

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

1 Einleitung	1
1.1 Problemstellung	1
1.2 Zielsetzung	1
1.3 Struktur	2
2 Grundlagen des Airborne Laserscanings	3
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
3 Technische Realisierung	10
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
4 Theoretische Datenverarbeitung	23
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
5	Entwicklung des Skriptes	29
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
6	Konfiguration des Raspberry Pi	36
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
7	Systemüberprüfungen	40
7.1	Untersuchung der Gleichzeitigkeit von PPS-Signalen von verschiedenen GNSS-Empfängern	40
7.2	Messgenauigkeit des Laserscanners im Vergleich	40
8	Ausblick	41
Literaturverzeichnis		42
Abbildungsverzeichnis		44
Tabellenverzeichnis		45
Anhang		46

A Python-Skripte	47
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	65
A.8 vdDataset.py	69
A.9 vdPoint.py	72
A.10 config.ini	75
A.11 convTxt2Obj.py	76
A.12 convBin2Obj.py	76
B Beispieldateien	78
B.1 Rohdaten vom Scanner	78
B.2 Dateiformat für Datenspeicherung als Text	78
B.3 Dateiformat für Datenspeicherung als OBJ	79

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-

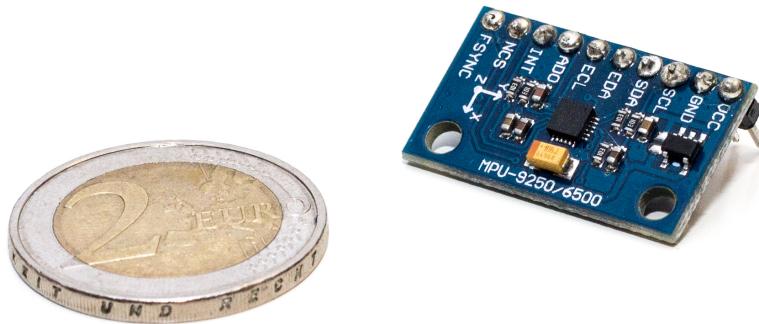


Abbildung 2.1: MPU-9250 - Low-Cost-MEMS-IMU-Modul wie es in vielen Consumer-Geräten und Multikoptern verwendet wird (schwarzes Bauteil mittig auf der Platine, eigene Aufnahme)

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 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)

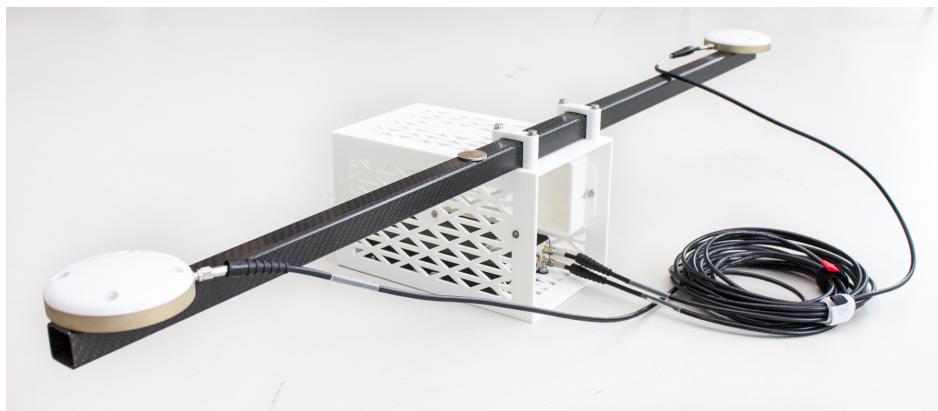


Abbildung 3.3: GNSS-Antennen des (links und rechts) iMAR iNAT-M200-Flat an Prototypen des modularen Gehäuses (eigene Aufnahme)

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-

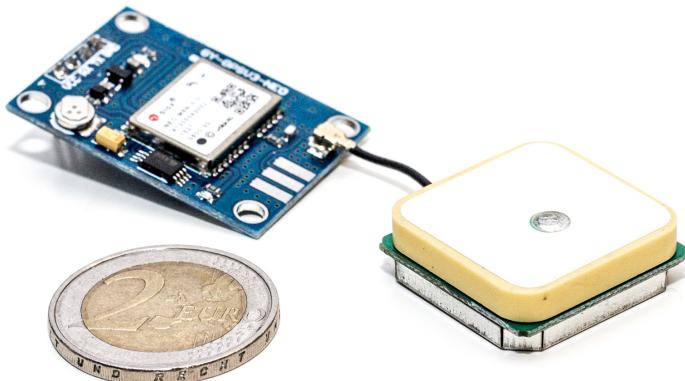


Abbildung 3.6: uBlox NEO-M8N, das Vorgängermodell NEO-6M mit PPS-Ausgang wurde verwendet (eigene Aufnahme)

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

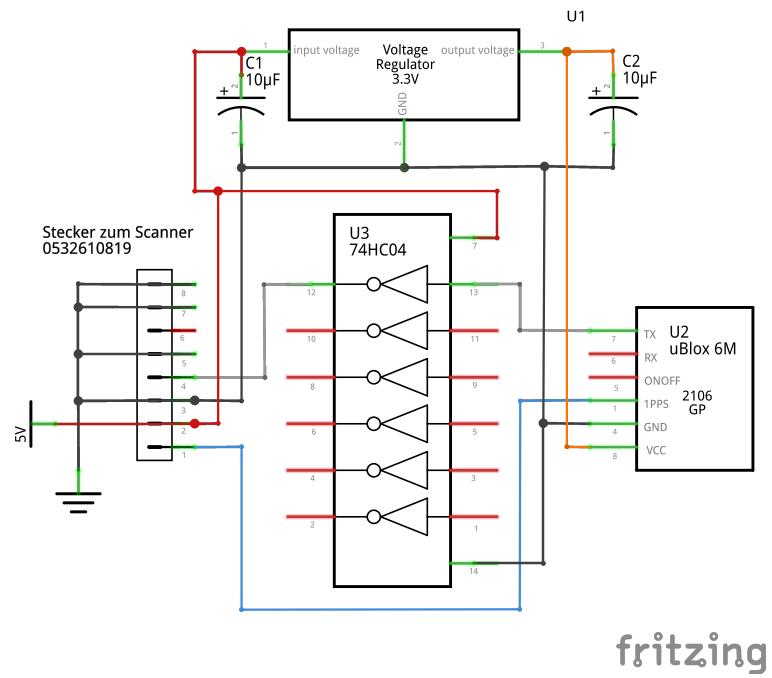
Der Betrieb des Raspberry Pi erfolgt im Betrieb ohne Tastatur und Bildschirm. Daher ist es notwendig, eine alternative Benutzerschnittstelle zu implementieren. Ein großer Steuerbedarf ist nicht gegeben, so dass wenige Tasten zum Stoppen der Datenaufzeichnung und zum Herunterfahren des Raspberry Pi ausreichend 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?

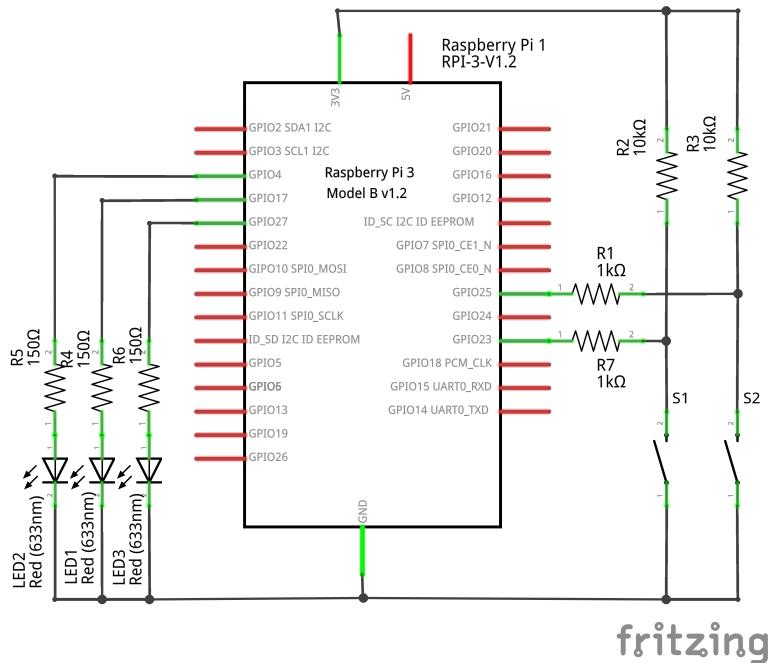
Widerstand

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 gelöteten 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 verlötet 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

