Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“
(UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

„Genauigkeitsverbesserte Georeferenzierung von UAS-Bildflugmaterial durch Nutzung von SAPOS-RINEX-Daten im Postprocessing“

vorgelegt von

Dipl.-Ing.(FH) Benedikt Stratmann
105341, UNIGIS MSc Jahrgang 2019

Betreuer:
Dr. Christian Neuwirth

Zur Erlangung des Grades
„Master of Science – MSc”

Büren-Steinhausen, 04.09.2022

Abstract

In this master thesis, procedural steps are developed and explained which - using low-cost hardware as well as low-cost and freely available software - enable accuracy-enhanced georeferencing of UAS video imagery (subsequently processed into 3D point clouds and orthophotos) by applying the post-processed kinematic method (PPK) (using RINEX-GNSS-reference station data provided free of charge). The result shows, after comparison with reference data, that with the developed method steps average position accuracies of about 0.4 m are achieved, which are on the accuracy level of comparable investigations and prove the practical suitability of the developed method.

Zusammenfassung

Im Rahmen dieser Masterthesis werden Verfahrensschritte erarbeitet und erläutert, die - unter Einbezug kostengünstiger Hardware sowie kostengünstiger und frei verfügbarer Software - eine genauigkeitsverbesserte Georeferenzierung von UAS-Videoaufnahmen (weiterverarbeitet zu 3D-Punktwolken und Orthofotos) durch Anwendung des Post-Processed-Kinematic-Verfahrens (PPK) (unter Verwendung von kostenlos bereitgestellten RINEX-GNSS-Referenzstationsdaten) ermöglichen.

Im Ergebnis zeigt sich, nach Abgleich mit Referenzdaten, dass mit den erarbeiteten Verfahrensschritten mittlere Positionsgenauigkeiten von ca. 0,4 m erreicht werden, die damit auf dem Genauigkeitsniveau vergleichbarer Untersuchungen liegen und die Praxistauglichkeit des erarbeiteten Verfahrens belegen.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe. Textpassagen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Büren-Steinhausen, 4. September 2022

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ALS	airborne laserscanner
CAD	engl.: computer-aided design
CNC	engl.: Computerized Numerical Control
D-GNSS	differenzielles GNSS
GCS	engl.: ground control station
GLONASS	russ.: Globalnaja nawigazionnaja sputnikowaja sistema
GNSS	globales Navigationssatellitensystem
GPIO	engl.: General Purpose Input/Output
GPS	engl.: Global Positioning System
GSD	engl.: ground sample distance
NC	engl.: numerical code
ODM	engl.: Open Drone Map
PPK	engl.: post processed kinematic
PWM	engl.: Pulse-width modulation
RINEX	engl.: Receiver Independent Exchange Format
RTK	engl.: reat time kinematic
SAPOS	Satellitenpositionierungsdienst der deutschen Landesvermessung
SfM	Structure-from-Motion
UART	engl.: universal asynchronous receiver-transmitter
UAS	unmanned aircraft system
UAV	unmanned aerial vehicle
USB	engl.: Universal Serial Bus
VRS	virtuelle Referenzstation
WP	engl.: waypoint

1 Einleitung

1.1 Fachliche Einordnung und Motivation

Mittels UAS[1] (unmanned aircraft system) ist die Erfassung von optischem Bildmaterial zur photogrammetrischen Weiterverarbeitung zu zeitlich und räumlich hochaufgelösten 3D-Messpunkten und darauf basierenden Orthofotos mit geringem Ressourceneinsatz möglich (Laliberte, et al. 2011). Direkte oder indirekte Georeferenzierung[2] setzt diese Produkte zu einem erdgebunden Koordinatensystem in Bezug.

Im Fall der direkten Georeferenzierung bestimmen primär die während des Bildflugs vom GNSS[3]-Empfänger berechneten und als Aufnahmeposition der Bilder verwendeten Koordinaten die geodätische Positionsgenauigkeit der 3D-Messpunkte und deren Derivaten. Die direkte Georeferenzierung bietet sich im Fall einer UAS-Befliegung an, da einerseits die aufwändige Einmessung von Passpunkten (GCP[4]) am Boden entfällt und andererseits ohnehin GNSS-Positionen im Zuge der UAS-Flugregelung gemessen werden.

An UAS im Low-Cost-Bereich verbaute GNSS-Empfänger berechnen, ohne Einbezug von Korrekturwerten, Positionen mit einer Ungenauigkeit von einigen Metern. (Im Fall des Modells u-blox NEO-M8T beispielsweise 2,5 bis 4 m (vgl. Anhang IV 19)). Eine erhebliche Verbesserung der Positionsberechnung, ergibt sich durch den Einsatz der RTK[5]-Technologie (s. 2.3.1), für den das UAS aber um zusätzliche Geräte zu erweitern ist und bestimmte Voraussetzungen erfüllt sein müssen (z. B. Mobilfunkanbindung).

Alternativ besteht die Möglichkeit GNSS-Messwerte im Nachhinein, durch das Post-Processed-Kinematic-Verfahren (PPK, s. 2.3.2) zu korrigieren und damit deren

---

[1] engl: unmanned aircraft system

[2] Die im Flug aufgezeichneten GNSS-Daten erlauben die direkte Inbezugsetzung zu einem geodätischen Koordinatensystem, während dies bei einer indirekten Georeferenzierung über den Umweg der Passpunkte (GCP=ground control points) erfolgt (Kraus 2004).

