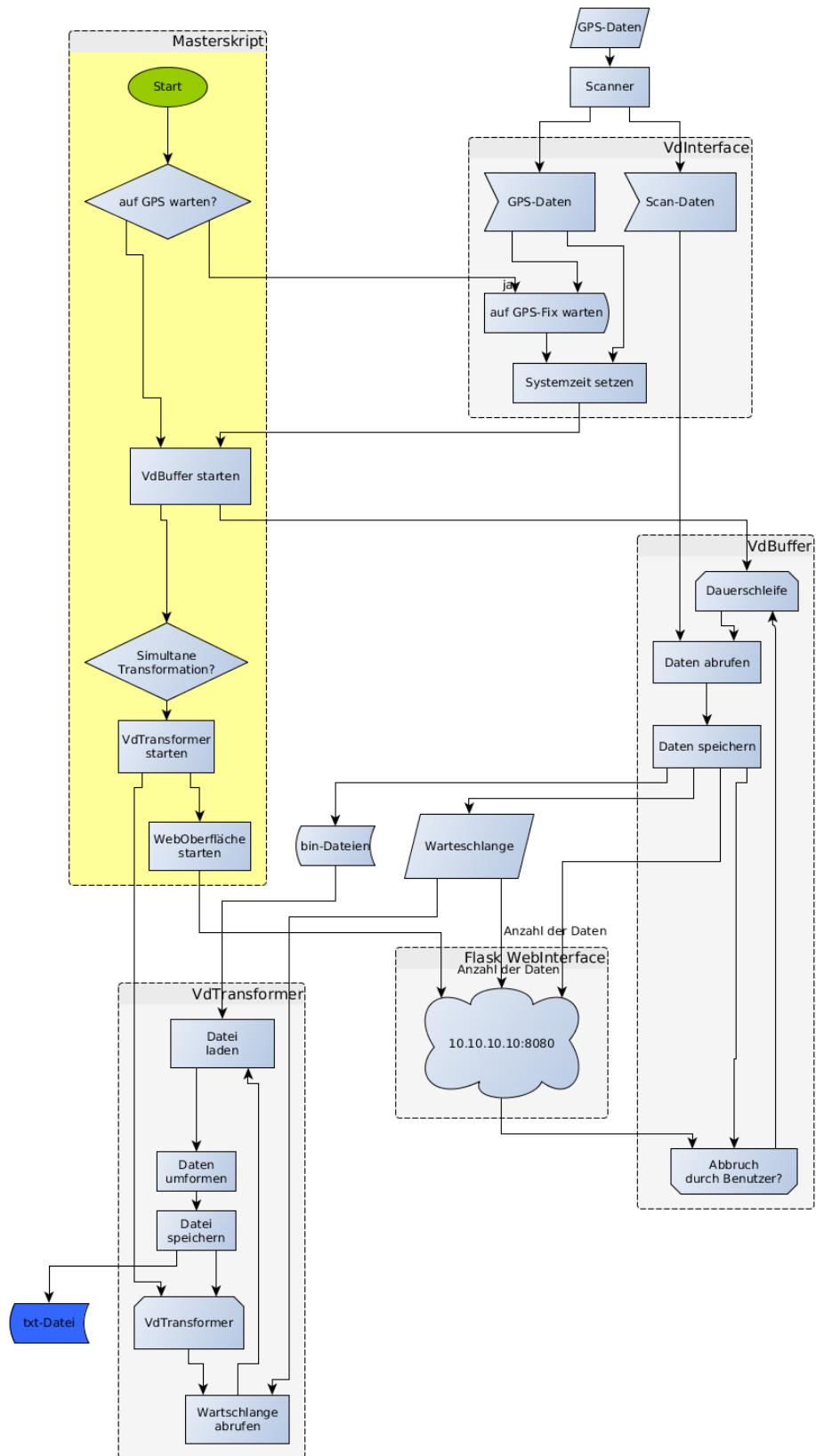


Abbildung 4.2: Vereinfachter Ablaufplan des Skriptes

# 5 Entwicklung des Skriptes

## 5.1 Klassenentwurf

Da das Skript objektorientiert programmiert werden soll, wurde zunächst mit Hilfe des Ablaufplanes aus Abbildung 4.2 die benötigten Klassen entworfen. Die endgültigen Klassen sind der Abbildung 5.1 zu entnehmen. Auf die genauen Funktionen der einzelnen Klassen wird im Abschnitt 5.4 eingegangen.

## 5.2 Evaluation einzelner Methoden

Um eine einfache Fehlersuche zu ermöglichen, wurden die grundlegenden Funktionen in einzelnen Skripten entwickelt und geprüft. Diese kleineren Skripte haben den Vorteil, dass Fehler schneller eingegrenzt und auch schon früh konzeptionelle Fehler entdeckt werden können. In diesem Schritt wurde bemerkt, dass ein großes Problem die Geschwindigkeit der Datenverarbeitung ist.

**Datenempfang** Die Verbindung zum Laserscanner mittels Python-Socket funktionierte ohne weitere Probleme. Die binären Daten konnten zeitgleich abgespeichert werden.

**Datentransformation** Zunächst war es geplant, die Daten direkt in das in Abschnitt 4.3 vorgestellte Datenmodell umzuformen. Hierzu sollte der Empfang der Daten direkt eine Umformmethode starten. Die Versuche erfolgten zunächst mit dem im vorherigen Test aufgezeichneten Daten. Schon hier zeigte sich, dass die Umwandlung der aufgezeichneten Daten etwa das Fünffache der Mess- und Aufzeichnungzeit beanspruchte. Wie erwartet, brachte auch das direkte Einlesen der Daten vom Scanner keinen Erfolg. Es folgte ein Überlauf des Netzwerk-Buffers und somit der Verlust von Messdaten. Grund hierfür war hauptsächlich die benötigte Prozessorzeit. Die Nutzung einer schnelleren Datenspeicherung auf einer Solid-State-Disk mit einer Schreibrate von bis zu 300 MB/- Sekunde änderte nichts an der Geschwindigkeit des Skriptes. Auch das Erzeugen eines neuen Threads für jeden empfangenen Datensatz war nicht erfolgsversprechend, da bis zu 1500 Threads pro Sekunde hierdurch gestartet wurden und das gesamte System überlastet wurde. Die Umformung musste daher von dem Datenempfang entkoppelt

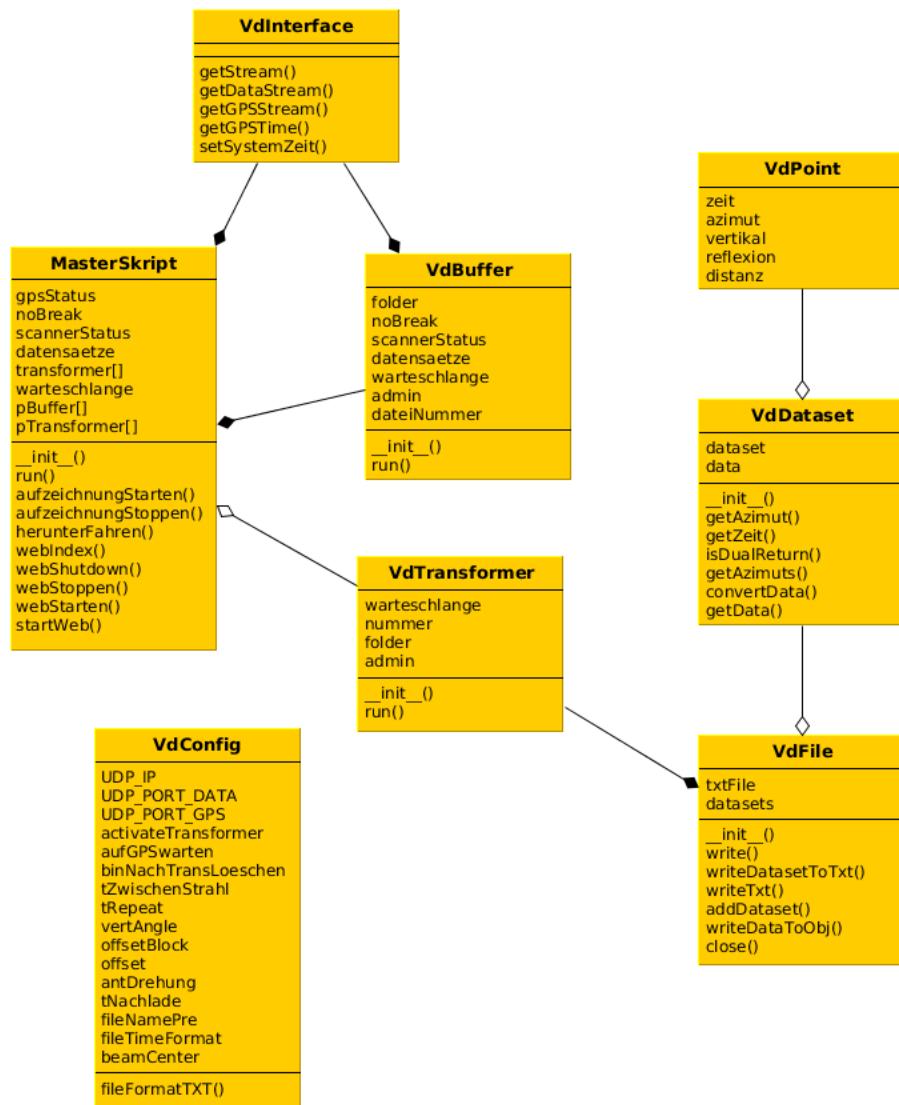


Abbildung 5.1: UML-Klassendiagramm

werden und das Skript für die Nutzung von Mehrkernprozessoren optimiert werden. Threads in Python laufen dennoch in einem Prozess und somit nur auf einem Prozessor. Es wurde das in Abschnitt 5.3 vorgestellte Multikern-Konzept erarbeitet.

**Hardware-Steuerung** Ein Tastendruck auf dem Steuerungsmodul (siehe Abschnitt 3.6) sollte den Raspberry Pi zum Beispiel herunterfahren. Auch dieses Skript wurde getestet. Ein Problem hierbei war es, dass das Skript Administratorrechte (**root**) benötigte, um den Rechner herunterfahren zu können. Hierfür wurde jedoch eine Lösung gefunden, indem dem Nutzer **pi** die entsprechenden Rechte zum Herunterfahren gegeben wurden (siehe Abschnitt 6.2). Eher zufällig zeigte sich aber noch ein anderes Problem: Sofern das Skript automatisch mit dem Start des Raspberry Pi gestartet wurde und das Steuermodul nicht angeschlossen war, fuhr der Raspberry Pi automatisch nach wenigen Sekunden Betrieb herunter. Da mit dem fehlenden Modul auch die Pull-Down-Widerstände fehlten, war der GPIO-Pin auf einem nicht definierten Zustand. Es kam dazu, dass er zufällig auf einem HIGH-Niveau war, welches als Drücken des Tasters interpretiert wurde. Nach Überschreiten der konfigurierten Haltezeit des Ausschalters von zwei Sekunden, wurde der Herunterfahrprozess gestartet. Um dieses Problem zu unterdrücken, wurde dem Skript zuerst eine vorherige Abfrage hinzugefügt, die beim Start überprüft, ob die beiden Taster sich auf einem Low-Niveau befinden, dass durch die beiden angeschlossenen Pull-Down-Widerstände erreicht wird. Falls dieses nicht der Fall ist, beendet sich die Hardwaresteuerung selbstständig. Im weiteren Verlauf der Entwicklung wurden dann jedoch das Signal gedreht und die internen Pull-Down-Widerstände des Raspberry Pi verwendet. Somit wurde diese Abfrage überflüssig.

## 5.3 Multikern-Verarbeitung der Daten

Da bei der Evaluation der einzelnen Methoden herausgefunden wurde, dass die Verarbeitungsgeschwindigkeit des Raspberry Pi für eine sofortige Transformation der Daten nach deren Eingang zu langsam ist, wurde ein Konzept erarbeitet, den hierdurch auftretenden Messdatenverlust zu unterdrücken.

Der Verarbeitung musste ein weiterer Buffer vorgeschaltet werden. Da aber das Abschalten des Raspberry Pi, zum Beispiel durch einen Verlust der Energieversorgung, nicht zu Datenverlusten führen sollte, konnten nicht die in Python integrierten Funktionen zur Datenzwischenspeicherung verwendet werden – diese setzen zur Zwischenspeicherung auf den Arbeitsspeicher, der durch Stromverlust gelöscht wird. Das dauerhafte Schreiben auf die Festplatte – im Fall des Raspberry Pi einer MicroSD-Speicherkarte – führt aber zur weiteren Verzögerung. Es wurde daher eine Hybridlösung erarbeitet.

Die Arbeit wird nun auf mehrere Prozesse verteilt:

- Start des Skriptes und Gesamtsteuerung in Prozess mit mittlerer Priorität (Klasse

`VdAutoStart`, mit Threads für Weboberfläche (Methode `startWeb()` in Klasse `VdAutoStart` und Hardwaresteuerung (Klasse `VdHardware`)

- Sammeln der Daten mit höchster Priorität (Klasse `VdBuffer`)
- Umformen der Daten durch mehrere Prozesse je nach Prozessorkernanzahl mit erhöhter Priorität (Klasse `VdTransformer`)

Die Daten werden nun zuerst für wenige Sekunden im Arbeitsspeicher gesammelt. Sofern 7.500 Datensätze zwischengespeichert wurden – je nach Einstellung des Laser-scanners in etwa fünf oder zehn Sekunden – werden diese im Dateisystem als binäre Datei abgelegt und der Dateiname in einer Warteschlange aus dem `multiprocessing`-Modul von Python (Klasse `Queue`) abgelegt. Die Prozesse zum Umformen der Daten fragen diese Warteschlange nun ab, verarbeiten jeweils eine binäre Datei und hängen die Ergebnisse an eine Ergebnis-Textdatei an. Die Dateinamen der binären Dateien, dessen Bearbeitung begonnen wurde, werden aus der Warteschlange entfernt. Nach dem Schreiben der umgeformten Daten werden die binären Dateien aus dem Dateisystem entfernt. Damit die Umformer-Prozesse beim Schreiben nicht auf einander warten müssen, schreibt jeder Prozess in eine andere Ergebnisdatei. Diese können nach der Messung einfach zusammengefügt werden. Falls nun zum Beispiel die Stromversorgung unterbrochen wird, sind nur die Daten der maximal letzten 10 Sekunden verloren. Daten, die älter sind, sind entweder als binäre Daten oder als Textdatei gespeichert. Durch neues Starten des Umformerprozesses können die restlichen, noch nicht gewandelten Daten umgeformt werden.