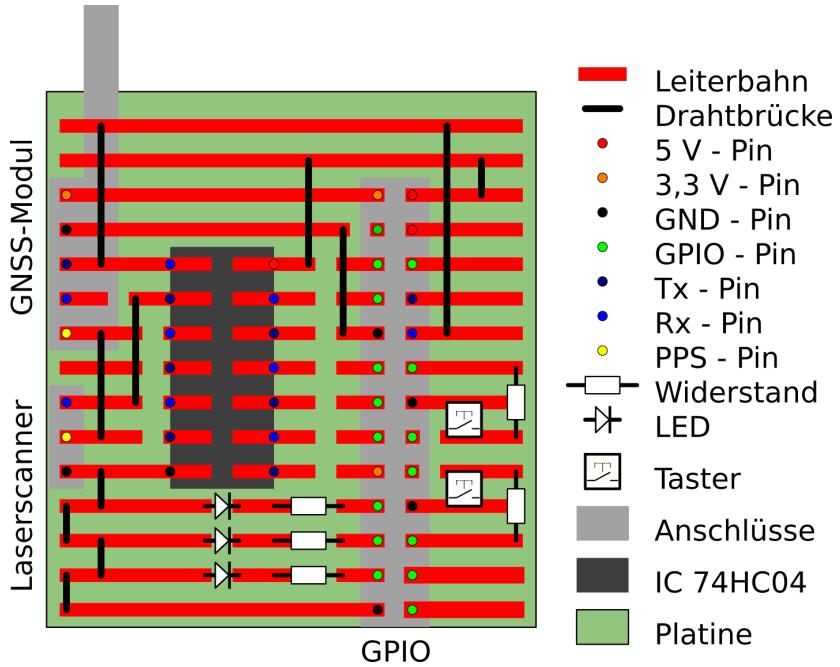


Abbildung 3.10: Layout der Lochstreifenplatine

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.

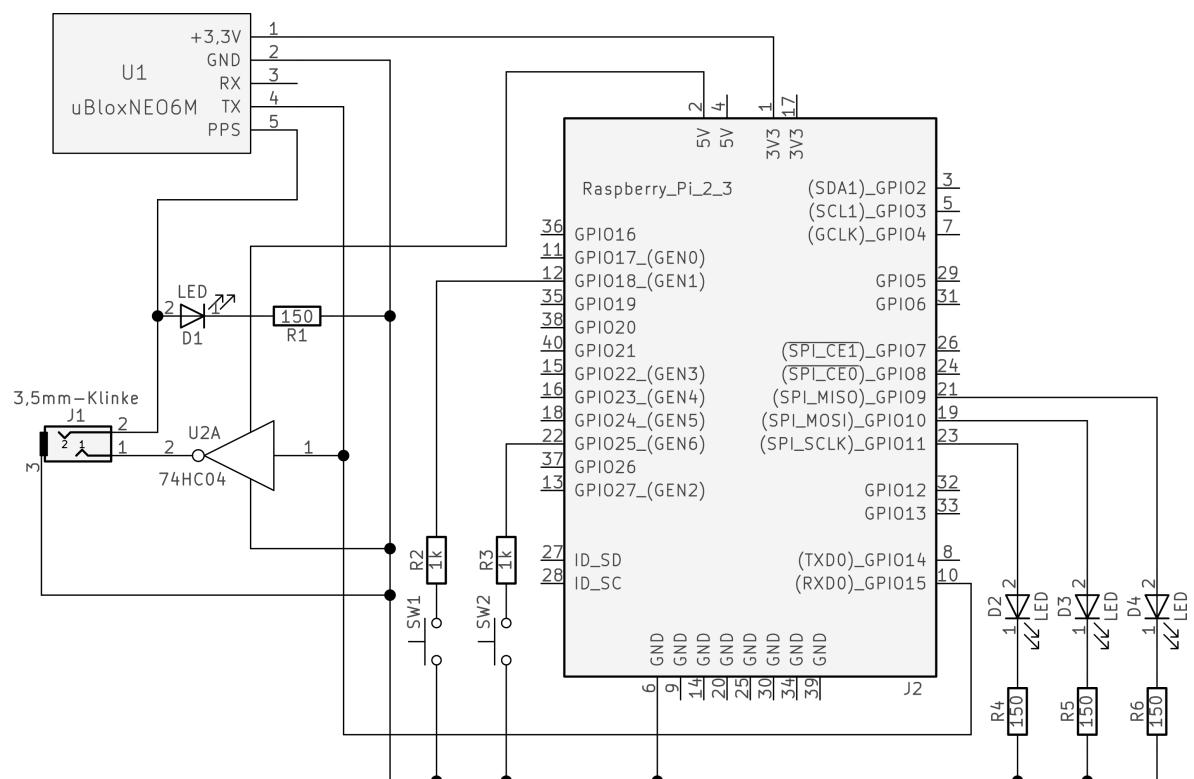


Abbildung 3.11: Endg ltiger Schaltplan

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.

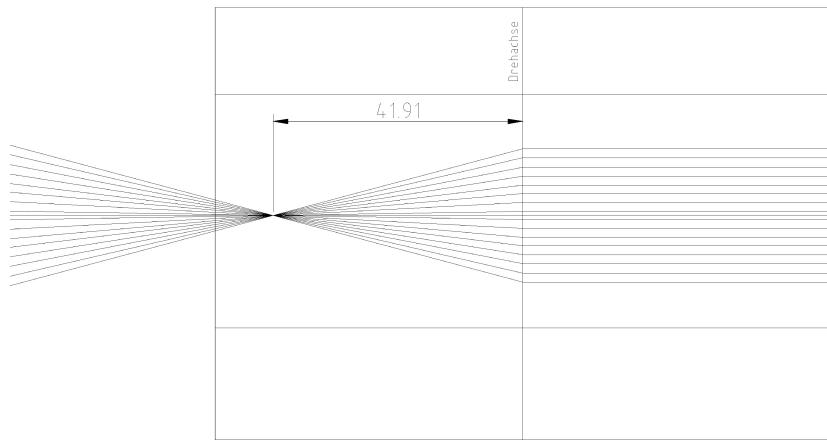


Abbildung 4.1: Strahlengang im Laserscanner VLP-16, Werte in Millimetern, nach Velodyne Lidar (2014)

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.