[3] engl: global navigation satellite system

[4] engl: ground control points

[5] engl: Real Time Kinematic

Positionsgenauigkeiten und die der georeferenzierten photogrammetrischen Erzeugnisse zu verbessern. Die dazu verwendeten Korrekturdaten werden im RINEX-Format im Rahmen des Dienstes SAPOS[6] bereitgestellt.

Die konkrete Ausgestaltung der vorbereitenden Verfahrensschritte hängt dabei von verschieden Faktoren, wie der Formatierungsart der GNSS-Rohdaten, der Synchronisation von Videobild und GNSS-Messwert oder wirtschaftlichen Überlegungen, wie dem einkalkulierten Budget für Ausrüstung, Software und Zeit ab.

1.2 Aktueller Stand der Forschung

Es gibt zahlreiche Untersuchungen, die die Verbesserung der Positionsgenauigkeit aufgrund der Anwendung des PPK-Verfahrens im Rahmen der direkten Georeferenzierung quantitativ untersuchen. Dabei werden die erzielten Ergebnisse auch mit den erzielbaren Positionsgenauigkeiten infolge einer indirekten, passpunktgestützten Georeferenzierung verglichen. Herausgestellt seien dazu die aktuellen und repräsentativen Arbeiten von Zhang, et al. (2019), Koller (2020), Dinkov und Kitev (2020), Padró, et al. (2019), sowie Iizuka, et al. (2021).

Tabelle 1 zeigt exemplarisch, welche Ergebnisse im Rahmen der Untersuchungen von Padró, et al. (2019) und Iizuka, et al. (2021) erzielt wurden.

Tabelle 1: Erzielte Positionsgenauigkeiten (hier: Lagegenauigkeit) von UAS-Bildmaterial bei Anwendung unterschiedlicher Georeferenzierungsmethoden in anderen Untersuchungen.

	Iizuka, et al. 2021	**Padró, et al. 2019**
	ΔXY (m	ΔXY (m)
Original	1,17	1,84
PPK	0,49	0,44
GCP	0,11	0,04

Es wurde festgestellt, dass die Positionsgenauigkeit bei direkter Georeferenzierung ohne nachprozessierte GNSS-Daten im Bereich von 1,17 bis 1,84 m und damit im Bereich der Ungenauigkeit der von 1-Frequenz-GNSS-Empfängern berechneten

[6] Satellitenpositionierungsdienst der deutschen Landesvermessung

Positionswerte liegt. Bei Georeferenzierung auf Basis mittels PPK nachkorrigierter GNSS-Daten konnten dagegen Genauigkeiten von 0,44 bis 0,49 m erreicht werden.

Jedoch zeigen die in Tabelle 1 wiedergegebenen Werte sowie die Ergebnisse weiterer Untersuchungen durchweg, dass bei indirekter Georeferenzierung mittels Passpunkten (GCP) noch weitaus höhere Genauigkeiten erzielt werden.

In den oben genannten Arbeiten wird das optische Bildmaterial durchweg im Form von Einzelfotos erfasst und verarbeitet. Im Gegensatz zu Videoaufnahmen ist dabei aus technischer Sicht eine einfachere Synchronisation mit den GNSS-Daten möglich. Die Bewältigung der aufwändigeren Videobild-Georeferenzierung wird in Nebiker und Eugster (2009) und Eugster, et al. (2012) behandelt. Jedoch, neben der direkten Georeferenzierung ohne PPK-Anwendung, im Zusammenhang mit einem indirekten Georeferenzierungsansatz (Koregistration auf Basis eines 3D-City-Modells). Bei diesen Untersuchungen lag die GNSS-Positionsgenauigkeit unkorrigiert bei 3-4 m und nachkorrigiert bei < 1 m.

Es fehlen derzeit integrative Untersuchungen zur Vorbereitung und Durchführung von Video-Bildflügen sowie der Nachbereitung der GNSS-Daten mittels PPK mit kostengünstiger Hardware sowie kostengünstiger und frei zugänglicher, möglichst quelloffener, Software.

1.3 Untersuchungsziel

Ziel der Masterarbeit ist die detaillierte Erarbeitung der erforderlichen Verfahrensschritte, die nötig sind um unter Nutzung von kostengünstiger Hardware sowie frei zugänglichen Mitteln, möglichst quelloffener Software, 3D-Punktwolken und Orthofotos mittels SfM anzufertigen.

Als Ergebnis ist festzustellen, ob die damit erzielten Ergebnisse den Anforderungen bzw. den in vorherigen Untersuchungen erzielten Resultaten entsprechen.

1.4 Aufbau der Masterarbeit

In Kapitel 2 werden zum Verständnis der Verfahrensschritte notwendige Grundlagen vorgestellt und erläutert. In Kapitel 3 werden die ausgearbeiteten Prozessschritte

detailliert vorgestellt und erläutert. In Kapitel 4 erfolgt eine Darstellung der Ergebnisse. Kapitel 5 schließt die Masterthesis mit einem Fazit und Ausblick ab.

Der ausgearbeitete Programmcode sowie Beispiel-Dateien und weitere Zusatzinformationen sind im Anhang aufgeführt. Zusätzlich ist der Programmcode in folgendem github-Repository im Internet hinterlegt:

https://github.com/BS-Code-01/UAS-Code-Rep

2 Grundlagen

2.1 Ausrüstung

Zur Ausrüstung gehört eine Kamera mit der an mehreren Standorten aus der Vogelperspektive (mit lotrecht zum Massenschwerpunkt der Erde ausgerichtetem Objektiv) das abzubildende Gelände als optisches Bildmaterial erfasst wird. Ein UAV[7] trägt die Kamera und vollzieht die Standortwechsel der Kamera.