Durch dieses Prinzip stört eine stockende Datenumformung nicht die Aufzeichnung der Daten vom Scanner. Sofern der Raspberry Pi nicht die Geschwindigkeit der Umformung halten kann, werden einfach mehr binäre Dateien zwischengespeichert, die gegebenenfalls im Postprocessing umgewandelt werden können.

## 5.4 Klassen

Es folgt die Beschreibung der einzelnen Klassen. Auf die Konstruktor-Methoden `__init__()` wird nicht eingegangen, da hier meist nur Variablen deklariert werden.

### 5.4.1 VdAutoStart

Die Klasse mit dem Namen `VdAutoStart` (siehe Anhang A.1) steuert den automatischen Start des Skriptes beim Hochfahren des Raspberry Pi. Sie ist verantwortlich für den korrekten Start der einzelnen Skriptteile in der richtigen Reihenfolge. Außerdem sind in der zugehörigen Datei auch alle Programmteile abgelegt, die nicht zu einer Klasse gehören, wie zum Beispiel der Startaufruf.

Zu Beginn wird geprüft, auf welcher Umgebung das Skript läuft. Sofern es auf einem Raspberry Pi läuft, werden später zusätzliche Klassen aufgerufen.

**run()** Das Skript startet mit dieser Methode. Als erstes wird die Hardwaresteuerung in einem neuen Thread aktiviert, sofern es sich bei der Umgebung um einen Raspberry Pi handelt und die entsprechenden Module zur Steuerung der GPIO-Pins installiert sind. Sofern auf ein GNSS-Fix zur Einstellung der Uhrzeit gewartet werden soll, wird zunächst die Methode `getGNSSTime()` der Klasse `VdInterface` aufgerufen. Außerdem wird der für die zu speichernden Daten genutzte Ordner angelegt.

Folgend wird ein Prozess der Klasse `VdBuffer` gestartet. Um eine Kommunikation zu diesem neuen Prozess zu ermöglichen, werden einige Pipes und eine Warteschlange (Queue) mit an den neuen Prozess übergeben.

Sofern die Daten simultan transformiert werden sollen, wird abgefragt, wie viele Prozessoren dem System zur Verfügung stehen und eine entsprechende Anzahl an Transformer-Prozessen gestartet ( $n - 1$ , mindestens 1). Auch hier werden zur Kommunikation Pipes und die Queue verwendet.

**aufzeichnungStarten()** Die Methode startet den Buffer-Vorgang des `VdBuffer`-Prozesses. Sie wird durch die Steuersysteme aufgerufen.

**aufzeichnungStoppen()** Diese Methode stoppt die Aufzeichnung durch den `VdBuffer`-Prozess und wird auch durch die Steuersysteme genutzt.

**herunterfahren()** Ermöglicht den Steuersystemen, dass die `VdBuffer`- und `VdTransformer`-Prozesse zu unterbrechen und das System herunterzufahren.

**\_\_main\_\_( )** Die Methode `__main__( )` gehört nicht zu der Klasse sondern ist nur mit in dieser Datei abgelegt. Sie erzeugt ein Objekt der Klasse `VdAutoStart` und startet die `run()`-Methode. Außerdem wird die Weboberfläche hieraus gestartet.

**Flask-Webinterface app** Die Weboberfläche zur Steuerung wird mit dem Modul `Flask` erzeugt. Die Weboberfläche wird durch die `main`-Methode in einem zusätzlichen Thread gestartet. Die Weboberfläche ist entsprechend ihrem geplanten Einsatzzweck optimiert für die mobile Anzeige auf Smartphones, lässt sich aber auch vom Laptop bedienen. Die Oberfläche selbst nutzt nur HTML und CSS - ist also nicht zusätzlichen Skriptsprachen auf dem verwendeten Gerät abhängig.

## 5.4.2 VdInterface

Die Klasse `VdInterface` (siehe Anhang A.4) übernimmt die Kommunikation mit dem Laserscanner.

**getDataStream()** Die Methode öffnet den Netzwerk-Stream, der die Messdaten des Laserscanners überträgt. Das Auslesen der Daten aus dem Stream erfolgt dann in der Klasse `VdBuffer`.

**getGNSSStream()** Durch die Methode wird der Netzwerkstream geöffnet, der die aktuellen Datensätze des an den Laserscanner angeschlossenen GNSS-Modules überträgt. Aus den Daten kann zum Beispiel die Uhrzeit gewonnen werden.

**getStream()** Da die benötigten Schritte zum Öffnen der beiden vorher vorgestellten Streams identisch sind, wurden diese Funktionalitäten in diese Methode ausgelagert, um den Code möglichst redundanzfrei zu halten.

**getGNSTime()** Diese Methode fragt die Daten, die über den GNSS-Netzwerkstream geliefert werden, solange ab, bis der NMEA-Datensatz eine GPRMC-Nachricht mit einem GNSS-Fix, also einer gültigen Position, enthält. Diese Nachricht enthält außer der aktuellen Position und dem GNSS-Fix-Status auch den aktuellen Datums- und Zeitstempel. Die erkannte Uhrzeit wird als Python-timestamp an die Methode `setSystemZeit()` weitergegeben.

**setSystemZeit()** Die Methode setzt die aktuelle Systemzeit auf Basis eines ihr übergebenen Zeitstempels. Hierzu werden Befehle des Linuxbetriebssystems angesprochen, dessen Verwendung vorher freigegeben werden muss (siehe Abschnitt 6.2). Zuerst wird die Netzwerk-Zeitsynchronisierung abgeschaltet, dann die Uhrzeit gesetzt und die Synchronisierung wieder aktiviert. Die Deaktivierung ist notwendig, da ansonsten Linux keine Änderung an der Uhrzeit erlaubt. Eine komplette Deaktivierung der Netzwerk-Zeitsynchronisierung ist nicht zu empfehlen, da so die Uhr immer manuell gestellt werden muss.

## 5.4.3 VdHardware

Die Klasse `VdHardware` übernimmt die Hardwaresteuerung des Raspberry Pi. Um etwas unabhängiger zu sein, stellt die Klasse einen eigenen Thread dar, der von der Klasse `VdAutoStart` je nach Hardware gestartet wird.

Bei der Initialisierung der Klasse werden die GPIO-Ports des Raspberry Pi entsprechend des Hardwaresteuerungsmodules aus Abschnitt 3.6 eingerichtet. Hierbei werden

bei den Eingangsports die internen Pull-Up-Widerstände aktiviert. Beim Start des Threads wird dann zusätzlich ein Eventhandler für die Eingangspins eingerichtet, der entsprechende Funktionen zum Starten oder Stoppen der Aufzeichnung sowie zum Herunterfahren aufrufen. Des Weiteren wird ein Timer aktiviert, der dafür sorgt, dass die LED-Anzeigen einmal sekündlich aktualisiert werden.

#### 5.4.4 **VdPoint**

Die **VdPoint**-Klasse stellt einen Messpunkt der Velodyne dar. Er nimmt als Attribute die Messdaten auf.

#### 5.4.5 **VdDataset**

Die Klasse **VdDataset** nimmt eine Menge von Messdaten auf. Diese Klasse sorgt auch für das Interpretieren der binären Daten vom Laserscanner. Die Daten werden dann als **VdPoint**-Objekte in einer Liste gesichert. Durch die Übergabe der Daten an die Klasse **VdFile** können diese dann als Datei gespeichert werden.

#### **getAzimuts()**

#### 5.4.6 **VdFile**

#### 5.4.7 **VdBuffer**

#### 5.4.8 **VdTransformer**

#### 5.4.9 **VdConfig**

### 5.5 Beispiel-Quelltext-Zitat

Die Daten werden eingelesen (siehe Zeile 18, Listing 5.1)

Listing 5.1: Quelltext-Test

```
15     creates and fills an ascii-file with point data
16     """
18     def __init__(self, conf, filename="", fileformat="txt"):
19         """
20             Creates a new ascii-file
```

# 6 Konfiguration des Raspberry Pi

Als Grundlage wurde auf die MicroSD-Karte, die dem Raspberry Pi als Festplatte dient, das Betriebssystem Raspbian aufgespielt. Hierbei handelt es sich um ein Derivat von Debian GNU/Linux, das speziell auf die Hardware des Raspberry Pi angepasst wurde. Die aktuelle Version (Stand 27.10.2017) nennt sich Raspian Stretch. Für die Verwendung als Verarbeitungsgerät ohne angeschlossenen Display reicht die Variante ohne grafische Benutzeroberfläche aus (Raspbian Stretch Lite). Die Konfiguration des Raspberry Pi erfolgt vollständig über Konfigurationsdateien. In dieser Arbeit erfolgte die Konfiguration per Fernzugriff über SSH, einem Standard für das Fernsteuern der Konsole über das Netzwerk. Eine Konfiguration hätte aber auch mittels einem angeschlossenen Display und einer USB-Tastatur erfolgen können.

Die Änderungen der Konfigurationsdateien erfolgten mit dem vorinstallierten Editor `nano` unter Nutzung von Administratorrechten. Ein solcher Aufruf erfolgt zum Beispiel mit dem Befehl `sudo nano /pfad/zur/konfiguration.txt`. Nachfolgend müssen die betroffenen Programme oder sogar das komplette Betriebssystem neugestartet werden. Der Neustart eines Services erfolgt zum Beispiel mit dem Aufruf `sudo service programmname restart`, der Neustart des Betriebssystems mit `sudo shutdown -r now`. Es empfiehlt sich, von allen zu ändernden Konfigurationsdateien Sicherungskopien anzulegen. Dies erfolgt zum Beispiel mit `sudo cp original.txt original.old.txt` (Kopieren) oder `sudo mv original.txt original.old.txt` (Verschieben, zum Beispiel zum Anlegen einer komplett neuen Datei). Auf diese Linux-Grundlagen wird im Folgenden nicht mehr eingegangen.

## 6.1 Installation von Raspbian

Die Installation von Raspbian erfolgt durch das Entpacken des Installationspaketes von der Website der Raspberry Pi Foundation auf einer leeren MicroSD-Karte mit dem Tool `Etcher`. Auf der nach dem Entpacken erzeugten boot-Partition wird eine leere Datei mit dem Namen `ssh` angelegt. Hierdurch wird sofort nach dem Start der SSH-Zugang über das Netzwerk zum Raspberry Pi ermöglicht, die IP-Adresse wird per DHCP, zum Beispiel von einem im Netzwerk vorhandenen Router, bezogen. Nach dem Einloggen zum Beispiel unter Linux mit dem Befehl `ssh pi@raspberrypi` und dem Passwort `raspberry`, kann mittels `passwd` das Passwort verändert werden.

	Schnittstelle	IP-Adresse bzw. Bereich	
Laserscanner	Ethernet	192.168.1.111	statisch
Raspberry Pi	Ethernet	192.168.2.110	statisch
	WiFi	10.10.10.10	statisch
Client	WiFi	10.10.10.100	- 10.10.10.254

Tabelle 6.1: IP-Adressen-Verteilung

## 6.2 Befehle mit Root-Rechten

Linux erlaubt das Ändern der Zeit und das Herunterfahren über die Kommandozeile nur dem Administrator (`root`). Da es jedoch nicht empfohlen ist, Skripte als `root` auszuführen, muss hier eine andere Lösung gefunden werden, um den Skripten die Möglichkeit zu geben, den Raspberry Pi auf Tastendruck oder per Web-Steuerung herunterzufahren. Hierfür wurden dem normalen Nutzer (`pi`) die Rechte gegeben, einzelne Befehle als Admin ohne Passwortabfrage auszuführen. Diese Rechte können dem Nutzer durch Eintragung in die Konfigurationsdatei `/etc/sudoers` gegeben werden. Da eine fehlerhafte Änderung der Datei den kompletten Administratorzugang zum System versperren kann, wird die Datei mit dem Befehl `visudo` überarbeitet, der nach dem Editieren die Datei auf Fehler prüft. Die zusätzlichen Einträge in der Konfiguration sind dem Listing 6.1 zu entnehmen.(ubuntuusers.de, 2017)

Listing 6.1: Änderung der `/etc/sudoers`

```

1 # Cmnd alias specification
2 Cmnd_Alias VLP = /sbin/shutdown, /sbin/timedatectl

4 # User privilege specification
5 pi  ALL=(ALL) NOPASSWD: VLP

```

## 6.3 IP-Adressen-Konfiguration

Per Ethernet soll der Raspberry auf die IP-Adresse 192.168.1.111 konfiguriert werden, da diese IP-Adresse im Laserscanner als Host eingestellt war und an diesen die Daten vom Scanner übertragen werden. Die IP-Adresse des Raspberry Pi im WLAN wurde fest auf die gut zu merkende Adresse 10.10.10.10 geändert, hierüber erfolgt später der Zugriff auf die Weboberfläche (siehe auch Tabelle 6.1).

Die Konfiguration der IP-Adressen für den Raspberry erfolgt in der Konfigurationsdatei `/etc/network/interfaces` (siehe Listing 6.2. Raspberry Pi Foundation (2017)

Original  
hinzufügen

Listing 6.2: Konfiguration der /etc/network/interfaces

```
1 # localhost
2 auto lo
3 iface lo inet loopback

5 # Ethernet
6 auto eth0
7 iface eth0 inet static
8   address 192.168.1.110
9   netmask 255.255.255.0
10  gateway 192.168.1.110

12 # WLAN
13 allow-hotplug wlan0
14 iface wlan0 inet static
15   address 10.10.10.10
16   netmask 255.255.255.0
17   network 10.10.10.0
```

Zur Konfiguration der dynamischen IP-Adressen der Clients im WLAN wird ein DHCP-Server eingerichtet. Ein solcher Server weißt neuen Geräten – beziehungsweise welchen, die länger nicht im Netzwerk waren – automatisch eine neue, unverwendete IP-Adresse zu. Hierdurch benötigen die Clients keine spezielle Konfiguration und ihre IP-Einstellungen können auf dem üblichen Standardeinstellungen verbleiben (automatische IP-Adresse beziehen). Als DHCP-Server wird hier das Paket `dnsmasq` verwendet. Außer dem DHCP-Server bietet dieses Paket auch einen DNS-Server, der es erlaubt, den Geräten auch einen Hostname zuzuweisen. So wäre der Zugriff zum Beispiel über den Hostname `raspberry.ip` anstatt durch Eingabe der IP-Adresse möglich.