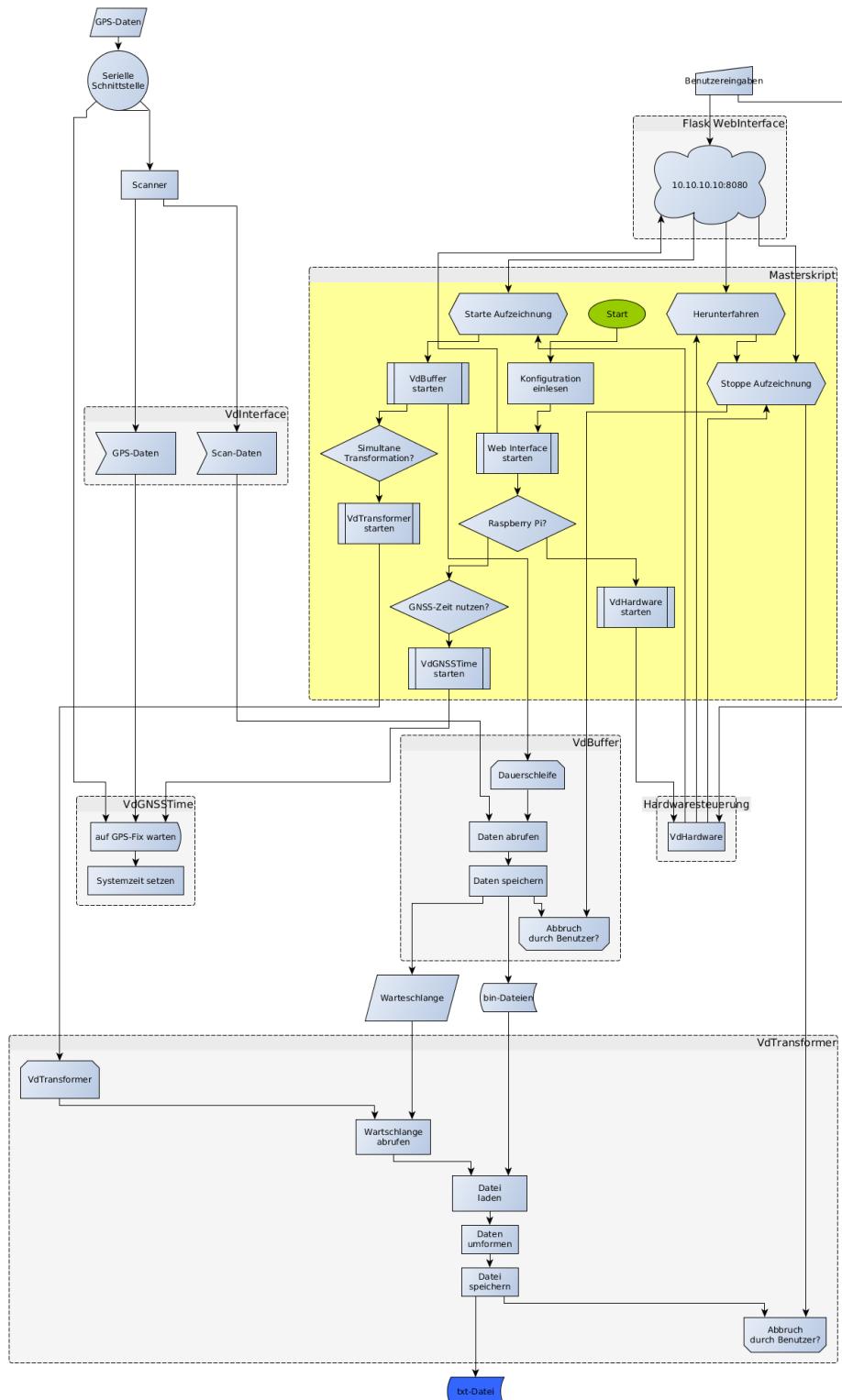


Abbildung 4.2: Vereinfachter Ablaufplan des Skriptes

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

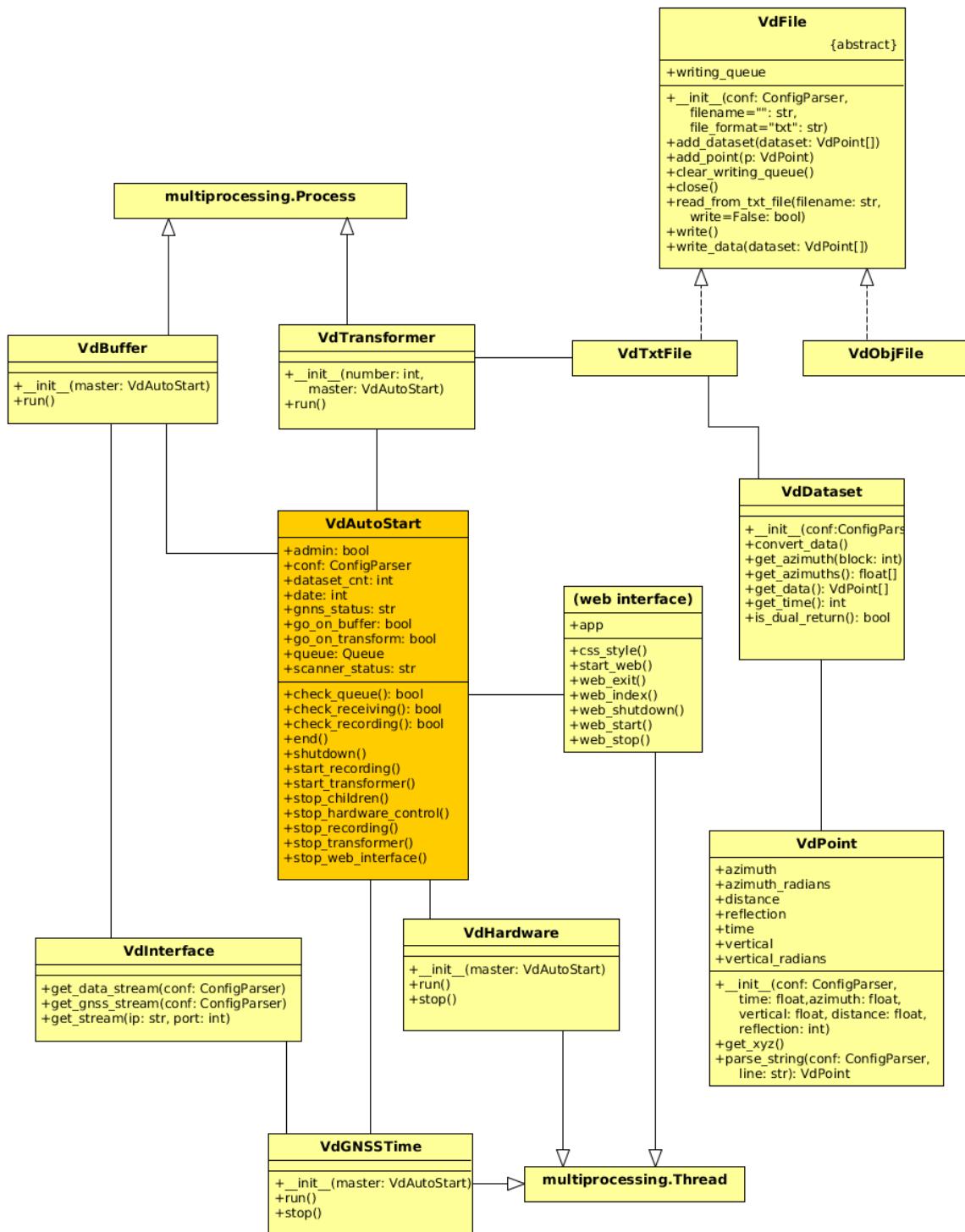


Abbildung 5.1: UML-Klassendiagramm

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

```
16     """ creates and fills an ascii-file with point data """
17
18     def __init__(self, conf, filename="", file_format="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.

Copperwaite, Matt (2015): Learning Flask Framework. Packt Publishing, Birmingham, ISBN 978-1-783-98336-0.

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.

Kleuker, Stephan (2013): Grundkurs Software-Engineering mit UML - Der pragmatische Weg zu erfolgreichen Softwareprojekten. Springer-Verlag, Berlin Heidelberg New York, ISBN 978-3-658-00642-6.

Lott, Steven F. (2014): Mastering Object-oriented Python. Packt Publishing Ltd, Birmingham, ISBN 978-1-783-28098-8.

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/>. (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. (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	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: 2017.11.19
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("functions", "use_gnss_time") == "True":
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("functions", "activateTransformer") == "True":
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         """ stops buffering data """
129         print("Recording is stopping... (10 seconds timeout before kill)")
130         self.__go_on_buffer.value = False
131         self.__date.value = None
132         if self.__pBuffer is not None:
133             self.__pBuffer.join(10)
134             if self.__pBuffer.is_alive():
135                 print("Could not stop process, it will be killed!")
136                 self.__pBuffer.terminate()
137                 print("Recording terminated!")
138         else:
139             print("Recording was not started!")

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

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

163     def stop_hardware_control(self):

```

```

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

```

```

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

```

```

282         :rtype: Manager
283         """
284     return self._dataset_cnt
285 dataset_cnt = property(_get_dataset_cnt)

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

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

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

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

324 # web control
325 app = Flask(__name__)

328 @app.route("/")
329 def web_index():
330     """ index page of web control """
331     runtime = "(inactive)"
332     pps = "(inactive)"
333     if ms.date.value is not None:
334         time_diff = datetime.now() - ms.date.value
335         td_sec = time_diff.seconds + \
336             (int(time_diff.microseconds / 1000) / 1000.)
337         seconds = td_sec % 60
338         minutes = int((td_sec // 60) % 60)
339         hours = int(td_sec // 3600)