Die in der Fernerkundung am häufigsten eingesetzten UAV-Typen sind Starr- und Drehflügler[8].

Das UAV ist mit Sensoren (Magnetometer[9], Barometer, GNSS-Empfänger, Beschleunigungssensor), einer Regelungs- und Steuereinheit (Flugregler) sowie Aktoren (Motorregler und Motoren) ausgestattet. Die Sensoren liefern dem Flugregler die Messgrößen, die er zur Bestimmung der Ist-Position und der Ist-Fluglage benötigt[10]. Der Flugregler gleicht in hoher Taktfrequenz (Mikrosekundenbereich) die Ist- und Soll-Werte ab und berechnet auf Basis der Wertabweichungen an die Aktoren gerichtete Steuersignale zur Erreichung der Soll-Werte.

2.2 Flugvorbereitung und -durchführung

Das abzubildende Gelände wird in parallel und antiparallel verlaufenden Flugachsen überflogen. Jede Flugachse wird als Strecke über 2 Koordinaten (Wegpunkte[11], kurz: WP) festgelegt. Auf eine lange Flugachse (z. B. WP 1 zu WP 2) folgt eine um 90° abgewinkelte kurze Flugachse (z. B. WP 2 zu WP3), darauf eine um 90° abgewinkelte, der ersten langen Flugachse antiparallel verlaufende, lange Flugachse (z. B. WP 3 zu WP4) und so weiter. Die Abfolge der Wegpunkte bildet eine im Speicher des Flugreglers abzuspeichernde Flugmission, die als Kommandosequenz (Programm)

---

[7] engl: unmanned aerial vehicle

[8] Weitere UAS-Typen: Ballone, Zeppeline, Drachen.

[9] Kompass

[10] Roll- und Nickwinkel (Beschleunigungssensor), Azimuthwinkel/Gierwinkel (0°-360°, Magnetometer), Höhe über Grund/NN (Barometer, GNSS), Ortskoordinaten (GNSS)

[11] engl: waypoint. X-, Y-, Z-Koordinaten (WGS84).

vom UAV ausgeführt wird. Das UAV fliegt die Flugroute automatisch-autonom ab, d. h. ohne Steuerung durch den Operator.

2.3 GNSS

Einerseits zur UAS-Navigation und andererseits zur Georeferenzierung der erfassten Bilder ist die Bereitstellung möglichst genauer und zeitlich hoch aufgelöster Positionsinformationen aus GNSS-Messungen erforderlich. Das UAV muss daher mit einem GNSS-Empfänger ausgestattet sein.

Bei GNSS-Empfängern im Zusammenhang mit UAS handelt es sich in der Regel um 1-Band-Empfänger: Von den 2 Hauptfrequenzbändern[12], in denen von jedem GNSS (z. B. GPS[13] oder GLONASS[14]) elektromagnetische Wellen als Signalträger emittiert werden, werden empfängerseitig nur Signale aus dem niederen Frequenzband (L1) verarbeitet. Ohne Signale aus dem zusätzlichem L2-Band sind bestimmte wellenlängenbasierte Korrekturrechnungen nicht unmittelbar im Empfänger durchführbar und die abschließend berechnete Positionslösung erreicht nicht die maximale Genauigkeit. Die Ist-Position des UAV kann daher bis zu 4 m[15] von der Soll-Position abweichen, obwohl die Soll-Ist-Wertdifferenz im Flugregler richtig berechnet wurde, jedoch auf ungenauen Messergebnissen beruht. Beispielsweise wird ein über X-, Y- und Z-Koordinaten (WGS84) bestimmter WP angeflogen: Laut GNSS-Messung ist der WP exakt erreicht, das UAV befindet sich aber in Wahrheit 1,5 m davon entfernt. Ein Bild, welches an dieser Position aufgenommen wird und über die Positionslösung des UAV-GNSS-Empfängers georeferenziert wird, weist eine Abweichung von 1,5 m in den Positionsdaten auf. Es würden sich durch Fehlerfortpflanzung Positionsungenauigkeiten im Verlauf der photogrammetrischen Weiterverarbeitung ergeben. Die Vermessungskunde unterscheidet grobe, systematische und zufällige

[12] Level 1 (L1) und Level 2 (L2)

[13] Global Positioning System

[14] russ.: Global'naya Navigatsionnaya Sputnikovaya Sistema

[15] Vgl. Datenblatt vom Modell Ublox Neo-M8T in Anhang IV 19.

Fehler (Kahmen 2006). Bei den hier vorgebrachten Fehleranteilen in der Positionsmessung handelt es sich um systematische Fehler[16].

Fehlerhafte bzw. ungenaue Positionierungsergebnisse ergeben sich durch Abweichungen zu den vom GNSS-Kontrollsegment vorhergesagten Satelliten-Orbits, Laufzeitschwankungen des GNSS-Signals aufgrund atmosphärischer Einflüsse, Asynchronität zwischen der Satelliten- und Empfängeruhr und weitere aber weniger starke Einflüsse (z. B. Empfängerrauschen). Die Kombination der L2-Band-Signale mit den L1-Band-Signalen vereinfacht die mathematische Modellierung und die darauf basierende weitgehende rechnerische Eliminierung der Fehlereinflüsse unmittelbar im Empfänger. Da das L2-Band in den meisten UAV-GNSS-Empfängern nicht verarbeitet wird, scheidet eine derartige Fehlereliminierung für das hier beschriebene Vorhaben aus.