Die Konfiguration des DHCP-Servers ist vergleichsweise einfach und benötigt nur das verwendete Netzwerk-Interface, hier `wlan0`, den zu nutzenden IP-Bereich, die Netzmarske und die Zeit, nach der eine IP-Adresse an ein anderes Gerät vergeben werden darf, die sogenannte Lease-Time (siehe Listing 6.3). Raspberry Pi Foundation (2017)

Listing 6.3: Konfiguration der /etc/dnsmasq.conf

```
1 interface=wlan0
2 dhcp-range=10.10.10.100,10.10.10.254,255.255.255.0,24h
```

## 6.4 Konfiguration als WLAN-Access-Point

Um einen Zugriff auf die Python-Weboberfläche des Skriptes und die Konfiguration des Laserscanners zu ermöglichen, soll der Raspberry Pi selbst als WLAN-Access-Point fungieren. Hierzu wurde das Paket `hostapd` verwendet. Zur Konfiguration werden die Einstellungen in die Datei `/etc/hostapd/hostapd.conf` geschrieben. Raspberry Pi Foundation (2017)

Listing 6.4: Konfiguration der /etc/hostapd/hostapd.conf

```
1 # WLAN-Router-Betrieb
```

```

3 # Schnittstelle und Treiber
4 interface=wlan0
5 #driver=nl80211

7 # WLAN-Konfiguration
8 ssid=VLPinterface
9 channel=1
10 hw_mode=g
11 ieee80211n=1
12 ieee80211d=1
13 country_code=DE
14 wmm_enabled=1

16 #WLAN-Verschluesselung
17 auth_algs=1
18 wpa=2
19 wpa_key_mgmt=WPA-PSK
20 rsn_pairwise=CCMP
21 wpa_passphrase=raspberry

```

## 6.5 Autostart des Skriptes

Damit das Skript vor der Messung mittels SSH-Zugang gestartet werden muss, wurde das Skript in den Autostart des Raspberry Pi eingetragen. Hierdurch erfolgt der Start des Skriptes unmittelbar nach dem Hochfahren des Betriebssystems.

Listing 6.5: Startskript startVLP.sh

```

1 su pi -c "python3 VdAutoStart.py"
2 exit 0

```

Der Pfad zur dem Startskript wurde dann in der Autostart-Konfigurationsdatei `/etc/rc.local` eingetragen.

zu  
Ende  
schrei-  
ben

# **7 Systemüberprüfungen**

Nachdem bei der Systemkonfiguration bisher auf die Angaben aus Handbüchern und Anleitungen vertraut wurde, sollte zusätzlich die Genauigkeit von einigen Systemkomponenten überprüft werden. Hierfür wurde einmal die Messgenauigkeit des Scanners ausgewählt sowie die Genauigkeit der Zeitangaben der GNSS-Systeme.

## **7.1 Untersuchung der Gleichzeitigkeit von PPS-Signalen von verschiedenen GNSS-Empfängern**

Für die Synchronisierung der Daten des Laserscanners und der inertialen Messeinheit wurden zwei verschiedene GNSS-Empfänger verwendet. Der für den Einbau in Consumer-Geräte gedachte uBlox-Chip, der am Raspberry Pi und am Laserscanner verwendet wird, ist leichter, unabhängiger und einfacher zu realisieren als die Übernahme der Daten mittels Adapterkabeln von der inertialen Messeinheit. Voraussetzung hierfür ist jedoch, dass beide Signale wirklich gleichzeitig erzeugt werden. Um dies zu überprüfen, soll das PPS-Signal (Impulssignal im Sekudentakt) von beiden Messsystemen mit einem Arduino überprüft werden.

## **7.2 Messgenauigkeit des Laserscanners im Vergleich**

## **8 Ausblick**

# Literaturverzeichnis

Bachfeld, Daniel (März 2013): Quadrokopter-Know-how. *c't Hacks*, (3).

Beraldin, J.-Angelo; Blais, François; Lohr, Uwe (2010): Laser Scanning Technology. In: Vosselman, George; Maas, Hans-Gerd (Hg.), Airborne and terrestrial laser scanning, S. 1 – 42, Whittles Publishing, Dunbeath, Vereinigtes Königreich, ISBN 978-1-4398-2798-7.

Ehring, Ehling; Klingbeil, Lasse; Kuhlmann, Heiner (2016): Warum UAVs und warum jetzt? In: Ehring, Ehling; Klingbeil, Lasse (Hg.), UAV 2016 – Vermessung mit unbemannten Flugsystemen, Band 82/2016, S. 9–30, DVW - Gesellschaft für Geodäsie, Geoinformation und Landmanagement e.V., ISBN 978-3-95786-067-5.

Heise Online (2017): Quadrocopter - Drohnen & Multikopter. <https://www.heise.de/thema/Quadrocopter>. (Aufruf: 27. Sep. 2017).

iMAR Navigation GmbH (2015): iNAT-M200-FLAT.

Möcker, Andrijan (28. Feb. 2017): 5 Jahre Raspberry Pi: Wie ein Platinchen die Welt eroberte. Heise Online, <https://heise.de/-3636046>. (Aufruf: 21. Sep. 2017).

Pack, Robert T.; Brooks, Valerie; Young, Jamie; Vilaça, Nuno; Vatslid, Svein; Rindle, Peter; Kurz, Sven; Parrish, Christopher E.; Craig, Rex; Smith, Philip W. (2012): An overview of ALS technology. In: Renslow, Michael S. (Hg.), Manual of airborne topographic lidar, S. 7 – 97, American Society for Photogrammetry and Remote Sensing, ISBN 1-570-83097-5.

Raspberry Pi Foundation (2017): AccessPoint. <https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md>. (Aufruf: 25. Okt. 2017).

RS Components Limited (2015): Raspberry Pi 3 Model B Datasheet. <http://docs-europe.electrocomponents.com/webdocs/14ba/0900766b814ba5fd.pdf>. (Aufruf: 21. Sep. 2017).

Schnabel, Patrick (2017): Raspberry Pi: Belegung GPIO (Banana Pi und WiringPi). Elektronik Kompendium, <https://www.elektronik-kompendium.de/sites/raspberry-pi/1907101.htm>. (Aufruf: 21. Sep. 2017).

Schulz, Jasper (2016): Aufbau und Betrieb eines Zeilenlaserscanners an einem Multikopter. <http://edoc.sub.uni-hamburg.de/hcu/volltexte/campus/2016/259/>. (Aufruf: 30. Sep. 2017), (unveröffentlicht).

Theis, Thomas (2011): Einstieg in Python. 3. Auflage, Galileo Press, Bonn.

ubuntuusers.de (2017): Herunterfahren. [https://wiki.ubuntuusers.de/Herunterfahren/](https://wiki.ubuntuusers.de/Herunterfahren). (Aufruf: 22. Okt. 2017).

Velodyne Lidar (2014): VLP-16 Envelope Drawing (2D). <http://velodynelidar.com/docs/drawings/86-0101%20REV%20B1%20ENVELOPE,VLP-16.pdf>. (Aufruf: 25. Sept. 2017).

Velodyne Lidar (2016): VLP-16 User's Manual and Programming Guide.

Velodyne Lidar (2017a): HDL-32E & VLP-16 Interface Box. [http://velodynelidar.com/docs/notes/63-9259%20REV%20C%20MANUAL,INTERFACE%20BOX,HDL-32E,VLP-16,VLP-32\\_Web-S.pdf](http://velodynelidar.com/docs/notes/63-9259%20REV%20C%20MANUAL,INTERFACE%20BOX,HDL-32E,VLP-16,VLP-32_Web-S.pdf). (Aufruf: 22. Okt. 2017).

Velodyne Lidar (2017b): VLP-16 Data Sheet.

Wilken, Mathias (2017): Untersuchung der RTK-Performance des INS/GNSS iNAT M200 Systems. (unveröffentlicht).

Witte, Bertold; Schmidt, Hubert (2006): Vermessungskunde und Grundlagen der Statistik für das Bauwesen. 6. Auflage, Herbert Wichmann Verlag, Heidelberg, ISBN 978-3-879-07435-8.

# Abbildungsverzeichnis

2.1	MPU-9250 - Low-Cost-MEMS-IMU-Modul wie es in vielen Consumer-Geräten und Multikoptern verwendet wird (schwarzes Bauteil mittig auf der Platine, eigene Aufnahme) . . . . .	8
3.1	Laserscanner Velodyne VLP-16 (eigene Aufnahme) . . . . .	11
3.2	iMAR iNAT-M200-Flat im Prototypen des modularen Gehäuses, Leitungen führen zu den GNSS-Antennen (eigene Aufnahme) . . . . .	12
3.3	GNSS-Antennen des (links und rechts) iMAR iNAT-M200-Flat an Prototypen des modularen Gehäuses (eigene Aufnahme) . . . . .	12
3.4	Raspberry Pi 3 (eigene Aufnahme) . . . . .	14
3.5	Multikopter Copterproject CineStar 6HL mit Gimbal Freefly MöVI M5 (eigene Aufnahme) . . . . .	15
3.6	uBlox NEO-M8N, das Vorgängermodell NEO-6M mit PPS-Ausgang wurde verwendet (eigene Aufnahme) . . . . .	17
3.7	Messung des Signals am uBlox NEO-6M (grün: Ausgangssignal; rot: Signal nach Nutzung eines Pegelwandler; 1000 Punkte entsprechen 5 Volt)	18
3.8	Entwurf des Schaltplanes zum Anschluss des GNSS-Modules an den Laserscanner, gezeichnet in Fritzing . . . . .	19
3.9	Entwurf des Schaltplanes für Steuerung des Raspberry, gezeichnet in Fritzing . . . . .	19
3.10	Layout der Lochstreifenplatine . . . . .	21
3.11	Vereinfachter, endgültiger Schaltplan . . . . .	22
4.1	Strahlengang im Laserscanner VLP-16, Werte in Millimetern, nach Velodyne Lidar (2014) . . . . .	26
4.2	Vereinfachter Ablaufplan des Skriptes . . . . .	28
5.1	UML-Klassendiagramm . . . . .	30

# **Tabellenverzeichnis**

3.1	Spannungs- und Strombedarf der einzelnen Module (Velodyne Lidar, 2017b; iMAR Navigation GmbH, 2015; RS Components Limited, 2015)	16
4.1	Aufbau der Daten des Netzwerkpaketes, nach Velodyne Lidar (2016) . . .	24
6.1	IP-Adressen-Verteilung . . . . .	37

# **Anhang**

# A Python-Skripte

## A.1 vdAutoStart.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-

4  """
5  @author: Florian Timm
6  @version: 2017.11.18
7  """

9  import configparser
10 import multiprocessing
11 import os
12 import signal
13 import sys
14 import time
15 from datetime import datetime
16 from multiprocessing import Queue, Manager
17 from threading import Thread
18 from flask import Flask
19 from vdBuffer import VdBuffer
20 from vdTransformer import VdTransformer
21 from vdGNSSTime import VdGNSSTime

24 class VdAutoStart(object):

26     """ main script for automatic start """

28     def __init__(self, web_interface):
29         """
30             Constructor
31             :param web_interface: Thread with Flask web interface
32             :type web_interface: Thread
33         """
34         self.__vd_hardware = None
35         print("Data Interface for VLP-16\n")

37         # load config file
38         self.__conf = configparser.ConfigParser()
39         self.__conf.read("config.ini")

41         # variables for child processes
42         self.__pBuffer = None
43         self.__pTransformer = None

45         # pipes for child processes
```

```

46     manager = Manager()
47     self.__gnss_status = "(unknown)"
48     # self.__gnssReady = manager.Value('gnssReady',False)
49     self.__go_on_buffer = manager.Value('_go_on_buffer', False)
50     self.__go_on_transform = manager.Value('_go_on_transform', False)
51     self.__scanner_status = manager.Value('__scanner_status', "(unknown)")
52     self.__dataset_cnt = manager.Value('__dataset_cnt', 0)
53     self.__date = manager.Value('__date', None)

55     # queue for transformer
56     self.__queue = Queue()

58     # attribute for web interface
59     self.__web_interface = web_interface

61     # check admin
62     try:
63         os.rename('/etc/foo', '/etc/bar')
64         self.__admin = True
65     except IOError:
66         self.__admin = False

68     # check raspberry pi
69     try:
70         import RPi.GPIO
71         self.__raspberry = True
72     except ModuleNotFoundError:
73         self.__raspberry = False

75     self.__gnss = None

77     def run(self):
78         """ start script """
79

80         # handle SIGINT
81         signal.signal(signal.SIGINT, self.__signal_handler)

83         # use hardware control on raspberry pi
84         if self.__raspberry:
85             print("Raspberry Pi was detected")
86             from vdHardware import VdHardware
87             self.__vd.hardware = VdHardware(self)
88             self.__vd.hardware.start()
89         else:
90             print("Raspberry Pi could not be detected")
91             print("Hardware control deactivated")

93         # set time by using gnss
94         if self.__conf.get("Funktionen", "GNSSZeitVerwenden"):
95             self.__gnss = VdGNSSTime(self)
96             self.__gnss.start()

98     def start_transformer(self):
99         """ Starts transformer processes"""
100        print("Start transformer...")
101        # number of transformer according number of processor cores
102        if self.__conf.get("Funktionen", "activateTransformer"):
103            self.__go_on_transform.value = True
104            n = multiprocessing.cpu_count() - 1

```

```

105         if n < 2:
106             n = 1
107             self.__pTransformer = []
108             for i in range(n):
109                 t = VdTransformer(i, self)
110                 t.start()
111                 self.__pTransformer.append(t)

113             print(str(n) + " transformer started!")