```

```

341         runtime = '{:02d}:{:02d} {:06.3f}'.format(hours, minutes, seconds)
343
344         pps = '{:.0f}'.format(ms.dataset_cnt.value / td_sec)
345
346     elif ms.go_on_buffer.value:
347         runtime = "(no data)"
348
349     output = """<html>
350         <head>
351             <title>VLP16-Data-Interface</title>
352             <meta name="viewport" content="width=device-width; initial-scale=1.0;" />
353             <link href="/style.css" rel="stylesheet">
354             <meta http-equiv="refresh" content="5; URL=/">
355         </head>
356         <body>
357             <h2>VLP16-Data-Interface</h3>
358             <table style="">
359                 <tr><td id="column1">GNSS-status:</td><td>"""+ ms.gnss_status + """</td></tr>
360                 <tr><td>Scanner:</td><td>"""+ ms.scanner_status.value + """</td></tr>
361                 <tr><td>Datasets</td>
362                     <td>"""+ str(ms.dataset_cnt.value) + """</td></tr>
363                 <tr><td>Queue:</td>
364                     <td>"""+ str(ms.queue.qsize()) + """</td></tr>
365                 <tr><td>Recording time:</td>
366                     <td>"""+ runtime + """</td>
367                 </tr>
368                 <tr><td>Points/seconds:</td>
369                     <td>"""+ pps + """</td>
370                 </tr>
371             </table><br />
372             """
373
374     if ms.check_recording():
375         output += """<a href="/stop" id="stop">
376             Stop recording</a><br />"""
377     else:
378         output += """<a href="/start" id="start">
379             Start recording</a><br />"""
380
381     output += """
382         <a href="/exit" id="exit">Terminate script<br />
383         (control by SSH available only)</a></td></tr><br />
384         <a href="/shutdown" id="shutdown">Shutdown Raspberry Pi</a>
385     </content>
386     </body>
387     </html>"""
388
389     return output
390
391 @app.route("/style.css")
392 def css_style():
393     """
394         css file of web control """
395
396     body, html, content =
397         text-align: center;
398     }
399
400     content {

```

```

399         max-width: 15cm;
400         display: block;
401         margin: auto;
402     }

404     table {
405         border-collapse: collapse;
406         width: 90%;
407         margin: auto;
408     }

410     td {
411         border: 1px solid black;
412         padding: 1px 2px;
413     }

415     td#column1 {
416         width: 30%;
417     }

419     a {
420         display: block;
421         width: 90%;
422         padding: 0.5em 0;
423         text-align: center;
424         margin: auto;
425         color: #fff;
426     }

428     a#stop {
429         background-color: #e90;
430     }

432     a#shutdown {
433         background-color: #b00;
434     }

436     a#start {
437         background-color: #1a1;
438     }

440     a#exit {
441         background-color: #f44;
442     }
443     """
444

446 @app.route("/shutdown")
447 def web_shutdown():
448     """ web control: shutdown """
449     ms.shutdown()
450     return """
451     <meta http-equiv="refresh" content="3; URL=/">
452     Shutdown..."""

455 @app.route("/exit")
456 def web_exit():
457     """ web control: exit """

```

```

458     ms.end()
459     return """
460     <meta http-equiv="refresh" content="3; URL=/">
461     Terminating..."""
462
463
464 @app.route("/stop")
465 def web_stop():
466     """ web control: stop buffering """
467     ms.stop_recording()
468     return """
469     <meta http-equiv="refresh" content="3; URL=/">
470     Recording is stopping..."""
471
472
473 @app.route("/start")
474 def web_start():
475     """ web control: start buffering """
476     ms.start_recording()
477     return """
478     <meta http-equiv="refresh" content="3; URL=/">
479     Recording is starting..."""
480
481
482 def start_web():
483     """ start web control """
484     print("Web server is starting...")
485     app.run('0.0.0.0', 8080)
486
487
488 if __name__ == '__main__':
489     w = Thread(target=start_web)
490     ms = VdAutoStart(w)
491     w.start()
492     ms.run()

```

A.2 vdBuffer.py

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """
5  @author: Florian Timm
6  @version: 2017.11.19
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 """

```

```

21     def __init__(self, master):
22         """
23             Constructor
24             :param master: instance of VdAutoStart
25             :type master: VdAutoStart
26         """
27         # constructor of super class
28         Process.__init__(self)
29
30         # safe pipes
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
40         self.__file_no = 0
41
42     @staticmethod
43     def __signal_handler(sig_no, frame):
44         """
45             handles SIGINT-signal
46             :param sig_no: signal number
47             :type sig_no: int
48             :param frame: execution frame
49             :type frame: frame
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         # checks time for file name and runtime
58         self.__date.value = datetime.now()
59         self.__folder = self.__conf.get("file", "namePre")
60         self.__folder += self.__date.value.strftime(
61             self.__conf.get("file", "timeFormat"))
62         # make folder
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         # open socket to scanner
71         sock = VdInterface.get_data_stream(self.__conf)
72         self.__scanner_status.value = "Socket connected"
73
74         buffer = b''
75         datasets_in_buffer = 0
76
77         self.__datasets.value = 0
78
79         # process priority

```

```

80         if self.__admin:
81             os.nice(-18)
82
83         transformer = self.__conf.get(
84             "functions",
85             "activateTransformer") == "True"
86         measurements_per_dataset = int(self.__conf.get(
87             "device", "valuesPerDataset"))
88
89         sock.settimeout(1)
90         while self.__go_on_buffering.value:
91             try:
92                 # get data from scanner
93                 data = sock.recvfrom(1248)[0]
94
95                 if datasets_in_buffer == 0 and self.__file_no == 0:
96                     self.__new_folder()
97
98                     # RAM-buffer
99                     buffer += data
100
101                     datasets_in_buffer += 1
102
103                     self.__datasets.value += measurements_per_dataset
104
105                     # safe data to file every 1500 datasets
106                     # (about 5 or 10 seconds)
107                     if (datasets_in_buffer >= 1500) or \
108                         (not self.__go_on_buffering.value):
109                         # write file
110                         f = open(
111                             self.__folder + "/" + str(self.__file_no) + ".bin",
112                             "wb")
113                         f.write(buffer)
114
115                         f.close()
116
117                         if transformer:
118                             self.__queue.put(f.name)
119
120                         # clear buffer
121                         buffer = b''
122                         datasets_in_buffer = 0
123
124                         # count files
125                         self.__file_no += 1
126
127                         if data == 'QUIT':
128                             break
129
130             except socket.timeout:
131                 print("No data")
132                 continue
133
134             sock.close()
135             self.__scanner_status.value = "recording stopped"
136             print("Disconnected!")

```

A.3 vdTransformer.py

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """

```

```

5  @author: Florian Timm
6  @version: 2017.11.19
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):
18     """ Process for transforming data from Velodyne VLP-16 """
20     def __init__(self, number, master):
21         """
22             Constructor
23             :param number: number of process
24             :type number: int
25             :param master: instance of VdAutoStart
26             :type master: VdAutoStart
27         """
29     # constructor of super class
30     Process.__init__(self)

32     self.__queue = master.queue
33     self.__number = number
34     self.__admin = master.admin
35     self.__go_on_transform = master.go_on_transform
36     self.__conf = master.conf

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

51     def run(self):
52         """
53             starts transforming process
54             signal.signal(signal.SIGINT, self.__signal_handler)

55             if self.__admin:
56                 os.nice(-15)

58             old_folder = ""

60             try:
61                 while self.__go_on_transform.value:
62                     try:
63                         # get file name from queue

```

```

64     filename = self._queue.get(True, 2)
65     folder = os.path.dirname(filename)
66     if dir != old_folder:
67         vd_file = VdTxtFile(
68             self._conf,
69             folder + "/obj_file" + str(self._number))
70         old_folder = folder

72     f = open(filename, "rb")

74     # count number of datasets
75     file_size = os.path.getsize(f.name)
76     dataset_cnt = int(file_size / 1206)

78     for i in range(dataset_cnt):
79         # read next
80         vd_data = VdDataset(self._conf, f.read(1206))

82         # convert data
83         vd_data.convert_data()

85         # add them on writing queue
86         vd_file.add_dataset(vd_data)

88         # write file
89         vd_file.write()
90         # close file
91         f.close()
92         # delete binary file
93         if self._conf.get("file", "deleteBin") == "True":
94             os.remove(f.name)
95     except Empty:
96         print("Queue empty!")
97         continue
98     except BrokenPipeError:
99         print("vdTransformer-Pipe broken")