2.3.1 RTK

Um trotzdem ein um die Fehleranteile bereinigtes Positionsergebnis zu erzielen sind die Prinzipien des differenziellen GNSS (D-GNSS), im Speziellen der Real-Time-Kinematik (RTK), bei der zusätzlich zur Codesignalauswertung die Trägerphasenmehrdeutigkeiten aufgelöst werden (Bauer 2018), anzuwenden (s. Abbildung 1). Für die Positionsberechnung mittels D-GNSS sind mindestens 2 GNSS-Empfänger erforderlich. Empfänger Nr. 1, der UAV-GNSS-Empfänger, dessen Positionierungsergebnis zu verbessern ist, wird als Rover (engl.) und Empfänger Nr. 2 als Base (engl.) bzw. Referenzstation bezeichnet, wobei Empfänger Nr. 2 ortsfest und seine Position mit hoher Genauigkeit bekannt ist. Beide Empfänger empfangen synchron die gleichen GNSS-Signale (Beobachtungen) und berechnen daraus Messergebnisse[17], die am Ende zu Positionsergebnissen weiterberechnet werden. Auf Seite der Referenzstation wird ein berechnetes Positionsergebnis mit der bekannten Position der Referenzstation verglichen und damit Messfehler und deren Ausmaß

[16] Laut (Kahmen 2006) handelt es sich bei systematischen Fehlern um solche, die durch Eichung der Messinstrumente, Wahl geeigneter Messverfahren und rechnerisches Berücksichtigen der einsinnigen Einflüsse zum größten Teil eliminiert werden. Diese Kriterien sind im Zusammenhang mit der GNSS-Messwerte-Korrektur gegeben.

[17] Pseudorange, Trägerphasenverschiebung, Doppler-Frequenzverschiebung, Dilution of Precision, Health usw.

erkannt. Es wird dann (für jeden Fehlereinfluss[18] getrennt) mittels iterativer rechnerischer Annäherung versucht die Fehlereinflüsse zu berechnen und auf Basis der Ergebnisse wiederum die Genauigkeit der abschließenden Positionierung zu maximieren. Sobald die Fehlereinflüsse bekannt sind, können sie Empfänger Nr.1 als Korrekturwerte zur Verfügung gestellt werden.

Die Korrekturdaten werden in Quasi-Echtzeit via RTCM-Protokoll über das Fernmeldenetz, dabei teilweise per Mobilfunk über die Luftschnittstelle, übertragen. Die Korrekturdatenbereitstellung in Quasi-Echtzeit erfordert einen relativ hohen technischen Aufwand[19], ist nicht immer und zuverlässig verfügbar[20] sowie mit Kosten[21] verbunden.

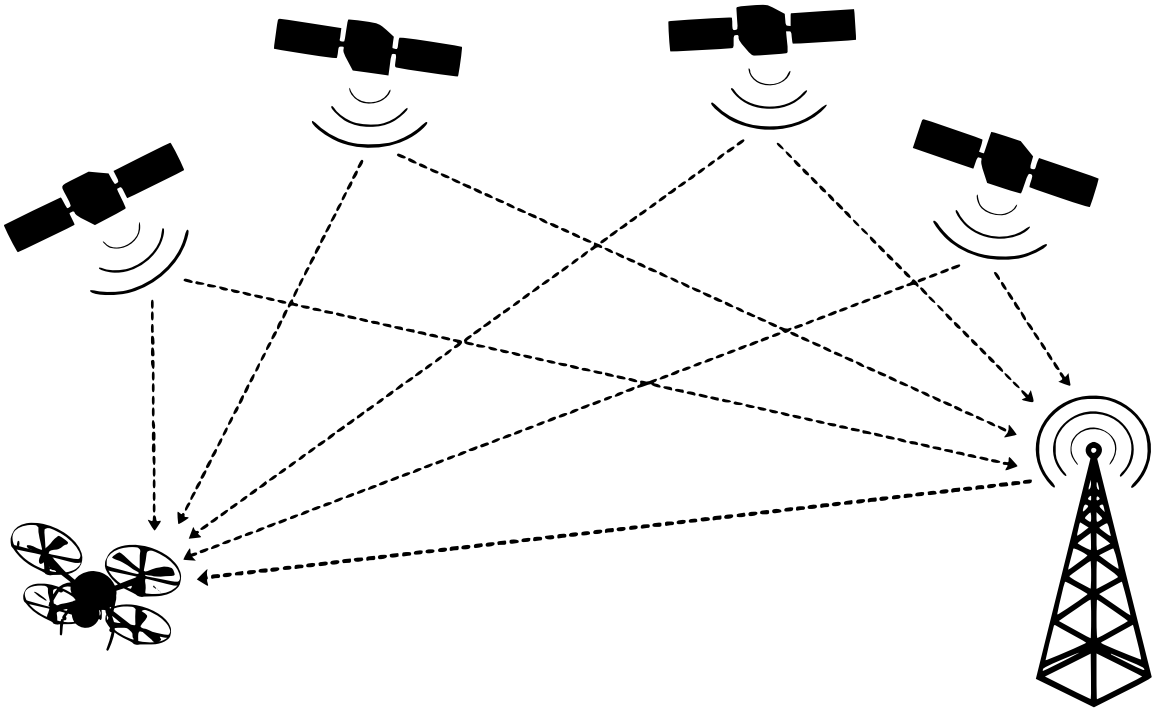


Abbildung 1: Prinzip des differenziellen GNSS bzw. der RTK.

[18] Orbitabweichung, Laufzeitverlängerung durch ionosphärische, troposphärische oder multipath-Effekte, Uhrzeitfehler

[19] Beschaffung und Installation entsprechender Schnittstellenkomponenten am UAV sowie Gewichtszunahme des UAS durch selbige.