115     def start_recording(self):
116         """ Starts recording process """
117         if not (self.__go_on_buffer.value and self.__pBuffer.is_alive()):
118             self.__go_on_buffer.value = True
119             print("Recording is starting...")
120             self.__scanner_status.value = "recording started"
121             # buffering process
122             self.__pBuffer = VdBuffer(self)
123             self.__pBuffer.start()
124             if self.__pTransformer is None:
125                 self.start_transformer()

127     def stop_recording(self):
128         print("Recording is stopping... (10 seconds timeout before kill)")
129         self.__go_on_buffer.value = False
130         self.__date.value = None
131         if self.__pBuffer is not None:
132             self.__pBuffer.join(10)
133             if self.__pBuffer.is_alive():
134                 print("Could not stop process, it will be killed!")
135                 self.__pBuffer.terminate()
136                 print("Recording terminated!")
137             else:
138                 print("Recording was not started!")

140     def stop_transformer(self):
141         """ Stops transformer processes """
142         print("Transformer is stopping... (10 seconds timeout before kill)")
143         self.__go_on_transform.value = False
144         if self.__pTransformer is not None:
145             for pT in self.__pTransformer:
146                 pT.join(15)
147                 if pT.is_alive():
148                     print(
149                         "Could not stop process, it will be killed!")
150                     pT.terminate()
151                     print("Transformer terminated!")
152                 else:
153                     print("Transformer was not started!")

155     def stop_web_interface(self):
156         """ Stop web interface -- not implemented now"""
157         # Todo
158         # self.__web_interface.exit()
159         # print("Web interface stopped!")
160         pass

162     def stop_hardware_control(self):
163         """ Stop hardware control """

```

```

164         if self.__vd_hardware is not None:
165             self.__vd_hardware.stop()
166             self.__vd_hardware.join(5)
167
168     def stop_children(self):
169         """ Stop child processes and threads """
170         print("Script is stopping...")
171         self.stop_recording()
172         self.stop_transformer()
173         self.stop_web_interface()
174         self.stop.hardware_control()
175         print("Child processes stopped")
176
177     def end(self):
178         """ Stop script complete """
179         self.stop_children()
180         sys.exit()
181
182     def __signal_handler(self, sig_no, frame):
183         """
184             handles SIGINT-signal
185             :param sig_no: signal number
186             :type sig_no: int
187             :param frame: execution frame
188             :type frame: frame
189         """
190         del sig_no, frame
191         print('Ctrl+C pressed!')
192         self.stop_children()
193         sys.exit()
194
195     def shutdown(self):
196         """ Shutdown Raspberry Pi """
197         self.stop_children()
198         os.system("sleep 5s; sudo shutdown -h now")
199         print("Shutdown Raspberry...")
200         sys.exit(0)
201
202     def check_queue(self):
203         """ Check, whether queue is filled """
204         if self.__queue.qsize() > 0:
205             return True
206         return False
207
208     def check_recording(self):
209         """ Check data recording by pBuffer """
210         if self.__pBuffer is not None and self.__pBuffer.is_alive():
211             return True
212         return False
213
214     def check_receiving(self):
215         """ Check data receiving """
216         x = self.__dataset_cnt.value
217         time.sleep(0.2)
218         y = self.__dataset_cnt.value
219         if x - y > 0:
220             return True
221         return False

```

```

223     # getter/setter methods
224     def __get_conf(self):
225         """
226             Gets config file
227             :return: config file
228             :rtype: configparser.ConfigParser
229             """
230             return self.__conf
231     conf = property(__get_conf)
232
233     def __get_gnss_status(self):
234         """
235             Gets GNSS status
236             :return: GNSS status
237             :rtype: Manager
238             """
239             return self.__gnss_status
240
241     def __set_gnss_status(self, gnss_status):
242         """
243             Sets GNSS status
244             :param gnss_status: gnss status
245             :type gnss_status: str
246             """
247             self.__gnss_status = gnss_status
248     gnss_status = property(__get_gnss_status, __set_gnss_status)
249
250     def __get_go_on_buffer(self):
251         """
252             Should Buffer buffer data?
253             :return: go on buffering
254             :rtype: Manager
255             """
256             return self.__go_on_buffer
257     go_on_buffer = property(__get_go_on_buffer)
258
259     def __get_go_on_transform(self):
260         """
261             Should Transformer transform data?
262             :return: go on transforming
263             :rtype: Manager
264             """
265             return self.__go_on_transform
266     go_on_transform = property(__get_go_on_transform)
267
268     def __get_scanner_status(self):
269         """
270             Gets scanner status
271             :return:
272             :rtype: Manager
273             """
274             return self.__scanner_status
275     scanner_status = property(__get_scanner_status)
276
277     def __get_dataset_cnt(self):
278         """
279             Gets number of buffered datasets
280             :return: number of buffered datasets
281             :rtype: Manager

```

```

282     """
283     return self._dataset_cnt
284 dataset_cnt = property(_get_dataset_cnt)

286     def _get_date(self):
287         """
288             Gets recording start time
289             :return: timestamp starting recording
290             :rtype: Manager
291         """
292         return self._date

294     def _set_date(self, date):
295         """
296             Sets recording start time
297             :param date: timestamp starting recording
298             :type date: datetime
299         """
300         self._date = date
301         #: recording start time
302 date = property(_get_date, _set_date)

304     def _is_admin(self):
305         """
306             Admin?
307             :return: Admin?
308             :rtype: bool
309         """
310         return self._admin
311 admin = property(_is_admin)

313     def _get_queue(self):
314         """
315             Gets transformer queue
316             :return: transformer queue
317             :rtype: Queue
318         """
319         return self._queue
320 queue = property(_get_queue)

323 # Websteuerung
324 app = Flask(__name__)

327 @app.route("/")
328 def web_index():
329     runtime = "(inactive)"
330     pps = "(inactive)"
331     if ms.date.value is not None:
332         timediff = datetime.now() - ms.date.value
333         td_sec = timediff.seconds + (int(timediff.microseconds / 1000) / 1000.)
334         sec = td_sec % 60
335         minu = int((td_sec // 60) % 60)
336         h = int(td_sec // 3600)

338         runtime = '{:02d}:{:02d}:{:06.3f}'.format(h, minu, sec)

340         pps = '{:.0f}'.format(ms.dataset_cnt.value / td_sec)

```

```

342     elif ms.go_on_buffer.value:
343         runtime = "(no data)"

345     ausgabe = """<html>
346     <head>
347         <title>VLP16-Data-Interface</title>
348         <meta name="viewport" content="width=device-width; initial-scale=1.0;" />
349         <link href="/style.css" rel="stylesheet">
350         <meta http-equiv="refresh" content="5; URL=/>
351     </head>
352     <body>
353     <content>
354         <h2>VLP16-Data-Interface</h2>
355         <table style="">
356             <tr><td id="spalte1">GNSS-status:</td><td>"""+ ms.gnss_status + """</td></tr>
357             <tr><td>Scanner:</td><td>"""+ ms.scanner_status.value + """</td></tr>
358             <tr><td>Datasets</td>
359                 <td>"""+ str(ms.dataset_cnt.value) + """</td></tr>
360             <tr><td>Queue:</td>
361                 <td>"""+ str(ms.queue.qsize()) + """</td></tr>
362             <tr><td>Recording time:</td>
363                 <td>"""+ runtime + """</td>
364             </tr>
365             <tr><td>Points/seconds:</td>
366                 <td>"""+ pps + """</td>
367             </tr>
368         </table><br />
369         """
370     if ms.check_recording():
371         ausgabe += """<a href="/stoppen" id="stoppen">
372             Stop recording</a><br />"""
373     else:
374         ausgabe += """<a href="/starten" id="starten">
375             Start recording</a><br />"""
376     ausgabe += """
377         <a href="/beenden" id="beenden">Terminate script<br />
378         (control by SSH available only)</a></td></tr><br />
379         <a href="/shutdown" id="shutdown">Shutdown Raspberry Pi</a>
380     </content>
381     </body>
382 </html>"""

384     return ausgabe

387 @app.route("/style.css")
388 def css_style():
389     return """
390     body, html, content {
391         text-align: center;
392     }

394     content {
395         max-width: 15cm;
396         display: block;
397         margin: auto;
398     }

```

```

400     table {
401         border-collapse: collapse;
402         width: 90%;
403         margin: auto;
404     }
405
406     td {
407         border: 1px solid black;
408         padding: 1px 2px;
409     }
410
411     td#spalte1 {
412         width: 30%;
413     }
414
415     a {
416         display: block;
417         width: 90%;
418         padding: 0.5em 0;
419         text-align: center;
420         margin: auto;
421         color: #fff;
422     }
423
424     a#stoppen {
425         background-color: #e90;
426     }
427
428     a#shutdown {
429         background-color: #b00;
430     }
431
432     a#starten {
433         background-color: #1a1;
434     }
435
436     a#beenden {
437         background-color: #f44;
438     }
439     """
440
441
442     @app.route("/shutdown")
443     def web_shutdown():
444         ms.shutdown()
445         return """
446             <meta http-equiv="refresh" content="3; URL=/">
447             Wird in 10 Sekunden heruntergefahren..."""
448
449
450     @app.route("/beenden")
451     def web_exit():
452         ms.end()
453         return """
454             <meta http-equiv="refresh" content="3; URL=/">
455             Terminating..."""

```

```

458 @app.route("/stoppen")
459 def web_stop():
460     ms.stop_recording()
461     return """
462         <meta http-equiv="refresh" content="3; URL=/">
463         Recording is stopping..."""
464
465
466 @app.route("/starten")
467 def web_start():
468     ms.start_recording()
469     return """
470         <meta http-equiv="refresh" content="3; URL=/">
471         Recording is starting..."""
472
473
474 def start_web():
475     print("Webserver is starting...")
476     app.run('0.0.0.0', 8080)
477
478
479 if __name__ == '__main__':
480     w = Thread(target=start_web)
481     ms = VdAutoStart(w)
482     w.start()
483     ms.run()

```

## A.2 vdBuffer.py

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """
5  @author: Florian Timm
6  @version: 2017.11.17
7  """
8
9  from vdInterface import VdInterface
10 import socket
11 from multiprocessing import Process
12 import os
13 from datetime import datetime
14 import signal
15
16
17 class VdBuffer(Process):
18
19     """ process for buffering binary data """
20
21     def __init__(self, master):
22         """
23             Constructor
24             :param master: instance of VdAutoStart
25             :type master: VdAutoStart
26         """
27         # Konstruktor der Elternklasse
28         Process.__init__(self)

```

```

30         # Pipes sichern
31         # self.__master = master
32         self.__go_on_buffering = master.go_on_buffer
33         self.__scanner_status = master.scanner_status
34         self.__datasets = master.dataset_cnt
35         self.__queue = master.queue
36         self.__admin = master.admin
37         self.__date = master.date
38         self.__conf = master.conf

39         self.__file_no = 0

40
41     @staticmethod
42     def __signal_handler(sig_no, frame):
43         """
44             handles SIGINT-signal
45             :param sig_no: signal number
46             :type sig_no: int
47             :param frame: execution frame
48             :type frame: frame
49         """
50
51         del sig_no, frame
52         # self.master.end()
53         print("SIGINT vdBuffer")

54
55     def __new_folder(self):
56         """ creates data folder """
57         # Uhrzeit abfragen fuer Laufzeitlaenge und Dateinamen
58         self.__date.value = datetime.now()
59         self.__folder = self.__conf.get("Datei", "fileNamePre")
60         self.__folder += self.__date.value.strftime(
61             self.__conf.get("Datei", "fileTimeFormat"))
62         # Speicherordner anlegen und ausgeben
63         os.makedirs(self.__folder)
64         print("Data folder: " + self.__folder)

65
66     def run(self):
67         """ starts buffering process """
68         signal.signal(signal.SIGINT, self.__signal_handler)

69
70         # Socket zum Scanner oeffnen
71         sock = VdInterface.get_data_stream(self.__conf)
72         self.__scanner_status.value = "Socket verbunden"

73
74         # Variablen initialisieren
75         minibuffer = b''
76         j = 0

77         self.__datasets.value = 0

78
79         # Prozessprioritaet hochschalten, sofern Adminrechte
80         if self.__admin:
81             os.nice(-18)

82         transformer = self.__conf.get("Funktionen", "activateTransformer")
83         measurements_per_dataset = int(self.__conf.get(
84             "Geraet", "messungProDatensatz"))

85
86         # Dauerschleife, solange kein Unterbrechen-Befehl kommt

```

```

90         sock.settimeout(1)
91         while self.__go_on_buffering.value:
92             try:
93                 # Daten vom Scanner holen
94                 data = sock.recvfrom(1248)[0]
95
96                 if j == 0 and self.__file_no == 0:
97                     self.__new_folder()
98                     # RAM-Buffer
99                     minibuffer += data
100                j += 1
101                self.__datasets.value += measurements_per_dataset
102                # Alle 5 bzw. 10 Sekunden Daten speichern
103                # oder wenn Abbrechenbefehl kommt
104                if (j >= 1500) or (not self.__go_on_buffering.value):
105                    # Datei schreiben
106                    f = open(
107                        self.__folder + "/" + str(self.__file_no) + ".bin",
108                        "wb")
109                    f.write(minibuffer)
110
111                    f.close()
112
113                    if transformer:
114                        self.__queue.put(f.name)
115
116                    # Buffer leeren
117                    minibuffer = b''
118                    j = 0
119
120                    # Dateizähler
121                    self.__file_no += 1
122
123                    if data == 'QUIT':
124                        break
125                    except socket.timeout:
126                        print("No data")
127                        continue
128                    sock.close()
129                    self.__scanner_status.value = "recording stopped"
130                    print("Disconnected!")