```

A.4 vdInterface.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """
5  @author: Florian Timm
6  @version: 2017.11.19
7  """
8
9  import socket
10 import sys
11
12
13 class VdInterface(object):
14
15     """ interface to velodyne scanner """
16
17     @staticmethod
18     def get_data_stream(conf):
19         """
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
317
318
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1007
1008
1009
1009
10010
10011
10012
10013
10014
10015
10016
10017
10018
10019
10019
10020
10021
10022
10023
10024
10025
10026
10027
10028
10029
10029
10030
10031
10032
10033
10034
10035
10036
10037
10038
10038
10039
10040
10041
10042
10043
10044
10045
10046
10047
10048
10048
10049
10050
10051
10052
10053
10054
10055
10056
10057
10058
10058
10059
10060
10061
10062
10063
10064
10065
10066
10067
10068
10068
10069
10070
10071
10072
10073
10074
10075
10076
10077
10077
10078
10078
10079
10080
10081
10082
10083
10084
10085
10086
10087
10087
10088
10088
10089
10090
10091
10092
10093
10094
10095
10095
10096
10096
10097
10097
10098
10099
10099
100100
100101
100102
100103
100104
100105
100106
100107
100108
100109
100109
100110
100111
100112
100113
100114
100115
100116
100117
100118
100118
100119
100120
100121
100122
100123
100124
100125
100126
100127
100128
100128
100129
100130
100131
100132
100133
100134
100135
100136
100137
100137
100138
100139
100140
100141
100142
100143
100144
100145
100146
100147
100147
100148
100149
100150
100151
100152
100153
100154
100155
100156
100157
100157
100158
100159
100160
100161
100162
100163
100164
100165
100166
100167
100167
100168
100169
100170
100171
100172
100173
100174
100175
100176
100177
100177
100178
100179
100180
100181
100182
100183
100184
100185
100186
100187
100187
100188
100189
100190
100191
100192
100193
100194
100195
100196
100197
100197
100198
100199
100199
100200
100201
100202
100203
100204
100205
100206
100207
100208
100209
100209
100210
100211
100212
100213
100214
100215
100216
100217
100217
100218
100219
100220
100221
100222
100223
100224
100225
100226
100227
100227
100228
100229
100230
100231
100232
100233
100234
100235
100236
100237
100237
100238
100239
100240
100241
100242
100243
100244
100245
100246
100247
100247
100248
100249
100250
100251
100252
100253
100254
100255
100256
100257
100257
100258
100259
100260
100261
100262
100263
100264
100265
100266
100267
100267
100268
100269
100270
100271
100272
100273
100274
100275
100276
100277
100277
100278
100279
100280
100281
100282
100283
100284
100285
100286
100287
100287
100288
100289
100290
100291
100292
100293
100294
100295
100296
100297
100297
100298
100299
100299
100300
100301
100302
100303
100304
100305
100306
100307
100308
100308
100309
100310
100311
100312
100313
100314
100315
100316
100317
100317
100318
100319
100320
100321
100322
100323
100324
100325
100326
100327
100327
100328
100329
100330
100331
100332
100333
100334
100335
100336
100337
100337
100338
100339
100340
100341
100342
100343
100344
100345
100346
100347
100347
100348
100349
100350
100351
100352
100353
100354
100355
100356
100357
100357
100358
100359
100360
100361
100362
100363
100364
100365
100366
100367
100367
100368
100369
100370
100371
100372
100373
100374
100375
100376
100377
100377
100378
100379
100380
100381
100382
100383
100384
100385
100386
100387
100387
100388
100389
100390
100391
100392
100393
100394
100395
100396
100397
100397
100398
100399
100399
100400
100401
100402
100403
100404
100405
100406
100407
100408
100408
100409
100410
100411
100412
100413
100414
100415
100416
100417
100417
100418
100419
100420
100421
100422
100423
100424
100425
100426
100427
100427
100428
100429
100430
100431
100432
100433
100434
100435
100436
100437
100437
100438
100439
100440
100441
100442
100443
100444
100445
100446
100447
100447
100448
100449
100450
100451
100452
100453
100454
100455
100456
100457
100457
100458
100459
100460
100461
100462
100463
100464
100465
100466
100467
100467
100468
100469
100470
100471
100472
100473
100474
100475
100476
100477
100477
100478
100479
100480
100481
100482
100483
100484
100485
100486
100487
100487
100488
100489
100490
100491
100492
100493
100494
100495
100496
100497
100497
100498
100499
100499
100500
100501
100502
100503
100504
100505
100506
100507
100508
100508
100509
100510
100511
100512
100513
100514
100515
100516
100517
100517
100518
100519
100520
100521
100522
100523
100524
100525
100526
100527
100527
100528
100529
100530
100531
100532
100533
100534
100535
100536
100537
100537
100538
100539
100540
100541
100542
100543
100544
100545
100546
100547
100547
100548
100549
100550
100551
100552
100553
100554
100555
100556
100557
100557
100558
100559
100560
100561
100562
100563
100564
100565
100566
100567
100567
100568
100569
100570
100571
100572
100573
100574
100575
100576
100577
100577
100578
100579
100580
100581
100582
100583
100584
100585
100586
100587
100587
100588
100589
100589
100590
100591
100592
100593
100594
100595
100596
100597
100597
100598
100599
100599
100600
100601
100602
100603
100604
100605
100606
100607
100608
100608
100609
100610
100611
100612
100613
100614
100615
100616
100617
100617
100618
100619
100620
100621
100622
100623
100624
100625
100626
100627
100627
100628
100629
100630
100631
100632
100633
100634
100635
100636
100637
100637
100638
100639
100640
100641
100642
100643
100644
100645
100646
100647
100647
100648
100649
100650
100651
100652
100653
100654
100655
100656
100657
100657
100658
100659
100660
100661
100662
100663
100664
100665
100666
100667
100667
100668
100669
100670
100671
100672
100673
100674
100675
100676
100677
100677
100678
100679
100680
100681
100682
100683
100684
100685
100686
100687
100687
100688
100689
100689
100690
100691
100692
100693
100694
100695
100696
100697
100697
100698
100699
100699
100700
100701
100702
100703
100704
100705
100706
100707
100708
100708
100709
100710
100711
100712
100713
100714
100715
100716
100717
100717
100718
100719
100720
100721
100722
100723
100724
100725
100726
100727
100727
100728
100729
100730
100731
100732
100733
100734
100735
100736
100737
100737
100738
100739
100740
100741
100742
100743
100744
100745
100746
100747
100747
100748
100749
100750
100751
100752
100753
100754
100755
100756
100757
100757
100758
100759
100760
100761
100762
100763
100764
100765
100766
100767
100767
100768
100769
100770
100771
100772
100773
100774
100775
100776
100777
100777
100778
100779
100779
100780
100781
100782
100783
100784
100785
100786
100787
100787
100788
100789
100789
100790
100791
100792
100793
100794
100795
100796
100797
100797
100798
100799
100799
100800
100801
100802
100803
100804
100805
100806
100807
100808
100808
100809
100810
100811
100812
100813
100814
100815
100816
100817
100817
100818
100819
100820
100821
100822
100823
100824
100825
100826
100827
100827
100828
100829
100830
100831
100832
100833
100834
100835
100836
100837
100837
100838
100839
100840
100841
100842
100843
100844
100845
100846
100847
100847
100848
100849
100850
100851
100852
100853
100854
100855
100856
100857
100857
100858
100859
100860
100861
100862
100863
100864
100865
100866
100867
100867
100868
100869
100870
100871
100872
100873
100874
100875
100876
100877
100877
100878
100879
100879
100880
100881
100882
100883
100884
100885
100886
100887
100887
100888
100889
100889
100890
100891
100892
100893
100894
100895
100896
100896
100897
100898
100899
100899
100900
100901
100902
100903
100904
100905
100906
100907
100908
100908
100909
100910
100911
100912
100913
100914
100915
100916
100917
100917
100918
100919
100920
100921
100922
100923
100924
100925
100926
100927
100927
100928
100929
100930
100931
100932
100933
100934
100935
100936
100937
100937
100938
100939
100939
100940
100941
100942
100943
100944
100945
100946
100947
100947
100948
100949
100949
100950
100951
100952
100953
100954
100955
100956
100957
100957
100958
100959
100959
100960
100961
100962
100963
100964
100965
100966
100967
100967
100968
100969
100969
100970
100971
100972
100973
100974
100975
100976
100977
100977
100978
100979
100979
100980
100981
100982
100983
100984
100985
100986
100987
100987
100988
100989
100989
100990
100991
100992
100993
100994
100995
100996
100996
100997
100998
100998
100999
100999
100100
100101
100102
100103
100104
100105
100106
100107
100108
100108
100109
100109
100110
100111
100112
100113
100114
100115
100116
100117
100117
100118
100119
100119
100120
100121
100122
100123
100124
100125
100126
100127
100127
100128
100129
100129
100130
100131
10
```