[20] Bei unzureichender Mobilfunk-Netzabdeckung.

[21] Mobilfunk-Verbindungskosten

2.3.2 PPK

Eine Genauigkeitsverbesserung der Positionierungsergebnisse des Rovers ist nicht nur in Quasi-Echtzeit, sondern auch im Nachhinein möglich. Die vom Rover an den Flugregler für die UAV-Navigation ausgegebenen Positionierungsergebnisse erreichen dabei weiterhin nur die Genauigkeiten ohne Einrechnung der Korrekturwerte der Referenzstation. Die programmierte Flugroute wird im automatisch-autonomen Flug unter diesen Bedingungen mit entsprechender Ungenauigkeit (WP werden um bis zu 4 m verfehlt) eingehalten. Aufgrund der im Nachhinein korrigierten Positionierungsergebnisse stehen jedoch die Aufenthaltspositionen des UAV und zugleich der Kamera zu einem gegebenen Zeitpunkt der Flugmission in verbesserter Genauigkeit zur Verfügung. Bilder bzw. deren Aufnahmeposition können somit mit erhöhter Genauigkeit verortet werden.

Das Post-Processed-Kinmatic-Verfahren (PPK) entspricht mathematisch dem RTK-Verfahren und strebt wie diese, zur Feinpositionierung, die Auflösung der Trägerphasenmehrdeutigkeiten an (Bauer 2018). Jedoch verläuft diese Berechnung nicht in Quasi-Echtzeit (vgl. Abbildung 2), sondern im Nachhinein auf Basis der Messgrößen, die in den RINEX-Dateien (Log-Dateien der Empfänger) aufgezeichnet sind (vgl. Abbildung 3).

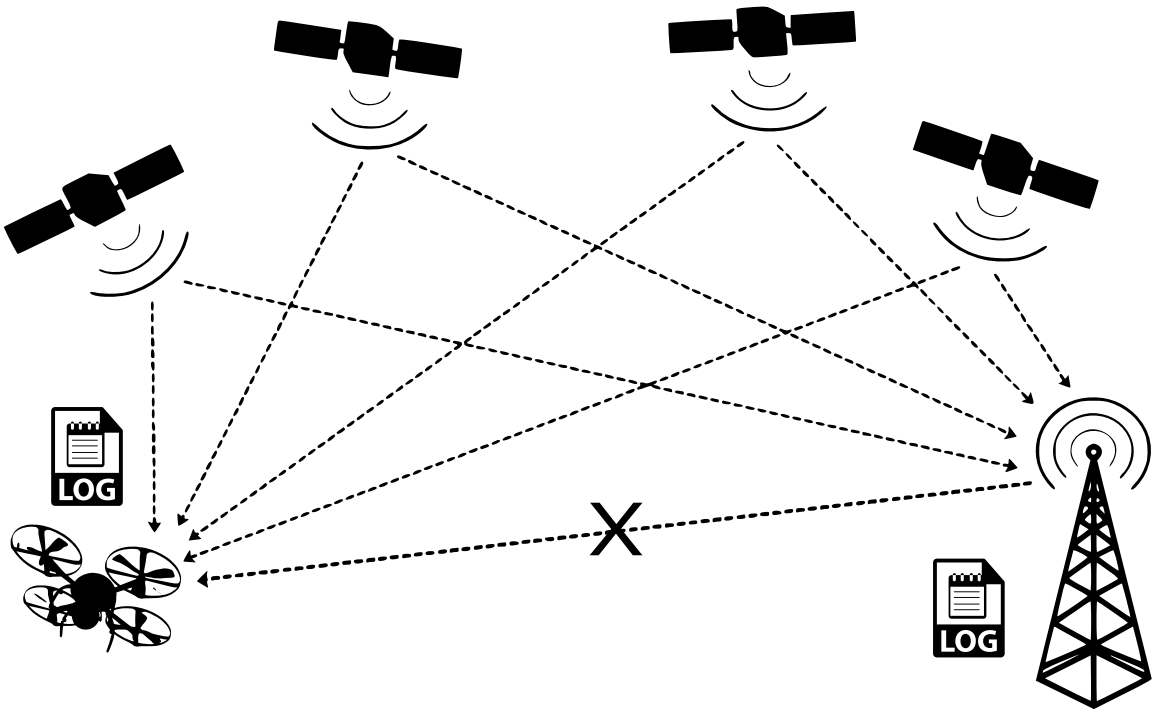


Abbildung 2: Teilprinzip 1 des PPK-Verfahrens

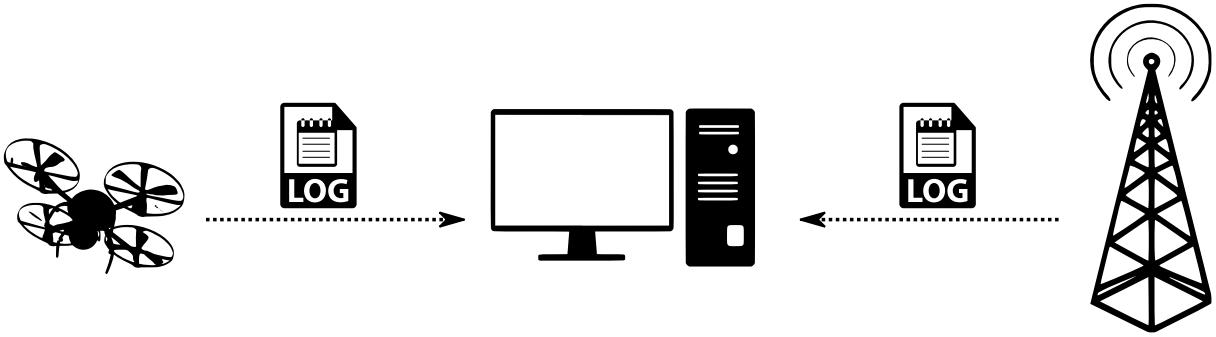


Abbildung 3: Teilprinzip 2 des PPK-Verfahrens.