```

## A.3 vdTransformer.py

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """
5  @author: Florian Timm
6  @version: 2017.11.17
7  """
8  import os
9  import signal
10 from multiprocessing import Process
11 from queue import Empty
12 from vdFile import VdTxtFile
13 from vdDataset import VdDataset

```

```

16 class VdTransformer(Process):
17
18     """
19         Erzeugt einen Prozess zum Umwandeln von binaeren Daten des
20         Velodyne VLP-16 zu TXT-Dateien
21     """
22
23     def __init__(self, nummer, master):
24         """
25             Konstruktor fuer Transformer-Prozess, erbt von multiprocessing.Process
26
27             Parameters
28             -----
29             nummer : int
30                 Nummer des Transformerprozess
31             master : VdAutoStart
32                 Objekt des Hauptskriptes
33         """
34
35
36         # Konstruktor der Elternklasse
37         Process.__init__(self)
38
39         # Parameter sichern
40         # self.__master = master
41         self.__queue = master.queue
42         self.__number = nummer
43         self.__admin = master.admin
44         self.__go_on_transform = master.go_on_transform
45         self.__conf = master.conf
46
47     @staticmethod
48     def __signal_handler(sig_no, frame):
49         """
50             handles SIGINT-signal
51             :param sig_no: signal number
52             :type sig_no: int
53             :param frame: execution frame
54             :type frame: frame
55         """
56
57         del sig_no, frame
58         # self.master.end()
59         print("SIGINT vdTransformer")
60
61     def run(self):
62         """
63             starts transforming process """
64         signal.signal(signal.SIGINT, self.__signal_handler)
65
66         if self.__admin:
67             os.nice(-15)
68         # Erzeugen einer TXT-Datei pro Prozess
69
70         old_folder = ""
71         # Dauerschleife
72
73         try:
74             while self.__go_on_transform.value:
75                 try:
76
77                     # Transforming
78
79                     # Write to file
80
81                     # Read from queue
82
83                     # Process data
84
85                     # Write to file
86
87                     # Read from queue
88
89                     # Process data
90
91                     # Write to file
92
93                     # Read from queue
94
95                     # Process data
96
97                     # Write to file
98
99                     # Read from queue
100
101                     # Process data
102
103                     # Write to file
104
105                     # Read from queue
106
107                     # Process data
108
109                     # Write to file
110
111                     # Read from queue
112
113                     # Process data
114
115                     # Write to file
116
117                     # Read from queue
118
119                     # Process data
120
121                     # Write to file
122
123                     # Read from queue
124
125                     # Process data
126
127                     # Write to file
128
129                     # Read from queue
130
131                     # Process data
132
133                     # Write to file
134
135                     # Read from queue
136
137                     # Process data
138
139                     # Write to file
140
141                     # Read from queue
142
143                     # Process data
144
145                     # Write to file
146
147                     # Read from queue
148
149                     # Process data
150
151                     # Write to file
152
153                     # Read from queue
154
155                     # Process data
156
157                     # Write to file
158
159                     # Read from queue
160
161                     # Process data
162
163                     # Write to file
164
165                     # Read from queue
166
167                     # Process data
168
169                     # Write to file
170
171                     # Read from queue
172
173                     # Process data
174
175                     # Write to file
176
177                     # Read from queue
178
179                     # Process data
180
181                     # Write to file
182
183                     # Read from queue
184
185                     # Process data
186
187                     # Write to file
188
189                     # Read from queue
190
191                     # Process data
192
193                     # Write to file
194
195                     # Read from queue
196
197                     # Process data
198
199                     # Write to file
200
201                     # Read from queue
202
203                     # Process data
204
205                     # Write to file
206
207                     # Read from queue
208
209                     # Process data
210
211                     # Write to file
212
213                     # Read from queue
214
215                     # Process data
216
217                     # Write to file
218
219                     # Read from queue
220
221                     # Process data
222
223                     # Write to file
224
225                     # Read from queue
226
227                     # Process data
228
229                     # Write to file
230
231                     # Read from queue
232
233                     # Process data
234
235                     # Write to file
236
237                     # Read from queue
238
239                     # Process data
240
241                     # Write to file
242
243                     # Read from queue
244
245                     # Process data
246
247                     # Write to file
248
249                     # Read from queue
250
251                     # Process data
252
253                     # Write to file
254
255                     # Read from queue
256
257                     # Process data
258
259                     # Write to file
260
261                     # Read from queue
262
263                     # Process data
264
265                     # Write to file
266
267                     # Read from queue
268
269                     # Process data
270
271                     # Write to file
272
273                     # Read from queue
274
275                     # Process data
276
277                     # Write to file
278
279                     # Read from queue
280
281                     # Process data
282
283                     # Write to file
284
285                     # Read from queue
286
287                     # Process data
288
289                     # Write to file
290
291                     # Read from queue
292
293                     # Process data
294
295                     # Write to file
296
297                     # Read from queue
298
299                     # Process data
299
300                     # Write to file
301
302                     # Read from queue
303
304                     # Process data
305
306                     # Write to file
307
308                     # Read from queue
309
310                     # Process data
311
312                     # Write to file
313
314                     # Read from queue
315
316                     # Process data
317
318                     # Write to file
319
320                     # Read from queue
321
322                     # Process data
323
324                     # Write to file
325
326                     # Read from queue
327
328                     # Process data
329
330                     # Write to file
331
332                     # Read from queue
333
334                     # Process data
335
336                     # Write to file
337
338                     # Read from queue
339
340                     # Process data
341
342                     # Write to file
343
344                     # Read from queue
345
346                     # Process data
347
348                     # Write to file
349
350                     # Read from queue
351
352                     # Process data
353
354                     # Write to file
355
356                     # Read from queue
357
358                     # Process data
359
360                     # Write to file
361
362                     # Read from queue
363
364                     # Process data
365
366                     # Write to file
367
368                     # Read from queue
369
370                     # Process data
371
372                     # Write to file
373
374                     # Read from queue
375
376                     # Process data
377
378                     # Write to file
379
380                     # Read from queue
381
382                     # Process data
383
384                     # Write to file
385
386                     # Read from queue
387
388                     # Process data
389
390                     # Write to file
391
392                     # Read from queue
393
394                     # Process data
395
396                     # Write to file
397
398                     # Read from queue
399
399
400                     # Process data
401
402                     # Write to file
403
404                     # Read from queue
405
406                     # Process data
407
408                     # Write to file
409
410                     # Read from queue
411
412                     # Process data
413
414                     # Write to file
415
416                     # Read from queue
417
418                     # Process data
419
420                     # Write to file
421
422                     # Read from queue
423
424                     # Process data
425
426                     # Write to file
427
428                     # Read from queue
429
430                     # Process data
431
432                     # Write to file
433
434                     # Read from queue
435
436                     # Process data
437
438                     # Write to file
439
440                     # Read from queue
441
442                     # Process data
443
444                     # Write to file
445
446                     # Read from queue
447
448                     # Process data
449
450                     # Write to file
451
452                     # Read from queue
453
454                     # Process data
455
456                     # Write to file
457
458                     # Read from queue
459
460                     # Process data
461
462                     # Write to file
463
464                     # Read from queue
465
466                     # Process data
467
468                     # Write to file
469
470                     # Read from queue
471
472                     # Process data
473
474                     # Write to file
475
476                     # Read from queue
477
478                     # Process data
479
480                     # Write to file
481
482                     # Read from queue
483
484                     # Process data
485
486                     # Write to file
487
488                     # Read from queue
489
490                     # Process data
491
492                     # Write to file
493
494                     # Read from queue
495
496                     # Process data
497
498                     # Write to file
499
500                     # Read from queue
501
502                     # Process data
503
504                     # Write to file
505
506                     # Read from queue
507
508                     # Process data
509
510                     # Write to file
511
512                     # Read from queue
513
514                     # Process data
515
516                     # Write to file
517
518                     # Read from queue
519
520                     # Process data
521
522                     # Write to file
523
524                     # Read from queue
525
526                     # Process data
527
528                     # Write to file
529
530                     # Read from queue
531
532                     # Process data
533
534                     # Write to file
535
536                     # Read from queue
537
538                     # Process data
539
540                     # Write to file
541
542                     # Read from queue
543
544                     # Process data
545
546                     # Write to file
547
548                     # Read from queue
549
550                     # Process data
551
552                     # Write to file
553
554                     # Read from queue
555
556                     # Process data
557
558                     # Write to file
559
560                     # Read from queue
561
562                     # Process data
563
564                     # Write to file
565
566                     # Read from queue
567
568                     # Process data
569
570                     # Write to file
571
572                     # Read from queue
573
574                     # Process data
575
576                     # Write to file
577
578                     # Read from queue
579
580                     # Process data
581
582                     # Write to file
583
584                     # Read from queue
585
586                     # Process data
587
588                     # Write to file
589
590                     # Read from queue
591
592                     # Process data
593
594                     # Write to file
595
596                     # Read from queue
597
598                     # Process data
599
599
600                     # Process data
601
602                     # Write to file
603
604                     # Read from queue
605
606                     # Process data
607
608                     # Write to file
609
610                     # Read from queue
611
612                     # Process data
613
614                     # Write to file
615
616                     # Read from queue
617
618                     # Process data
619
620                     # Write to file
621
622                     # Read from queue
623
624                     # Process data
625
626                     # Write to file
627
628                     # Read from queue
629
630                     # Process data
631
632                     # Write to file
633
634                     # Read from queue
635
636                     # Process data
637
638                     # Write to file
639
640                     # Read from queue
641
642                     # Process data
643
644                     # Write to file
645
646                     # Read from queue
647
648                     # Process data
649
650                     # Write to file
651
652                     # Read from queue
653
654                     # Process data
655
656                     # Write to file
657
658                     # Read from queue
659
660                     # Process data
661
662                     # Write to file
663
664                     # Read from queue
665
666                     # Process data
667
668                     # Write to file
669
670                     # Read from queue
671
672                     # Process data
673
674                     # Write to file
675
676                     # Read from queue
677
678                     # Process data
679
680                     # Write to file
681
682                     # Read from queue
683
684                     # Process data
685
686                     # Write to file
687
688                     # Read from queue
689
690                     # Process data
691
692                     # Write to file
693
694                     # Read from queue
695
696                     # Process data
697
698                     # Write to file
699
700                     # Read from queue
701
702                     # Process data
703
704                     # Write to file
705
706                     # Read from queue
707
708                     # Process data
709
710                     # Write to file
711
712                     # Read from queue
713
714                     # Process data
715
716                     # Write to file
717
718                     # Read from queue
719
720                     # Process data
721
722                     # Write to file
723
724                     # Read from queue
725
726                     # Process data
727
728                     # Write to file
729
730                     # Read from queue
731
732                     # Process data
733
734                     # Write to file
735
736                     # Read from queue
737
738                     # Process data
739
740                     # Write to file
741
742                     # Read from queue
743
744                     # Process data
745
746                     # Write to file
747
748                     # Read from queue
749
750                     # Process data
751
752                     # Write to file
753
754                     # Read from queue
755
756                     # Process data
757
758                     # Write to file
759
760                     # Read from queue
761
762                     # Process data
763
764                     # Write to file
765
766                     # Read from queue
767
768                     # Process data
769
770                     # Write to file
771
772                     # Read from queue
773
774                     # Process data
775
776                     # Write to file
777
778                     # Read from queue
779
780                     # Process data
781
782                     # Write to file
783
784                     # Read from queue
785
786                     # Process data
787
788                     # Write to file
789
790                     # Read from queue
791
792                     # Process data
793
794                     # Write to file
795
796                     # Read from queue
797
798                     # Process data
799
799
800                     # Process data
801
802                     # Write to file
803
804                     # Read from queue
805
806                     # Process data
807
808                     # Write to file
809
810                     # Read from queue
811
812                     # Process data
813
814                     # Write to file
815
816                     # Read from queue
817
818                     # Process data
819
820                     # Write to file
821
822                     # Read from queue
823
824                     # Process data
825
826                     # Write to file
827
828                     # Read from queue
829
830                     # Process data
831
832                     # Write to file
833
834                     # Read from queue
835
836                     # Process data
837
838                     # Write to file
839
840                     # Read from queue
841
842                     # Process data
843
844                     # Write to file
845
846                     # Read from queue
847
848                     # Process data
849
850                     # Write to file
851
852                     # Read from queue
853
854                     # Process data
855
856                     # Write to file
857
858                     # Read from queue
859
860                     # Process data
861
862                     # Write to file
863
864                     # Read from queue
865
866                     # Process data
867
868                     # Write to file
869
870                     # Read from queue
871
872                     # Process data
873
874                     # Write to file
875
876                     # Read from queue
877
878                     # Process data
879
880                     # Write to file
881
882                     # Read from queue
883
884                     # Process data
885
886                     # Write to file
887
888                     # Read from queue
889
890                     # Process data
891
892                     # Write to file
893
894                     # Read from queue
895
896                     # Process data
897
898                     # Write to file
899
900                     # Read from queue
901
902                     # Process data
903
904                     # Write to file
905
906                     # Read from queue
907
908                     # Process data
909
910                     # Write to file
911
912                     # Read from queue
913
914                     # Process data
915
916                     # Write to file
917
918                     # Read from queue
919
920                     # Process data
921
922                     # Write to file
923
924                     # Read from queue
925
926                     # Process data
927
928                     # Write to file
929
930                     # Read from queue
931
932                     # Process data
933
934                     # Write to file
935
936                     # Read from queue
937
938                     # Process data
939
940                     # Write to file
941
942                     # Read from queue
943
944                     # Process data
945
946                     # Write to file
947
948                     # Read from queue
949
950                     # Process data
951
952                     # Write to file
953
954                     # Read from queue
955
956                     # Process data
957
958                     # Write to file
959
960                     # Read from queue
961
962                     # Process data
963
964                     # Write to file
965
966                     # Read from queue
967
968                     # Process data
969
970                     # Write to file
971
972                     # Read from queue
973
974                     # Process data
975
976                     # Write to file
977
978                     # Read from queue
979
980                     # Process data
981
982                     # Write to file
983
984                     # Read from queue
985
986                     # Process data
987
988                     # Write to file
989
990                     # Read from queue
991
992                     # Process data
993
994                     # Write to file
995
996                     # Read from queue
997
998                     # Process data
999
1000                    # Write to file
1001
1002                    # Read from queue
1003
1004                    # Process data
1005
1006                    # Write to file
1007
1008                    # Read from queue
1009
1010                    # Process data
1011
1012                    # Write to file
1013
1014                    # Read from queue
1015
1016                    # Process data
1017
1018                    # Write to file
1019
1020                    # Read from queue
1021
1022                    # Process data
1023
1024                    # Write to file
1025
1026                    # Read from queue
1027
1028                    # Process data
1029
1030                    # Write to file
1031
1032                    # Read from queue
1033
1034                    # Process data
1035
1036                    # Write to file
1037
1038                    # Read from queue
1039
1040                    # Process data
1041
1042                    # Write to file
1043
1044                    # Read from queue
1045
1046                    # Process data
1047
1048                    # Write to file
1049
1050                    # Read from queue
1051
1052                    # Process data
1053
1054                    # Write to file
1055
1056                    # Read from queue
1057
1058                    # Process data
1059
1060                    # Write to file
1061
1062                    # Read from queue
1063
1064                    # Process data
1065
1066                    # Write to file
1067
1068                    # Read from queue
1069
1070                    # Process data
1071
1072                    # Write to file
1073
1074                    # Read from queue
1075
1076                    # Process data
1077
1078                    # Write to file
1079
1080                    # Read from queue
1081
1082                    # Process data
1083
1084                    # Write to file
1085
1086                    # Read from queue
1087
1088                    # Process data
1089
1090                    # Write to file
1091
1092                    # Read from queue
1093
1094                    # Process data
1095
1096                    # Write to file
1097
1098                    # Read from queue
1099
1099
1100                    # Process data
1101
1102                    # Write to file
1103
1104                    # Read from queue
1105
1106                    # Process data
1107
1108                    # Write to file
1109
1110                    # Read from queue
1111
1112                    # Process data
1113
1114                    # Write to file
1115
1116                    # Read from queue
1117
1118                    # Process data
1119
1120                    # Write to file
1121
1122                    # Read from queue
1123
1124                    # Process data
1125
1126                    # Write to file
1127
1128                    # Read from queue
1129
1130                    # Process data
1131
1132                    # Write to file
1133
1134                    # Read from queue
1135
1136                    # Process data
1137
1138                    # Write to file
1139
1140                    # Read from queue
1141
1142                    # Process data
1143
1144                    # Write to file
1145
1146                    # Read from queue
1147
1148                    # Process data
1149
1150                    # Write to file
1151
1152                    # Read from queue
1153
1154                    # Process data
1155
1156                    # Write to file
1157
1158                    # Read from queue
1159
1159
1160                    # Process data
1161
1162                    # Write to file
1163
1164                    # Read from queue
1165
1166                    # Process data
1167
1168                    # Write to file
1169
1170                    # Read from queue
1171
1172                    # Process data
1173
1174                    # Write to file
1175
1176                    # Read from queue
1177
1178                    # Process data
1179
1180                    # Write to file
1181
1182                    # Read from queue
1183
1184                    # Process data
1185
1186                    # Write to file
1187
1188                    # Read from queue
1189
1189
1190                    # Process data
1191
1192                    # Write to file
1193
1194                    # Read from queue
1195
1196                    # Process data
1197
1198                    # Write to file
1199
1200                    # Read from queue
1201
1202                    # Process data
1203
1204                    # Write to file
1205
1206                    # Read from queue
1207
1208                    # Process data
1209
1210                    # Write to file
1211
1212                    # Read from queue
1213
1214                    # Process data
1215
1216                    # Write to file
1217
1218                    # Read from queue
1219
1219
1220                    # Process data
1221
1222                    # Write to file
1223
1224                    # Read from queue
1225
1226                    # Process data
1227
1228                    # Write to file
1229
1230                    # Read from queue
1231
1232                    # Process data
1233
1234                    # Write to file
1235
1236                    # Read from queue
1237
1238                    # Process data
1239
1240                    # Write to file
1241
1242                    # Read from queue
1243
1244                    # Process data
1245
1246                    # Write to file
1247
1248                    # Read from queue
1249
1249
1250                    # Process data
1251
1252                    # Write to file
1253
1254                    # Read from queue
1255
1256                    # Process data
1257
1258                    # Write to file
1259
1260                    # Read from queue
1261
1262                    # Process data
1263
1264                    # Write to file
1265
1266                    # Read from queue
1267
1268                    # Process data
1269
1270                    # Write to file
1271
1272                    # Read from queue
1273
1274                    # Process data
1275
1276                    # Write to file
1277
1278                    # Read from queue
1279
1279
1280                    # Process data
1281
1282                    # Write to file
1283
1284                    # Read from queue
1285
1286                    # Process data
1287
1288                    # Write to file
1289
1290                    # Read from queue
1291
1292                    # Process data
1293
1294                    # Write to file
1295
1296                    # Read from queue
1297
1298                    # Process data
1299
1300                    # Write to file
1301
1302                    # Read from queue
1303
1304                    # Process data
1305
1306                    # Write to file
1307
1308                    # Read from queue
1309
1310                    # Process data
1311
1312                    # Write to file
1313
1314                    # Read from queue
1315
1316                    # Process data
1317
1318                    # Write to file
1319
1320                    # Read from queue
1321
1322                    # Process data
1323
1324                    # Write to file
1325
1326                    # Read from queue
1327
1328                    # Process data
1329
1330                    # Write to file
1331
1332                    # Read from queue
1333
1334                    # Process data
1335
1336                    # Write to file
1337
1338                    # Read from queue
1339
1339
1340                    # Process data
1341
1342                    # Write to file
1343
1344                    # Read from queue
1345
1346                    # Process data
1347
1348                    # Write to file
1349
1350                    # Read from queue
1351
1352                    # Process data
1353
1354                    # Write to file
1355
1356                    # Read from queue
1357
1358                    # Process data
1359
1360                    # Write to file
1361
1362                    # Read from queue
1363
1364                    # Process data
1365
1366                    # Write to file
1367
1368                    # Read from queue
1369
1369
1370                    # Process data
1371
1372                    # Write to file
1373
1374                    # Read from queue
1375
1376                    # Process data
1377
1378                    # Write to file
1379
1380                    # Read from queue
1381
1382                    # Process data
1383
1384                    # Write to file
1385
1386                    # Read from queue
1387
1388                    # Process data
1389
1390                    # Write to file
1391
1392                    # Read from queue
1393
1394                    # Process data
1395
1396                    # Write to file
1397
1398                    # Read from queue
1399
1399
1400                    # Process data
1401
1402                    # Write to file
1403
1404                    # Read from queue
1405
1406                    # Process data
1407
1408                    # Write to file
1409
1410                    # Read from queue
1411
1412                    # Process data
1413
1414                    # Write to file
1415
1416                    # Read from queue
1417
1418                    # Process data
1419
1420                    # Write to file
1421
1422                    # Read from queue
1423
1424                    # Process data
1425
1426                    # Write to file
1427
1428                    # Read from queue
1429
1429
1430                    # Process data
1431
1432                    # Write to file
1433
1434                    # Read from queue
1435
1436                    # Process data
1437
1438                    # Write to file
1439
1440                    # Read from queue
1441
1442                    # Process data
1443
1444                    # Write to file
1445
1446                    # Read from queue
1447
1448                    # Process data
1449
1450                    # Write to file
1451
1452                    # Read from queue
1453
1454                    # Process data
1455
1456                    # Write to file
1457
1458                    # Read from queue
1459
1459
1460                    # Process data
1461
1462                    # Write to file
1463
1464                    # Read from queue
1465
1466                    # Process data
1467
1468                    # Write to file
1469
1470                    # Read from queue
1471
1472                    # Process data
1473
1474                    # Write to file
1475
1476                    # Read from queue
1477
1478                    # Process data
1479
1480                    # Write to file
1481
1482                    # Read from queue
1483
1484                    # Process data
1485
1486                    # Write to file
1487
1488                    # Read from queue
1489
1489
1490                    # Process data
1491
1492                    # Write to file
1493
1494                    # Read from queue
1495
1496                    # Process data
1497
1498                    # Write to file
1499
1500                    # Read from queue
1501
1502                    # Process data
1503
1504                    # Write to file
1505
1506                    # Read from queue
1507
1508                    # Process data
1509
1510                    # Write to file
1511
1512                    # Read from queue
1513
1514                    # Process data
1515
1516                    # Write to file
1517
1518                    # Read from queue
1519
1519
1520                    # Process data
1521
1522                    # Write to file
1523
1524                    # Read from queue
1525
1526                    # Process data
1527
1528                    # Write to file
1529
1530                    # Read from queue
1531
1532                    # Process data
1533
1534                    # Write to file
1535
1536                    # Read from queue
1537
1538                    # Process data
1539
1540                    # Write to file
1541
1542                    # Read from queue
1543
1544                    # Process data
1545
1546                    # Write to file
1547
1548                    # Read from queue
1549
1549
1550                    # Process data
1551
1552                    # Write to file
1553
1554                    # Read from queue
1555
1556                    # Process data
1557
1558                    # Write to file
1559
1560                    # Read from queue
1561
1562                    # Process data
1563
1564                    # Write to file
1565
1566                    # Read from queue
1567
1568                    # Process data
1569
1570                    # Write to file
1571
1572                    # Read from queue
1573
1574                    # Process data
1575
1576                    # Write to file
1577
1578                    # Read from queue
1579
1579
1580                    # Process data
1581
1582                    # Write to file
1583
1584                    # Read from queue
1585
1586                    # Process data
1587
1588                    # Write to file
1589
1590                    # Read from queue
1591
1592                    # Process data
1593
1594                    # Write to file
1595
1596                    # Read from queue
1597
1598                    # Process data
1599
1599
1600                    # Write to file
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
19
```