```

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("network", "UDP_IP"),
27                                   int(conf.get("network", "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("network", "UDP_IP"),
39                               int(conf.get("network", "UDP_PORT_GNSS")))
40
41     @staticmethod
42     def get_stream(ip, port):
43         """
44             Creates socket to scanner stream
45             :param ip: ip address 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         # IPv4 UDP
56         sock = socket.socket(type=socket.SOCK_DGRAM)
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     sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
65     # Allows broadcast UDP packets to be sent and received.
66
67     # Bind socket to local host and port
68     try:
69         sock.bind((ip, port))
70     except socket.error:
71         print('Bind failed.')
72
73     print('Socket connected!')
74
75     # now keep talking with the client
76     print('Listening on: ' + ip + ':' + str(port))
77
78     return sock

```

A.5 vdGNSSTime.py

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

4  """
5  @author: Florian Timm
6  @version: 2017.11.19
7  """

9  from datetime import datetime
10 from threading import Thread
11 import socket
12 import os
13 import serial

15 from vdInterface import VdInterface

18 class VdGNSSTime(Thread):

20     """ system time by gnss data """

22     def __init__(self, master):
23         """
24             Constructor
25             :param master: instance of VdAutoStart
26             :type master: VdAutoStart
27         """
28         Thread.__init__(self)
29         self.__tScanner = None
30         self.__tSerial = None
31         self.__master = master
32         self.__conf = master.conf
33         self.__time_corrected = False

35     def run(self):
36         """
37             starts threads for time detection
38         """
39         # get data from serial port
40         self.__tSerial = Thread(target=self.__get_gnss_time_from_serial())
41         self.__tSerial.start()

43         # get data from scanner
44         self.__tScanner = Thread(target=self.__get_gnss_time_from_scanner())
45         self.__tScanner.start()

47         self.__master.gnss_status = "Connecting..."

49     def __get_gnss_time_from_scanner(self):
50         """
51             gets data by scanner network stream
52             sock = VdInterface.get_gnss_stream(self.__conf)
53             sock.settimeout(1)
54             self.__master.gnss_status = "Wait for fix..."
55             while not self.__time_corrected:
56                 try:
57                     data = sock.recvfrom(2048)[0] # buffer size is 2048 bytes
58                     message = data[206:278].decode('utf-8', 'replace')
```

```

58             if self._get_gnss_time_from_string(message):
59                 break
60         except socket.timeout:
61             continue
62     # else:
63     #     print(message)
64     if data == 'QUIT':
65         break
66     sock.close()

68 # noinspection PyArgumentList
69 def __get_gnss_time_from_serial(self):
70     """ get data by serial port """
71     ser = None
72     try:
73         port = self._conf.get("serial", "GNSSport")
74         ser = serial.Serial(port, timeout=1)
75         self._master.gnss_status = "Wait for fix..."
76         while not self._time_corrected:
77             line = ser.readline()
78             message = line.decode('utf-8', 'replace')
79             if self._get_gnss_time_from_string(message):
80                 break
81             # else:
82             #     print(message)
83     except serial.SerialTimeoutException:
84         pass
85     except serial.serialutil.SerialException:
86         print("Could not open serial port!")
87     finally:
88         if ser is not None:
89             ser.close()

91 def __get_gnss_time_from_string(self, message):
92     if message[0:6] == "$GPRMC":
93         p = message.split(",")
94         if p[2] == "A":
95             print("GNSS-Fix")
96             timestamp = datetime.strptime(p[1] + "D" + p[9],
97                                           '%H%M%S.000%z')
98             self._set_system_time(timestamp)
99             self._time_corrected = True
100            self._master.gnss_status = "Got time!"
101            return True
102        return False

104 def __set_system_time(self, timestamp):
105     """
106     sets system time
107     :param timestamp: current timestamp
108     :type timestamp: datetime
109     :return:
110     :rtype:
111     """
112     os.system("timedatectl set-ntp 0")
113     os.system("timedatectl set-time \" +
114                 timestamp.strftime("%Y-%m-%d %H:%M:%S") + "\"")
115     os.system("timedatectl set-ntp 1")
116     self._master.set_gnss_status("System time set")

```

```

118     def stop(self):
119         """ stops all threads """
120         self.__master.gnss_status = "Stopped"
121         self.__time_corrected = True

```

A.6 vdHardware.py

```

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

4  """
5  @author: Florian Timm
6  @version: 2017.11.19
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         self.__master = master

43         # activate input pins
44         # recording start/stop
45         GPIO.setup(self.__taster1, GPIO.IN, pull_up_down=GPIO.PUD_UP)
46         # shutdown
47         GPIO.setup(self.__taster2, GPIO.IN, pull_up_down=GPIO.PUD_UP)

49         # activate outputs
50         for l in self.__led:

```

```

51         GPIO.setup(1, GPIO.OUT) # GPS-Fix
52         GPIO.output(1, GPIO.LOW)

54         self.__go_on = True

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

67         self.__timer_check_leds()

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

76     def __check_leds(self):
77         """ check LEDs """
78         self.__set_recording(self.__master.check_recording())
79         self.__set_receiving(self.__master.check_receiving())
80         self.__set_queue(self.__master.check_queue())

82     def __button1_pressed(self):
83         """ raised when button 1 is pressed """
84         time.sleep(0.1) # contact bounce

86         # > 2 seconds
87         wait = GPIO.wait_for_edge(self.__taster1, GPIO.RISING, timeout=1900)

89         if wait is None:
90             # no rising edge = pressed
91             if self.__master.go_on_buffer.value:
92                 self.__master.stop_recording()
93             else:
94                 self.__master.start_recording()

96     def __button2_pressed(self):
97         """ raised when button 1 is pressed """
98         time.sleep(0.1) # contact bounce

100        # > 2 seconds
101        wait = GPIO.wait_for_edge(self.__taster2, GPIO.RISING, timeout=1900)

103        if wait is None:
104            # no rising edge = pressed
105            self.__master.shutdown()

107     def __switch_led(self, led, yesno):
108         """
109             switch led

```

```

110         :param led: pin of led
111         :type led: int
112         :param yesno: True = on
113         :type yesno: bool
114         """
115         if yesno:
116             GPIO.output(self._led[led], GPIO.HIGH)
117         else:
118             GPIO.output(self._led[led], GPIO.LOW)
119
120     def __update_leds(self):
121         """ switch all LEDs to right status """
122         self.__switch_led(0, self.__receiving)
123         self.__switch_led(1, self.__queue)
124         self.__switch_led(2, self.__recording)
125
126     def __set_receiving(self, yesno):
127         """
128             set receiving variable and led
129             :param yesno: True = on
130             :type yesno: bool
131         """
132         if self.__receiving != yesno:
133             self.__receiving = yesno
134             self.__update_leds()
135
136     def __set_queue(self, yesno):
137         """
138             set queue variable and led
139             :param yesno: True = on
140             :type yesno: bool
141         """
142         if self.__queue != yesno:
143             self.__queue = yesno
144             self.__update_leds()
145
146     def __set_recording(self, yesno):
147         """
148             set recording variable and led
149             :param yesno: True = on
150             :type yesno: bool
151         """
152         if self.__recording != yesno:
153             self.__recording = yesno
154             self.__update_leds()
155
156     def stop(self):
157         """ stops thread """
158         self.__go_on = False
159         GPIO.cleanup()