2.4 Video vs. Einzelbilder

Photogrammetrische Produkte werden aus statischen Fotographien von Objekten abgeleitet. Ein Video ist eine Abfolge statischer Fotographien bekannter Anzahl pro Zeiteinheit[22].

Die Aufnahme von optischem Bildmaterial als Video weist gegenüber einzeln ausgelösten Aufnahmen folgende Vorteile auf:

- Wenig Schaltungsaufwand: Es muss nicht für jedes Bild ein Auslösevorgang durchgeführt werden. An Schaltwerke werden daher geringere Anforderungen gestellt, was die Investitions- und Wartungskosten minimiert. Auch der programmatische Aufwand zur Auslösung der Bildaufnahme wird minimiert.
- Es fällt bei Videoaufnahmen eine hohe Dichte an Einzelaufnahmen an (z. B. 10 Bilder/s). Es steht somit mehr Bildmaterial zur photogrammetrischen Verarbeitung zur Verfügung.

Nachteile:

- Einzelbilder können nicht direkt mit Georeferenzierungsinformationen synchronisiert werden.
- Einzelbilder müssen aus dem Video extrahiert werden.
- Exif-Information (mit Geokoordinaten für die direkte Georeferenzierung) müssen nachträglich angefügt werden.

[22] Z.B. 10 fps = 10 Einzelbilder bzw. statische Fotographien pro Sekunde.

- Durch die hohe Bilddichte und die Mitnahme von Tonspuren ergibt sich ein hoher Speicherplatzbedarf.

2.5 Structure from Motion

Beim Structure from Motion-Process werden in einem Satz statischer, optischer Bilder (Fotos) Muster (z. B. z. B. Gebäudeecken, Kreuz-Pflasterfugen) identifiziert (und über Deskriptoren beschrieben), die in mindestens 3 Einzelbildern wiederzufinden sind (feature matching). Solche features wurden pro Bild jeweils aus verschiedenen Aufnahmerichtungen und –entfernungen erfasst. Der SfM-Algorithmus rekonstruiert die relativen Aufnahmerichtungen und –entfernungen zu den punktuell verorteten features und leitet daraus die Kamerapositionen und –parameter (z. B. Brennweite, Verzeichnungskoeffizienten) ab (Toffanin 2019) (Westoby, et al. 2012).

Dem SfM-Prozess schließt sich der Multi-View-Stereo-Prozess (MVS) an, der auf Basis der zuvor rekonstruierten Kamerapositionen und –parameter eine 3D-Punktwolke erzeugt, die wiederum als Basis zur Erstellung von Meshing-Modellen[23] und Orthofotos dient. Bei der Orthofotoerstellung werden die Pixel der ursprünglichen 2D-Bilder mit den Punkten der 3D-Puntkwolke in Beziehung gesetzt und die RGB-Farbwerte in die 3D-Punktwolke übernommen. Durch Draufsicht auf die endgültige, zur interpolierten Oberfläche weiterverarbeiteten 3D-Punktwolke entsteht ein Orthofoto (Toffanin 2019) (Westoby, et al. 2012).

[23] Erstellung von interpolierten Oberflächen aus Punktdaten.

2.6 Positionsgenauigkeit

2.6.1 Einflüsse auf die Positionsgenauigkeit

Neben der Genauigkeit der Positionsmessungen am GNSS-Empfänger beeinflussen folgende Faktoren die Positionsgenauigkeit der photogrammetrischen Produkte:

- Wetter
  - Wirkt sich auf das Flugverhalten des UAS und damit infolge auf die geometrische Gleichheit des aufgenommenen Bildmaterials aus. Flugdurchführung daher möglichst bei windstiller Wetterlage[24] (Toffanin 2019).
  - Wirkt sich auf die Ausleuchtung des Geländes und damit auf die Güte des Bildmaterials aus. Flugdurchführung möglichst in der Mittagszeit bei maximaler Ausleuchtung und minimaler Verschattung sowie bei maximaler Bewölkung (hoher Anteil diffuser Strahlung, Schattenminimierung) (Toffanin 2019).
- Kameraspezifikationen
  - Größere und bessere Sensoren erzeugen weniger Rauschen und klarer fokussierte Bilder (Toffanin 2019).
  - Kamerakalibrierung (Bestimmung der wahren Brennweite, Verzeichnungskoeffizienten und Sensorgröße) (Toffanin 2019)
- Flughöhe:
  - Je größer die Flughöhe, desto größer der Bildausschnitt und die GSD[25]. Je größer die GSD, desto geringer ist die Genauigkeit, da weniger Details auf den Fotos abgebildet werden und damit weniger features identfiziert auf deren Basis die Rekonstruktionsvorgänge beim SfM (s. 2.5) basieren (Toffanin 2019).
- Fluggeschwindigkeit (Toffanin 2019)

[24] Mit steigender Windstärke erhöht sich die Verdriftung des UAV. Die Soll-Position kann unter diesen Umständen vom Regelkreis des Flight-Controllers, der Stromquelle und am Ende der Aktoren (Motorregler und Motoren) nicht gehalten werden, da (1) die Windkräfte stärker als die durch die Aktoren aufzubringenden Kräfte sind oder (2) in Richtung und Stärke pro Zeiteinheit zu variabel sind und das Ausregelungsvermögen des Flight-Controllers pro Zeiteinheit übersteigen. Hinzu kommt die erhöhte Schwankung des Luftdrucks innerhalb turbulenter Luft: Diese wird vom hochsensiblen Barometer, der als primärer Höhensensor fungiert, gemessen. Bei hohen Windstärken und damit erhöhter Luftmassenturbulenz kann das UAV daher die Soll-Höhe nicht halten. Insgesamt sind bei hohen Windstärken daher im erhöhten Maß unbeabsichtigte horizontale und vertikale Bewegungen des UAV im Umkreis der Soll-Position zu beobachten.