```

73     # Dateinummer aus Warteschleife abfragen und öffnen
74     filename = self.__queue.get(True, 2)
75     folder = os.path.dirname(filename)
76     if dir != old_folder:
77         vd_file = VdTxtFile(
78             self.__conf,
79             folder + "/vd_file" + str(self.__number))
80     old_folder = folder

82     f = open(filename, "rb")

84     # Anzahl an Datensaetzen in Datei pruefen
85     file_size = os.path.getsize(f.name)
86     dataset_cnt = int(file_size / 1206)

88     for i in range(dataset_cnt):
89         # naechsten Datensatz lesen
90         vd_data = VdDataset(self.__conf, f.read(1206))

92         # Daten konvertieren und speichern
93         vd_data.convert_data()

95         # Datensatz zu Datei hinzufuegen
96         vd_file.add_dataset(vd_data)

98         # Datei schreiben
99         vd_file.write()
100        # Txt-Datei schliessen
101        f.close()
102        # Bin-Datei ggf. loeschen
103        if self.__conf.get("Datei", "binNachTransLoeschen"):
104            os.remove(f.name)
105    except Empty:
106        print("Warteschlange leer!")
107        continue
108    except BrokenPipeError:
109        print("vdTransformer-Pipe defekt")

```

## A.4 vdInterface.py

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-

4  """
5  @author: Florian Timm
6  @version: 2017.11.18
7  """

9  import socket
10 import sys

13 class VdInterface(object):

15     """ interface to velodyne scanner """

17     @staticmethod
18     def get_data_stream(conf):

```

```

19     """
20     Creates socket to scanner data stream
21     :param conf: configuration file
22     :type conf: configparser.ConfigParser
23     :return: socket to scanner
24     :rtype: socket.socket
25     """
26     return VdInterface.get_stream(conf.get("Netzwerk", "UDP_IP"),
27                                   int(conf.get("Netzwerk", "UDP_PORT_DATA")))
28
29 @staticmethod
30 def get_gnss_stream(conf):
31     """
32     Creates socket to scanner gnss stream
33     :param conf: configuration file
34     :type conf: configparser.ConfigParser
35     :return: socket to scanner
36     :rtype: socket.socket
37     """
38     return VdInterface.get_stream(conf.get("Netzwerk", "UDP_IP"),
39                               int(conf.get("Netzwerk", "UDP_PORT_GNSS")))
40
41 @staticmethod
42 def get_stream(ip, port):
43     """
44     Creates socket to scanner stream
45     :param ip: ip adress of scanner
46     :type ip: str
47     :param port: port of scanner
48     :type port: int
49     :return: socket to scanner
50     :rtype: socket.socket
51     """
52
53     # Create Datagram Socket (UDP)
54     try:
55         sock = socket.socket(socket.AF_INET,      # Socket Family: IPv4
56                               socket.SOCK_DGRAM) # Socket Type: UDP
57         print('Socket created!')
58     except socket.error:
59         print('Could not create socket!')
60         sys.exit()
61
62     # Sockets Options
63     sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
64     # Erlaubt, dass man sich auf einem Rechner mehrfach mit
65     # einem Port verbinden kann. (optional)
66     sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
67     # Allows broadcast UDP packets to be sent and received.
68
69     # Bind socket to local host and port
70     try:
71         sock.bind((ip, port))
72     except socket.error:
73         print('Bind failed.')
74
75     print('Socket connected!')
76
77     # now keep talking with the client

```

```

78         print('Listening on: ' + ip + ':' + str(port))
79
80     return sock

```

## A.5 vdGNSSTime.py

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """
5  @author: Florian Timm
6  @version: 2017.11.18
7  """
8
9  import datetime
10 from threading import Thread
11 import socket
12 import os
13 import serial
14
15 from vdInterface import VdInterface
16
17
18 class VdGNSSTime(Thread):
19
20     """
21     system time by gnss data
22     """
23
24     def __init__(self, master):
25         """
26             Constructor
27             :param master: instance of VdAutostart
28             :type master: VdAutoStart
29         """
30         Thread.__init__(self)
31         self.tScanner = None
32         self.tSerial = None
33         self.__master = master
34         self.__conf = master.conf
35         self.__time_corrected = False
36
37     def run(self):
38         """
39             starts threads for time detection
40         """
41         # Thread zur Erkennung der seriellen Schnittstelle
42         self.tSerial = Thread(target=self.__get_gnss_time_from_serial())
43         self.tSerial.start()
44
45         # Thread zur Erkennung des Scanners
46         self.tScanner = Thread(target=self.__get_gnss_time_from_scanner())
47         self.tScanner.start()
48
49         self.__master.gnss_status = "Connecting..."
50
51     def __get_gnss_time_from_scanner(self):
52         """ gets data by scanner network stream """

```