```

A.7 vdFile.py

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """
5  @author: Florian Timm

```

```

6  @version: 2017.11.19
7  """
9  import datetime
10 from abc import abstractmethod, ABC
11 from vdPoint import VdPoint

14 class VdFile(ABC):
16     """ creates and fills an ascii-file with point data """
18     def __init__(self, conf, filename="", file_format="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 file_format: file suffix, default="txt"
26             :type file_format: str
27         """
28         self.__conf = conf
29         # create filename, if not set
30         if filename == "":
31             filename = self.__make_filename(file_format)
32         elif not filename.endswith("." + file_format):
33             filename += "." + file_format
34         # create file
35         self.__file = open(filename, 'a')
36         self.__writing_queue = []

38     def __get_writing_queue(self):
39         """
40             Returns points in queue
41             :return: points in queue
42             :rtype: VdPoint[]
43         """
44         return self.__writing_queue

46     writing_queue = property(__get_writing_queue)

48     def clear_writing_queue(self):
49         """
50             clears writing queue """
51         self.__writing_queue = []

52     def __make_filename(self, file_format):
53         """
54             generates a new filename from timestamp
55             :param file_format: file suffix
56             :type file_format: str
57             :return: string with date and suffix
58             :rtype: str
59         """
60         #
61         filename = self.__conf.get("file", "namePre")
62         filename += datetime.datetime.now().strftime(
63             self.__conf.get("file", "timeFormat"))
64         filename = "." + file_format

```

```

65         return filename
66
67     def _write2file(self, data):
68         """
69             writes ascii data to file
70             :param data: data to write
71             :type data: str
72         """
73         self._file.write(data)
74
75     def write_data(self, data):
76         """
77             adds data and writes it to file
78             :param data: ascii data to write
79             :type data: VdPoint []
80         """
81         self.add_dataset(data)
82         self.write()
83
84     def write(self):
85         """writes data to file """
86         txt = ""
87         for d in self.writing_queue:
88             if d.distance > 0.0:
89                 txt += self._format(d)
90         self._write2file(txt)
91         self.clear_writing_queue()
92
93     @abstractmethod
94     def _format(self, p):
95         raise NotImplementedError("not implemented, use child classes")
96
97     def add_point(self, p):
98         """
99             Adds a point to write queue
100            :param p: point
101            :type p: VdPoint
102        """
103         self._writing_queue.append(p)
104
105    def add_dataset(self, dataset):
106        """
107            adds multiple points to write queue
108            :param dataset: multiple points
109            :type dataset: VdPoint []
110        """
111        self._writing_queue.extend(dataset)
112
113    def read_from_txt_file(self, filename, write=False):
114        """
115            Parses data from txt file
116            :param filename: path and filename of txt file
117            :type filename: str
118            :param write: write data to new file while reading txt
119            :type write: bool
120        """
121        txt = open(filename)
122
123        for no, line in enumerate(txt):

```

```

124         try:
125             p = VdPoint.parse_string(self.__conf, line)
126             self.writing_queue.append(p)
127             print("Line {0} was parsed".format(no + 1))
128         except ValueError as e:
129             print("Error in line {0}: {1}".format(no + 1, e))

131         if write and len(self.writing_queue) > 50000:
132             self.write()
133         if write:
134             self.write()

136     def close(self):
137         """ close file """
138         self.__file.close()

141 class VdObjFile(VdFile):

143     """ creates and fills an obj-file """

145     def __init__(self, conf, filename=""):
146         """
147             Creates a new obj-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, "obj")

155     def _format(self, p):
156         """
157             Formats point for OBJ
158             :param p: VdPoint
159             :type p: VdPoint
160             :return: obj point string
161             :rtype: str
162         """
163         x, y, z = p.get_yxz()
164         format_string = 'v {:.3f} {:.3f} {:.3f}\n'
165         return format_string.format(x, y, z)

168 class VdTtxtFile(VdFile):

170     """ creates and fills an txt-file """

172     def __init__(self, conf, filename=""):
173         """
174             Creates a new txt-file
175             :param conf: configuration file
176             :type conf: configparser.ConfigParser
177             :param filename: name and path to new file
178             :type filename: str
179         """
180         VdFile.__init__(self, conf, filename)

182     def _format(self, p):

```

```

183     """
184     Formats point for TXT
185     :param p: VdPoint
186     :type p: VdPoint
187     :return: txt point string
188     :rtype: str
189     """
190     format_string = '{:012.1f}\t{:07.3f}\t{: 03.0f}\t{:06.3f}\t{:03.0f}\n'
191     return format_string.format(p.time,
192                                 p.azimuth,
193                                 p.vertical,
194                                 p.distance,
195                                 p.reflection)

```

A.8 vdDataset.py

```

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

4  """
5  @author: Florian Timm
6  @version: 2017.11.19
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.__vertical_angle = json.loads(
30             self.__conf.get("device", "verticalAngle"))
31         self.__offset = json.loads(self.__conf.get("device", "offset"))
32         self.__data = []

34     def get_azimuth(self, block):
35         """
36             gets azimuth of a data block
37             :param block: number of data block
38             :type block: int
39             :return: azimuth
40             :rtype: float
41         """

```

```

43     offset = self._offset[block]
44     # change byte order
45     azi = ord(self._dataset[offset + 2:offset + 3]) + \
46           (ord(self._dataset[offset + 3:offset + 4]) << 8)
47     azi /= 100.0
48     # print(azi)
49     return azi

51     def get_time(self):
52         """
53             gets timestamp of dataset
54             :return: timestamp of dataset
55             :rtype: int
56         """
57
58         time = ord(self._dataset[1200:1201]) + \
59               (ord(self._dataset[1201:1202]) << 8) + \
60               (ord(self._dataset[1202:1203]) << 16) + \
61               (ord(self._dataset[1203:1204]) << 24)
62         # print(time)
63         return time

64     def is_dual_return(self):
65         """
66             checks whether dual return is activated
67             :return: dual return active?
68             :rtype: bool
69         """
70
71
72         mode = ord(self._dataset[1204:1205])
73         if mode == 57:
74             return True
75         else:
76             return False

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
85         # create empty lists
86         azimuths = [0.] * 24
87         rotation = [0.] * 12
88
89         # read existing azimuth values
90         for j in range(0, 24, 2):
91             a = self.get_azimuth(j // 2)
92             azimuths[j] = a
93
94         #: rotation angle
95         d = 0
96
97         # DualReturn active?
98         if self.is_dual_return():
99             for j in range(0, 19, 4):
100                 d2 = azimuths[j + 4] - azimuths[j]
101                 if d2 < 0:

```

```

102             d2 += 360.0
103             d = d2 / 2.0
104             a = azimuths[j] + d
105             azimuths[j + 1] = a
106             azimuths[j + 3] = a
107             rotation[j // 2] = d
108             rotation[j // 2 + 1] = d

110         rotation[10] = d
111         azimuths[21] = azimuths[20] + d

113     # Strongest / Last-Return
114     else:
115         for j in range(0, 22, 2):
116             d2 = azimuths[j + 2] - azimuths[j]
117             if d2 < 0:
118                 d2 += 360.0
119             d = d2 / 2.0
120             a = azimuths[j] + d
121             azimuths[j + 1] = a
122             rotation[j // 2] = d

124     # last rotation angle from angle before
125     rotation[11] = d
126     azimuths[23] = azimuths[22] + d

128     # >360 -> -360
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     # timestamp from dataset
141     time = self.get_time()

143     azimuth, rotation = self.get_azimuths()

145     dual_return = self.is_dual_return()
146     t_between_laser = float(self._conf.get("device", "tInterBeams"))
147     t_recharge = float(self._conf.get("device", "tRecharge"))
148     part_rotation = float(self._conf.get("device", "ratioRotation"))