[25] engl.: ground sampling distance

2.6.2 Anforderungen an die Positionsgenauigkeit

Die geforderte Positionsgenauigkeit hängt vom Verwendungszweck der 3D-Punktwolke bzw. des Orthofotos ab. Niedrige Positionsgenauigkeiten sind in der Regel mit einem geringeren Kosten- und Zeitaufwand verbunden, da kostengünstigere GNSS-Empfänger eingesetzt werden und /oder Nachkorrekturen entfallen.

- Ist die Abbildung des Geoobjekts (digital twin) lediglich maßstabslos, ohne Georeferenz, beabsichtigt (z. B. für Visualisierungszwecke) kann die Georeferenzierung entfallen.
- Bei Inbezugsetzung zu Satellitendaten ist laut Padró, et al. (2019) eine mittlere Positionsgenauigkeit im Bereich der Lagegenauigkeit der Satellitenbilder ausreichend. (hier: Lagegenauigkeit). Diese liegt beispielsweise bei Sentinel 1A bei 2,5 m (Schubert, et al. 2014). Dem ist jedoch hinzuzufügen, dass sich die mittleren Lagegenauigkeiten im ungünstigsten Fall in entgegengesetzter Richtung ausprägen, sodass mit Lageungenauigkeiten von bis zu 5 m zu rechnen ist. Mögliche Einsatzbereiche wären grobe Change-Detection-Anwendungen. Beispielsweise qualitative, nicht-quantitative Feststellungen baulicher Veränderungen oder Veränderungen an Vegetationsbeständen (Neues Gebäude hinzugekommen? Fichtenbestand abgeholzt?).
- Bei Inbezugsetzung zu ALS[26]-Daten, Orthofotos (z. B. DOP10) oder sehr hoch aufgelösten Satellitendaten sind mittlere Positionsgenauigkeiten im Bereich von 0,5 m angemessen (Padró, et al. 2019).
- Bei Inbezugsetzungen zu anderen UAV-Aufnahmen (z. B. multitemporale Abgleichungen) oder bei Vermessungszwecken sind Genauigkeiten bis 0,05 m anzustreben (Padró, et al. 2019).