```

53     sock = VdInterface.get_gnss_stream(self.__conf)
54     sock.settimeout(1)
55     self.__master.gnss_status = "Wait for fix..."
56     while not self.__time_corrected:
57         # Daten empfangen vom Scanner
58         # print("Daten kommen...")
59         try:
60             data = sock.recvfrom(2048)[0] # buffer size is 2048 bytes
61             message = data[206:278].decode('utf-8', 'replace')
62             if self.__get_gnss_time_from_string(message):
63                 break
64         except socket.timeout:
65             continue
66         # else:
67         #     print(message)
68         if data == 'QUIT':
69             break
70     sock.close()

72     def __get_gnss_time_from_serial(self):
73         """ get data by serial port """
74         ser = None
75         try:
76             port = self.__conf.get("Seriell", "GNSSport")
77             ser = serial.Serial(port, 9600, timeout=1)
78             self.__master.gnss_status = "Wait for fix..."
79             while not self.__time_corrected:
80                 line = ser.readline()
81                 message = line.decode('utf-8', 'replace')
82                 if self.__get_gnss_time_from_string(message):
83                     break
84                 # else:
85                 #     print(message)
86         except serial.SerialTimeoutException:
87             pass
88         except serial.serialutil.SerialException:
89             print("Could not open serial port!")
90         finally:
91             if ser is not None:
92                 ser.close()

94     def __get_gnss_time_from_string(self, message):
95         if message[0:6] == "$GPRMC":
96             p = message.split(",")
97             if p[2] == "A":
98                 print("GNSS-Fix")
99                 timestamp = datetime.strptime(p[1] + "D" + p[9],
100                                              '%H%M%S.000%d%m%y')
101                 self.__set_system_time(timestamp)
102                 self.__time_corrected = True
103                 self.__master.gnss_status = "Got time!"
104                 return True
105             return False

107     def __set_system_time(self, timestamp):
108         """
109             sets system time
110             :param timestamp: current timestamp
111             :type timestamp: datetime

```

```

112         :return:
113         :rtype:
114         """
115         os.system("timedatectl set-ntp 0")
116         os.system("timedatectl set-time \"\" +
117                     timestamp.strftime("%Y-%m-%d %H:%M:%S") + "\"")
118         os.system("timedatectl set-ntp 1")
119         self.__master.set_gnss_status("System time set")

121     def stop(self):
122         """ stops all threads """
123         self.__master.gnss_status = "Stopped"
124         self.__time_corrected = True

```

## A.6 vdHardware.py

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-

4  """
5  @author: Florian Timm
6  @version: 2017.11.17
7  """

9  import RPi.GPIO as GPIO
10 import time
11 from threading import Thread
12 import threading

15 class VdHardware(Thread):

17     """ controls hardware control, extends Thread """

19     def __init__(self, master):
20         """
21             Constructor
22             :param master: instance of VdAutoStart
23             :type master: VdAutoStart
24         """
25         Thread.__init__(self)

27         GPIO.setmode(GPIO.BCM)

29         self.__taster1 = 18 # start / stop
30         self.__taster2 = 25 # shutdown

32         # led-pins:
33         # 0: receiving
34         # 1: queue
35         # 2: recording
36         self.__led = [10, 9, 11]
37         self.__receiving = False
38         self.__queue = False
39         self.__recording = False

41         # Masterobjekt sichern
42         self.__master = master

```

```

44         # Eingaenge aktivieren
45         # Aufzeichnung starten/stoppen
46         GPIO.setup(self.__taster1, GPIO.IN, pull_up_down=GPIO.PUD_UP)
47         # Herunterfahren
48         GPIO.setup(self.__taster2, GPIO.IN, pull_up_down=GPIO.PUD_UP)

49         # Ausgaenge aktivieren und auf Low schalten
50         for l in self.__led:
51             GPIO.setup(l, GPIO.OUT)    # GPS-Fix
52             GPIO.output(l, GPIO.LOW)

53             self.__go_on = True

54

55     def run(self):
56         """ run thread and start hardware control """
57         GPIO.add_event_detect(
58             self.__taster1,
59             GPIO.FALLING,
60             self.__button1_pressed)
61         GPIO.add_event_detect(
62             self.__taster2,
63             GPIO.FALLING,
64             self.__button1_pressed)

65             self.__timer_check_leds()

66

67     def __timer_check_leds(self):
68         """ checks leds every second """
69         self.__check_leds()
70         if self.__go_on:
71             t = threading.Timer(1, self.__timer_check_leds)
72             t.start()

73

74     def __check_leds(self):
75         """ check leds """
76         self.__set_recording(self.__master.check_recording())
77         self.__set_receiving(self.__master.check_receiving())
78         self.__set_queue(self.__master.check_queue())

79

80     def __button1_pressed(self):
81         """ raised when button 1 is pressed """
82         time.sleep(0.1)  # Prellen abwarten

83

84         # mindestens 2 Sekunden druecken
85         wait = GPIO.wait_for_edge(self.__taster1, GPIO.RISING, timeout=1900)

86

87         if wait is None:
88             # keine steigende Kante = gehalten
89             if self.__master.go_on_buffer.value:
90                 self.__master.stop_recording()
91             else:
92                 self.__master.start_recording()

93

94     def __button2_pressed(self):
95         """ raised when button 1 is pressed """
96         time.sleep(0.1)  # Prellen abwarten

97

98         # mindestens 2 Sekunden druecken
99

```

```

102     wait = GPIO.wait_for_edge(self.__taster2, GPIO.RISING, timeout=1900)
103
104     if wait is None:
105         # keine steigende Kante = gehalten
106         self.__master.shutdown()
107
108     def __switch_led(self, led, yesno):
109         """
110             switch led
111             :param led: pin of led
112             :type led: int
113             :param yesno: True = on
114             :type yesno: bool
115         """
116
117         if yesno:
118             GPIO.output(self.__led[led], GPIO.HIGH)
119         else:
120             GPIO.output(self.__led[led], GPIO.LOW)
121
122     def __update_leds(self):
123         """ switch all leds to right status """
124         self.__switch_led(0, self.__receiving)
125         self.__switch_led(1, self.__queue)
126         self.__switch_led(2, self.__recording)
127
128     def __set_receiving(self, yesno):
129         """
130             set receiving variable and led
131             :param yesno: True = on
132             :type yesno: bool
133         """
134
135         if self.__receiving != yesno:
136             self.__receiving = yesno
137             self.__update_leds()
138
139     def __set_queue(self, yesno):
140         """
141             set queue variable and led
142             :param yesno: True = on
143             :type yesno: bool
144         """
145
146         if self.__queue != yesno:
147             self.__queue = yesno
148             self.__update_leds()
149
150     def __set_recording(self, yesno):
151         """
152             set recording variable and led
153             :param yesno: True = on
154             :type yesno: bool
155         """
156
157         if self.__recording != yesno:
158             self.__recording = yesno
159             self.__update_leds()
160
161     def stop(self):
162         """ stops thread """
163         self.__go_on = False
164         GPIO.cleanup()

```

## A.7 vdFile.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-

4  """
5  @author: Florian Timm
6  @version: 2017.11.18
7  """

9  import datetime

12 class VdFile(object):

14     """
15     creates and fills an ascii-file with point data
16     """

18     def __init__(self, conf, filename="", fileformat="txt"):
19         """
20             Creates a new ascii-file
21             :param conf: configuration file
22             :type conf: configparser.ConfigParser
23             :param filename: name and path to new file
24             :type filename: str
25             :param fileformat: file suffix, default="txt"
26             :type fileformat: str
27         """
28         self.__conf = conf
29         # Dateiname erzeugen, sofern kein Dateiname mitgeliefert
30         if filename == "":
31             filename = self.__make_filename(fileformat)
32         elif not filename.endswith("." + fileformat):
33             filename += "." + fileformat
34         # Datei erzeugen
35         self.__file = open(filename, 'a')
36         self.__write_queue = []

38     def __get_write_queue(self):
39         """
40             Returns points in queue
41             :return: points in queue
42             :rtype: VdPoint []
43         """
44         return self.__write_queue
45     write_queue = property(__get_write_queue)

47     def clear_write_queue(self):
48         self.__write_queue = []

50     def __make_filename(self, fileformat):
51         """
52             generates a new filename from timestamp
53             :param fileformat: file suffix
54             :type fileformat: str
55             :return: string with date and suffix
56             :rtype: str
57         """
58         """
```

```

58     # Jahr-Monat-TagTStunde:Minute:Sekunde an Dateinamen anhaengen
59     filename = self.__conf.get("Datei", "fileNamePre")
60     filename += datetime.datetime.now().strftime(
61         self.__conf.get("Datei", "fileTimeFormat"))
62     filename = "." + fileformat
63     return filename
64
64
65     def _write2file(self, data):
66         """
67             writes ascii data to file
68             :param data: data to write
69             :type data: str
70         """
71         self.__file.write(data)
72
72
73     def write_data(self, data):
74         """
75             adds data and writes it to file
76             :param data: ascii data to write
77             :type data: VdPoint []
78         """
79         self.add_dataset(data)
80         self.write()
81
81
82     def write(self):
83         """writes data to file """
84         txt = ""
85         for d in self.write_queue:
86             if d.distance > 0.0:
87                 txt += self.format(d)
88         self._write2file(txt)
89         self.clear_write_queue()
90
90
91     def format(self, p):
92         print("not implemented")
93
93
94     def add_point(self, p):
95         """
96             Adds a point to write queue
97             :param p: point
98             :type p: VdPoint
99         """
100        self.__write_queue.append(p)
101
101
102    def add_dataset(self, dataset):
103        """
104            adds multiple points to write queue
105            :param dataset: multiple points
106            :type dataset: VdPoint []
107        """
108        self.__write_queue.extend(dataset)
109
109
110    def close(self):
111        """ close file """
112        self.__file.close()
113
113
114
115    class VdObjFile(VdFile):

```

```

117     """ creates and fills an obj-file """
118
119     def __init__(self, conf, filename=""):
120         """
121             Creates a new obj-file
122             :param conf: configuration file
123             :type conf: configparser.ConfigParser
124             :param filename: name and path to new file
125             :type filename: str
126             """
127         VdFile.__init__(self, conf, filename, "obj")
128
129     def format(self, p):
130         """
131             Format point for OBJ
132             :param p: VdPoint
133             :type p: VdPoint
134             :return: obj point string
135             :rtype: str
136             """
137         x, y, z = p.get_yxz()
138         format_string = 'v {:.3f} {:.3f} {:.3f}\n'
139         return format_string.format(x, y, z)
140
141     class VdTtxtFile(VdFile):
142
143         """ creates and fills an txt-file """
144
145         def __init__(self, conf, filename=""):
146             """
147                 Creates a new txt-file
148                 :param conf: configuration file
149                 :type conf: configparser.ConfigParser
150                 :param filename: name and path to new file
151                 :type filename: str
152                 """
153         VdFile.__init__(self, conf, filename, "txt")
154
155         def format(self, p):
156             """
157                 format point for TXT
158                 :param p: VdPoint
159                 :type p: VdPoint
160                 :return: txt point string
161                 :rtype: str
162                 """
163         format_string = '{:012.1f}\t{:07.3f}\t{:03.0f}\t{:06.3f}\t{:03.0f}\n'
164         return format_string.format(p.time,
165                                     p.azimuth,
166                                     p.vertical,
167                                     p.distance,
168                                     p.reflection)

```

## A.8 vdDataset.py

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-

```

```

4    """
5  @author: Florian Timm
6  @version: 2017.11.17
7  """

9  from vdPoint import VdPoint
10 import json

13 class VdDataset(object):

15     """ representation of one dataset of velodyne vlp-16 """

17     def __init__(self, conf, dataset):
18         """
19             Constructor
20             :param conf: config-file
21             :type conf: configparser.ConfigParser
22             :param dataset: binary dataset
23             :type dataset: bytes
24         """

26         self.__dataset = dataset
27         self.__conf = conf

29         self.__vertAngle = json.loads(self.__conf.get("Geraet", "vertAngle"))
30         self.__offset = json.loads(self.__conf.get("Geraet", "offset"))
31         self.__data = []

33     def get_azimuth(self, block):
34         """
35             gets azimuth of a data block
36             :param block: number of data block
37             :type block: int
38             :return: azimuth
39             :rtype: float
40         """

42         offset = self.__offset[block]
43         # Horizontalrichtung zusammensetzen, Bytereihenfolge drehen
44         azi = ord(self.__dataset[offset + 2:offset + 3]) + \
45             (ord(self.__dataset[offset + 3:offset + 4]) << 8)
46         azi /= 100.0
47         # print(azi)
48         return azi

50     def get_time(self):
51         """
52             gets timestamp of dataset
53             :return: timestamp of dataset
54             :rtype: int
55         """

57         time = ord(self.__dataset[1200:1201]) + \
58             (ord(self.__dataset[1201:1202]) << 8) + \
59             (ord(self.__dataset[1202:1203]) << 16) + \
60             (ord(self.__dataset[1203:1204]) << 24)
61         # print(time)
62         return time

```

```

64     def is_dual_return(self):
65         """
66             checks wheater dual return is activated
67             :return: dual return active?
68             :rtype: bool
69             """
70
71         mode = ord(self._dataset[1204:1205])
72         if mode == 57:
73             return True
74         else:
75             return False
76
77     def get_azimuths(self):
78         """
79             get all azimuths and rotation angles from dataset
80             :return: azimuths and rotation angles
81             :rtype: list, list
82             """
83
84         # Leere Listen erzeugen
85         azimuths = [0.] * 24
86         rotation = [0.] * 12
87
88         # Explizit uebermittelte Azimut-Werte einlesen
89         for j in range(0, 24, 2):
90             a = self.get_azimuth(j // 2)
91             azimuths[j] = a
92
93         # Drehwinkelvariable initialisieren
94         d = 0
95
96         # DualReturn aktiv?
97         if self.is_dual_return():
98             for j in range(0, 19, 4):
99                 d2 = azimuths[j + 4] - azimuths[j]
100                if d2 < 0:
101                    d2 += 360.0
102                d = d2 / 2.0
103                a = azimuths[j] + d
104                azimuths[j + 1] = a
105                azimuths[j + 3] = a
106                rotation[j // 2] = d
107                rotation[j // 2 + 1] = d
108            # Zweiten
109            rotation[10] = d
110            azimuths[21] = azimuths[20] + d
111
112         # Strongest / Last-Return
113         else:
114             for j in range(0, 22, 2):
115                 d2 = azimuths[j + 2] - azimuths[j]
116                 if d2 < 0:
117                     d2 += 360.0
118                 d = d2 / 2.0
119                 a = azimuths[j] + d
120                 azimuths[j + 1] = a
121                 rotation[j // 2] = d