150     # data package has 12 blocks with 32 measurements
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                 # get distance
157                 dist = ord(self._dataset[4 + offset:5 + offset]) \
158                     + (ord(self._dataset[5 + offset:6 + offset]) << 8)
159                 dist /= 500.0

```

```

161             reflection = ord(self._dataset[6 + offset:7 + offset])
162
163             # offset for next loop
164             offset += 3
165
166             # interpolate azimuth
167             a = azi_block + rotation[i] * k * part_rotation
168
169             # a += time / 1000000 * 0.1
170
171             # create point
172             p = VdPoint(
173                 self._conf, round(
174                     time, 1), a, self._vertical_angle[k],
175                     dist, reflection)
176             self._data.append(p)
177             time += t_between_laser
178
179             if dual_return and j == 0:
180                 time -= t_between_laser * 16
181             else:
182                 time += t_recharge
183
184     def get_data(self):
185         """
186             get all point data
187             :return: list of VdPoints
188             :rtype: list
189         """
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: 2017.11.19
7  """
8
9  import math
10
11
12 class VdPoint(object):
13
14     """ Represents a point """
15
16     _dRho = math.pi / 180.0
17
18     def __init__(self, conf, time, azimuth, vertical, distance, reflection):
19         """
20             Constructor
21             :param conf: config file
22             :type conf: configparser.ConfigParser
23             :param time: recording time in microseconds
24             :type time: float
25             :param azimuth: Azimuth direction in degrees

```

```

26         :type azimuth: float
27         :param vertical: Vertical angle in degrees
28         :type vertical: float
29         :param distance: distance in metres
30         :type distance: float
31         :param reflection: reflection 0-255
32         :type reflection: int
33         """
34
35         self.__time = time
36         self.__azimuth = azimuth
37         self.__vertical = vertical
38         self.__reflection = reflection
39         self.__distance = distance
40         self.__conf = conf
41
42     @staticmethod
43     def parse_string(conf, line):
44         """
45             Parses string to VdPoint
46             :param conf: config file
47             :type conf: configparser.ConfigParser
48             :param line: Point as TXT
49             :type line: str
50             :return: Point
51             :rtype: VdPoint
52             :raise ValueError: malformed string
53             """
54
55         d = line.split()
56         if len(d) > 4:
57             time = float(d[0])
58             azimuth = float(d[1])
59             vertical = float(d[2])
60             distance = float(d[3])
61             reflection = int(d[4])
62             return VdPoint(conf, time, azimuth,
63                            vertical, distance, reflection)
63         else:
64             raise ValueError('Malformed string')
65
66     def __deg2rad(self, degree):
67         """
68             converts degree to radians
69             :param degree: degrees
70             :type degree: float
71             :return: radians
72             :rtype: float
73             """
74
75     def get_yxz(self):
76         """
77             Gets local coordinates
78             :return: local coordinates x, y, z in metres
79             :rtype: float, float, float
80             """
81
82         beam_center = float(self.__conf.get("device", "beamCenter"))
83
84         # slope distance to beam center
85         d = self.distance - beam_center

```

```

86         # vertical angle in radians
87         v = self.vertical.radians

89         # azimuth in radians
90         a = self.azimuth.radians

92         # horizontal distance
93         s = d * math.cos(v) + beam_center

95         x = s * math.sin(a)
96         y = s * math.cos(a)
97         z = d * math.sin(v)

99         return x, y, z

101     def __get_time(self):
102         """
103             Gets recording time
104             :return: recording time in microseconds
105             :rtype: float
106         """
107         return self.__time

109     def __get_azimuth(self):
110         """
111             Gets azimuth direction
112             :return: azimuth direction in degrees
113             :rtype: float
114         """
115         return self.__azimuth

117     def __get_azimuth_radians(self):
118         """
119             Gets azimuth in radians
120             :return: azimuth direction in radians
121             :rtype: float
122         """
123         return self.__deg2rad(self.azimuth)

125     def __get_vertical(self):
126         """
127             Gets vertical angle in degrees
128             :return: vertical angle in degrees
129             :rtype: float
130         """
131         return self.__vertical

133     def __get_vertical_radians(self):
134         """
135             Gets vertical angle in radians
136             :return: vertical angle in radians
137             :rtype: float
138         """
139         return self.__deg2rad(self.vertical)

141     def __get_reflection(self):
142         """
143             Gets reflection

```

```

144         :return: reflection between 0 and 255
145         :rtype: int
146         """
147         return self.__reflection
148
149     def __get_distance(self):
150         """
151         Gets distance
152         :return: distance in metres
153         :rtype: float
154         """
155         return self.__distance
156
157     # properties
158     time = property(__get_time)
159     azimuth = property(__get_azimuth)
160     azimuth_radians = property(__get_azimuth_radians)
161     vertical = property(__get_vertical)
162     vertical_radians = property(__get_vertical_radians)
163     reflection = property(__get_reflection)
164     distance = property(__get_distance)

```

A.10 config.ini

```

1 [network]
2 UDP_IP = 0.0.0.0
3 UDP_PORT_DATA = 2368
4 UDP_PORT_GNSS = 8308
5
6 [serial]
7 # Serieller Port
8 #Raspberry
9 GNSSport = /dev/ttyAMA0
10 #Ubuntu
11 #GNSSport = /dev/ttyUSB0
12
13 [functions]
14 # Zeitgleiche Transformation zu txt aktivieren
15 activateTransformer = True
16
17 # GNSS-Zeit verwenden
18 use_gnss_time = True
19
20
21 [file]
22 # Binaere Dateien nach deren Transformation loeschen
23 deleteBin = True
24
25 #Takt zur Speicherung Buffer -> HDD
26 takt = 5
27
28 # Format der Zeit am Dateinamen
29 dateFormat = %%Y-%%m-%%dT%%H:%%M:%%S
30
31 # Dateipraefix der zu speichernden Datei
32 namePre = data
33

```

```

36 [device]
37 # Zeit zwischen den Messungen der Einzelstrahlen
38 tInterBeams = 2.304

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

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

46 # Anteil der Zeit zwischen Einzellasern an Wiederholungszeit,
47 # fuer Interpolation des Horizontalwinkels
48 ratioRotation = 0.041666666666666664
49 #tZwischenStrahl / tRepeat

51 # Zeit nach letztem Strahl bis zum naechsten
52 tRecharge = 20.736
53 #tRepeat - 15 * tZwischenStrahl

55 # Abstand des Strahlenzentrums von der Drehachse
56 beamCenter = 0.04191

58 valuesPerDataset = 384
59 #12*32

61 # Bytes pro Messdatenblock
62 offsetBlock = 100
63 # 3 * 32 + 4

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

```

A.11 convTxt2Obj.py

```

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

4 """
5 @author: Florian Timm
6 @version: 2017.11.19
7 """
8 from vdFile import VdObjFile
9 import configparser

11 fileName = "Beispieldateien/test.txt"

13 conf = configparser.ConfigParser()
14 conf.read("config.ini")

16 f = VdObjFile(conf, fileName)
17 f.read_from_txt_file(fileName, True)

```

A.12 convBin2Obj.py

```
1 #!/usr/bin/env python
```

```

2  # -*- coding: utf-8 -*-
3
4  """
5  @author: Florian Timm
6  @version: 2017.11.19
7  """
8
9  import os
10 from vdDataset import VdDataset
11 from vdFile import VdObjFile
12 from glob import glob
13 import configparser
14
15 # load config file
16 conf = configparser.ConfigParser()
17 conf.read("config.ini")
18
19 fs = glob(
20     "/ssd/daten/ThesisMessung/data2017-11-16T14:06:31_SicherungBin/*.bin")
21
22 if len(fs) > 0:
23     folder = os.path.dirname(fs[0])
24     obj_file = VdObjFile(
25         conf,
26         folder + "/file")
27
28     for filename in fs:
29         print(filename)
30
31     bin_file = open(filename, "rb")
32
33     # Calculate number of datasets
34     fileSize = os.path.getsize(bin_file.name)
35     cntDatasets = int(fileSize / 1206)
36
37     for i in range(cntDatasets):
38         vdData = VdDataset(conf, bin_file.read(1206))
39         vdData.convert_data()
40         obj_file.write_data(vdData.get_data())
41     bin_file.close()
42     obj_file.close()

```

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