```

```

123     # letzter Drehwinkel wird immer vom vorherigen uebernommen,
124     # letzte Horizontalrichtung ergibt sich aus diesem
125     rotation[11] = d
126     azimuths[23] = azimuths[22] + d

128     # Auf Werte ueber 360 Grad pruefen
129     for j in range(24):
130         if azimuths[j] > 360.0:
131             azimuths[j] -= 360.0

133     # print (azimuths)
134     # print (rotation)
135     return azimuths, rotation

137 def convert_data(self):
138     """ converts binary data to objects """
139
140     # Zeitstempel aus den Daten auslesen
141     zeit = self.get_time()
142
143     # Richtung und Drehwinkel auslesen
144     azimuth, rotation = self.get_azimuths()
145
146     dual_return = self.is_dual_return()
147     t_between_laser = float(self.__conf.get("Geraet", "tZwischenStrahl"))
148     t_recharge = float(self.__conf.get("Geraet", "tNachlade"))
149     part_rotation = float(self.__conf.get("Geraet", "antDrehung"))

150     # Datenpaket besteht aus 12 Bloecken aus jeweils 32 Messergebnissen
151     for i in range(12):
152         offset = self.__offset[i]
153         for j in range(2):
154             azi_block = azimuth[i + j]
155             for k in range(16):
156                 # Entfernung zusammensetzen
157                 dist = ord(self.__dataset[4 + offset:5 + offset]) \
158                     + (ord(self.__dataset[5 + offset:6 + offset]) << 8)
159                 dist /= 500.0
160
161                 # Reflektivitaet auslesen
162                 refl = ord(self.__dataset[6 + offset:7 + offset])
163
164                 # Offset in Daten fuer den naechsten Durchlauf
165                 offset += 3
166
167                 # Horizontalwinkel interpolieren
168                 a = azi_block + rotation[i] * k * part_rotation
169
170                 # a += zeit / 1000000 * 0.1
171
172                 # Punkt erzeugen und anhaengen
173                 p = VdPoint(
174                     self.__conf, round(zeit, 1), a, self.__vertAngle[k],
175                     dist, refl)
176                 self.__data.append(p)
177                 zeit += t_between_laser
178
179                 if dual_return and j == 0:

```

```
181                     zeit -= t_between_laser * 16
182             else:
183                 zeit += t_recharge
184
185     def get_data(self):
186         """
187             get all point data
188         :return: list of VdPoints
189         :rtype: list
190         """
191     return self.__data
```

## A.9 vdPoint.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """
5  @author: Florian Timm
6  @version: 2017.11.17
7  """
8
9  import math
10
11
12 class VdPoint(object):
13
14     """ Represents a point """
15     __dRho = math.pi / 180.0
16
17     def __init__(self, conf, time, azimuth, vertical, distance, reflection):
18         """
19             Constructor
20             :param conf: config file
21             :type conf: configparser.ConfigParser
22             :param time: recording time in microseconds
23             :type time: float
24             :param azimuth: Azimuth direction in degrees
25             :type azimuth: float
26             :param vertical: Vertical angle in degrees
27             :type vertical: float
28             :param distance: distance in metres
29             :type distance: float
30             :param reflection: reflection 0-255
31             :type reflection: int
32         """
33
34         self.__time = time
35         self.__azimuth = azimuth
36         self.__vertical = vertical
37         self.__reflection = reflection
38         self.__distance = distance
39         self.__conf = conf
40
41     def __deg2rad(self, degree):
42         return degree * self.__dRho
43
44     def get_yxz(self):
45         """
46             Returns the point as a tuple (x,y,z) in metres
47             :return: tuple (x,y,z)
48         """
49         x = self.__distance * math.sin(math.radians(self.__vertical)) * math.cos(
50             math.radians(self.__azimuth))
51         y = self.__distance * math.sin(math.radians(self.__vertical)) * math.sin(
52             math.radians(self.__azimuth))
53         z = self.__distance * math.cos(math.radians(self.__vertical))
54
55         return (x, y, z)
```

```

45     Gets local coordinates
46     :return: local coordinates x, y, z in metres
47     :rtype: float, float, float
48     """
49     beam_center = float(self._conf.get("Geraet", "beamCenter"))

51     # Schraegstrecke zum Strahlenzentrum
52     d = self.distance - beam_center

54     # Vertikalwinkel in Bogenmass
55     v = self.vertical_radians

57     # Azimut in Bogenmass
58     a = self.azimuth_radians

60     # Horizontalstrecke bis Drehpunkt
61     s = d * math.cos(v) + beam_center

63     x = s * math.sin(a)
64     y = s * math.cos(a)
65     z = d * math.sin(v)

67     return x, y, z

69     def __get_time(self):
70     """
71         Gets recording time
72         :return: recording time in microseconds
73         :rtype: float
74     """
75     return self._time

77     def __get_azimuth(self):
78     """
79         Gets azimuth direction
80         :return: azimuth direction in degrees
81         :rtype: float
82     """
83     return self._azimuth

85     def __get_azimuth_radians(self):
86     """
87         Gets azimuth in radians
88         :return: azimuth direction in radians
89         :rtype: float
90     """
91     return self._deg2rad(self.azimuth)

93     def __get_vertical(self):
94     """
95         Gets vertical angle in degrees
96         :return: vertical angle in degrees
97         :rtype: float
98     """
99     return self._vertical

101    def __get_vertical_radians(self):
102    """
103        Gets vertical angle in radians

```

```

104         :return: vertical angle in radians
105         :rtype: float
106         """
107         return self._deg2rad(self.vertical)

109     def __get_reflection(self):
110         """
111         Gets reflection
112         :return: reflection between 0 and 255
113         :rtype: int
114         """
115         return self._reflection

117     def __get_distance(self):
118         """
119         Gets distance
120         :return: distance in metres
121         :rtype: float
122         """
123         return self._distance

125     # properties
126     time = property(__get_time)
127     azimuth = property(__get_azimuth)
128     azimuth_radians = property(__get_azimuth_radians)
129     vertical = property(__get_vertical)
130     vertical_radians = property(__get_vertical_radians)
131     reflection = property(__get_reflection)
132     distance = property(__get_distance)

```

## A.10 config.ini

```

1 [Netzwerk]
2 UDP_IP = 0.0.0.0
3 UDP_PORT_DATA = 2368
4 UDP_PORT_GNSS = 8308

6 [Seriell]
7 # Serieller Port
8 #Raspberry
9 GNSSport = /dev/ttyAMA0
10 #Ubuntu
11 #GNSSport = /dev/ttyUSBO

13 [Funktionen]
14 # Zeitgleiche Transformation zu txt aktivieren
15 activateTransformer = True

17 # GNSS-Zeit verwenden
18 GNSSZeitVerwenden = True

22 [Datei]
23 # Binaere Dateien nach deren Transformation loeschen
24 binNachTransLoeschen = True

26 #Takt zur Speicherung Buffer -> HDD

```

```

27 takt = 5

29 # Format der Zeit am Dateinamen
30 fileTimeFormat = '%Y-%m-%dT%H:%M:%S'

32 # Dateipraefix der zu speichernden Datei
33 fileNamePre = data

35 [Geraet]
36 # Zeit zwischen den Messungen der Einzelstrahlen
37 tZwischenStrahl = 2.304

39 # Zeit zwischen zwei Aussendungen des gleichen Messlasers
40 tRepeat = 55.296

42 # Hoehenwinkel der 16 Messstrahlen
43 vertAngle = [-15, 1, -13, -3, -11, 5, -9, 7, -7, 9, -5, 11, -3, 13, -1, 15]

45 messungProDatensatz = 384
46 #12*32

48 # Bytes pro Messdatenblock
49 offsetBlock = 100 # 3 * 32 + 4

51 # Versatz vom Start fuer jeden Messblock
52 offset = [0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100]
53 #list(range(0,1206,offsetBlock))[0:12]

55 # Anteil der Zeit zwischen Einzellasern an Wiederholungszeit,
56 # fuer Interpolation des Horizontalwinkels
57 antDrehung = 0.04166666666666664
58 #tZwischenStrahl / tRepeat

60 # Zeit nach letztem Strahl bis zum naechsten
61 tNachlade = 20.736
62 #tRepeat - 15 * tZwischenStrahl

64 # Abstand des Strahlenzentrums von der Drehachse
65 beamCenter = 0.04191

```

## A.11 convTxt2Obj.py

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-

4 """
5 @author: Florian Timm
6 @version: 2017.10.22
7 """
8 from vdPoint import VdPoint
9 from vdFile import VdObjFile
10 import configparser

13 class ConvTxt2Obj:

15     def __init__(self):
16         """ Constructor """

```

```

17         self.fileName = "BeispielDateien/test.txt"
18
19     def run(self):
20         """ starts script """
21         data = []
22         conf = configparser.ConfigParser()
23         conf.read("config.ini")
24         txt = open(self.fileName, 'rb')
25         f = VdObjFile(conf, self.fileName)
26         for line in txt:
27             dataline = line.split()
28             data.append(
29                 VdPoint(
30                     conf, float(dataline[0]), float(
31                         dataline[1]), float(dataline[2]),
32                         float(dataline[3]), int(dataline[4])))
33             # print ("test")
34             if len(data) > 50000:
35                 f.write_data(data)
36                 data = []
37             f.write_data(data)
38
39
40 if __name__ == '__main__':
41     ConvTxt20bj().run()

```

# B Beispieldateien

## B.1 Rohdaten vom Scanner

Netzwerk-Header

Flag (FF EE) Horizontalrichtung

Strecke Reflektivität

Timestamp Return-Modus

```
0000      ff ff ff ff ff ff 60 76 88 00 00 00 00 08 00 45 00
0010      04 d2 00 00 40 00 ff 11 b4 aa c0 a8 01 c8 ff ff
0020      ff ff 09 40 09 40 04 be 00 00 ff ee 02 4d 00 00
0030      0f 00 00 0a 00 00 16 f0 01 04 00 00 0d 00 00 0a
0040      00 00 0d 00 00 06 00 00 0b 00 00 08 00 00 10 00
0050      00 05 92 01 05 00 00 04 00 00 0f 00 00 07 00 00
0060      0f 00 00 0a 00 00 16 e6 01 08 00 00 0d 00 00 0a
0070      00 00 0d 00 00 06 00 00 0b 00 00 08 00 00 10 00
0080      00 05 7e 01 05 00 00 04 00 00 0f 00 00 07 ff ee
0090      02 4d 00 00 0f 00 00 0a 00 00 16 f0 01 04 00 00
...
04d0      05 00 00 04 00 00 0f 00 00 07 8c 25 44 63 39 22
```

## B.2 Dateiformat für Datenspeicherung als Text

```
1 210862488 36.18 -15 2.234 46
2 210862490.304 36.188 1 2.18 46
3 210862492.60799998 36.197 -13 2.214 41
4 210862494.91199997 36.205 -3 2.16 42
5 210862497.21599996 36.213 -11 2.204 50
6 210862499.51999995 36.222 5 2.164 55
7 210862501.82399994 36.23 -9 2.184 47
8 210862504.12799993 36.238 7 2.17 42
9 210862506.43199992 36.247 -7 2.192 43
10 210862508.7359999 36.255 9 2.21 40
11 210862511.0399999 36.263 -5 2.186 41
12 210862513.3439999 36.272 11 2.206 46
13 210862515.64799988 36.28 -3 2.182 47
```

```

14 210862517.95199987 36.288 13 2.192 42
15 210862520.25599986 36.297 -1 2.16 43
16 210862522.55999985 36.305 15 2.214 47
17 210862545.59999985 36.38 -15 2.224 45
18 210862547.90399984 36.388 1 2.168 48
19 210862550.20799983 36.397 -13 2.192 39
20 210862552.51199982 36.405 -3 2.152 44

```

## B.3 Dateiformat für Datenspeicherung als OBJ

```

1 v 1.2746902491848617 1.7429195788236858 -0.5673546405787847
2 v 1.2869596160904488 1.7591802795096634 0.037314815679491506
3 v 1.274630423722104 1.741752967944279 -0.48861393562976563
4 v 1.274145749563782 1.7405806715041912 -0.1108522655586169
5 v 1.2786300911436552 1.7461950253652434 -0.41254622081367376
6 v 1.2739693304356217 1.7392567142322626 0.1849523301273779
7 v 1.2752183957072614 1.7404521494414935 -0.33509670321802815
8 v 1.2733984465128143 1.7374593339109177 0.25934893100706025
9 v 1.2865822747749043 1.7548695003015347 -0.26203005656197353
10 v 1.291164267762529 1.7606036538453675 0.33916399930907415
11 v 1.2881769097946472 1.756015963302377 -0.1868697564678264
12 v 1.2815890463056323 1.7464602478174986 0.4129278388044268
13 v 1.2894233312788135 1.7566219901831923 -0.11200365659596165
14 v 1.264708293723956 1.7224476905470605 0.4836650124342007
15 v 1.2784663490171557 1.7406120436549135 -0.036965767550745834
16 v 1.2670514982720475 1.7245661497369091 0.5621782596767342
17 v 1.2750371359697306 1.730682783609054 -0.5647664501277595
18 v 1.2859745413187607 1.7450184890932041 0.0371053868022441
19 v 1.267982802947427 1.7200385827982603 -0.4836650124342007
20 v 1.2754723412692703 1.729692556621396 -0.11043357790867334

```

## **Erklärung**

Hiermit versichere ich, dass ich die beiliegende Bachelor-Thesis ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe.

Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Hamburg, den 12. Dez. 2017

---

Ort, Datum

Florian Timm