[26] engl.: airborne laserscanner

3 Methodik

3.1 Ausrüstung

3.1.1 Onboard

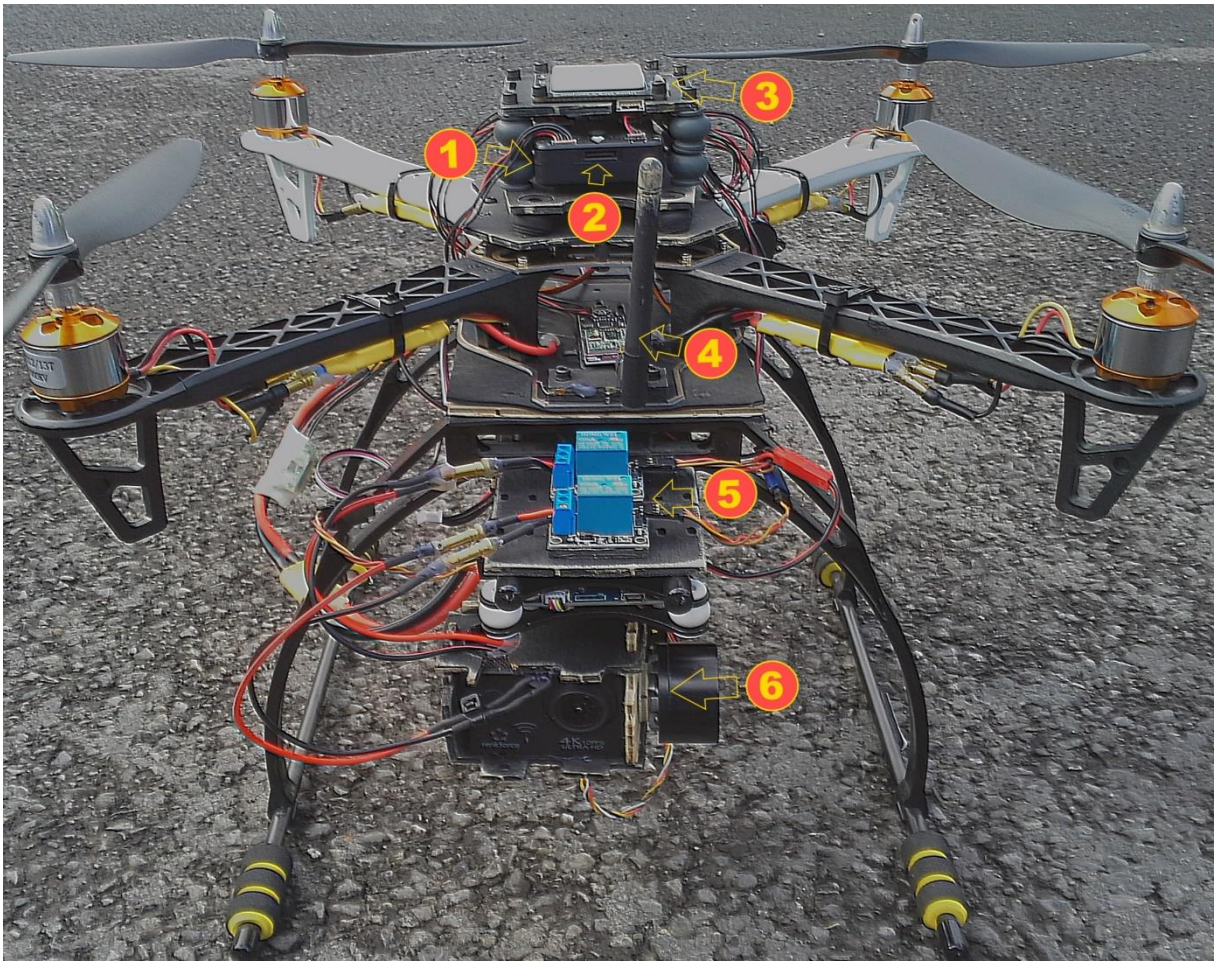


Abbildung 4: Vorderansicht des UAS. Wichtige Komponenten sind mit Nummer gekennzeichnet: (1) Flugregler (Pixhawk), (2) SD-Karten-Slot am Flugregler, (3) UBLOX-M8T-GNSS-Empfänger, (4) 433-MHz-Transceiver, (5) Relais 1 (vorne) und Relais 2 (hinten), (6) Kardanische Kamera-Aufhängung mit eingesetzter Kamera.

3.1.1.1 Rahmen und Antrieb

Beim eingesetzten UAV handelt es sich um einen 4-motorigen Drehflügler (Quadrokopter) mit einer Brutto-Startmasse von 1996 g[27], die damit knapp unterhalb der luftfahrtrechtlich kritischen Gewichtsgrenze von 2000 g liegt. Der UAV-Rahmen

[27] Mit 446 g ist der an der Unterseite angebrachte Lithium-Polymer-Akku die schwerste Komponente am UAV.

(Ausleger und Stromverteilerboard) ist ein chinesisches Fabrikat und Klon des Modells DJI F450. Der Achsabstand der Motoren[28] beträgt 455 mm. Die 4 Motoren müssen zusammen eine Schubkraft von 5988 g (1996 g * 3) aufbringen um eine stabiles Flugverhalten des Drehflüglers zu bewirken (Büchi 2018). Die anfangs verwendeten 1000kV-Motoren, mussten daher im Verlauf der Untersuchung durch leistungsstärkere Motoren ersetzt werden. Auch die Motorregler mussten durch solche mit größerem Stromdurchsatz ersetzt werden (vorher: 30A, nachher: 40A). Die Investitionskosten (s. 3.1.3) haben sich dadurch leicht erhöht.

Zusätzliche Bauteile, wie das Kamera-Case am Gimbal (s. 3.1.1.4 und Abbildung 4 (6)), die Akkuhalterung an der Unterseite und die vibrationsgedämpfte Halterung des Autopiloten und des GNSS-Empfängers an der Oberseite wurden selbstständig über eine CAD[29]-Anwendung[30] konstruiert, in NC-Code übersetzt[31] und über eine CNC[32]-Fräse aus Sperrholz gefertigt und anschließend lackiert. Die zusätzlichen Komponenten haben die Startmasse merklich erhöht und die oben beschriebene Leistungsverstärkung beim Antrieb erforderlich gemacht.

3.1.1.2 Flugregler

Die Entgegennahme und Verarbeitung von Steuerungskommandos, die Speicherung von programmierten Kommandos, die Regelung des Flugverhaltens und die Ausgabe von Steuerungssignalen an Aktoren wird durch den Flugregler Pixhawk 1 des Herstellers 3DR vollzogen (s. Abbildung 4 (1)). Dieses Modell wird mittlerweile nicht mehr hergestellt und wurde von leistungsstärkeren und weiterentwickelten Nachfolgemodellen am Markt abgelöst[33] (Dronecode Project, Inc. 2021). Der Autopilot basiert auf dem offenen Hardwaredesign PX4FMUv2[34]. Der Autopilot ist mit einem SD-

[28] Diagonal gemessen. Z. B. von vorne links nach hinten rechts.

[29] engl.: computer aided design

[30] Design-Spark Mechanical

[31] Estlcam

[32] engl.: Computerized Numerical Control

[33] mRo Pixhawk Flight Controller (Pixhawk 1) ist das gleiche Modell, das durch einen anderen Hersteller (mRo) hergestellt wird (Dronecode Project, Inc. 2021).

[34] PX4 FMU 2.4.6

Kartenslot ausgestattet (s. Abbildung 4 (2)). Log-Daten werden dateibasiert auf der SD-Karte gespeichert.

3.1.1.3 GNSS-Empfänger (Rover)

Zur Bestimmung der für die automatisierte UAV-Navigation notwendigen aktuellen Geoposition und des Azimutalwinkels (Himmelsrichtung) greift der Flugregler auf den Datenausgabestrom eines GNSS-Empfängers (Rover) und eines elektrischen Kompasses zu. Beim eingesetzten GNSS-Empfänger handelt es sich um das Modell NEO-M8T des Herstellers Ublox (ublox 2021) und beim Kompass um das Modell LIS3MDL des Herstellers STMicroelectronics (STMicroelectronics 2017). Die beiden Komponenten sind zum Modul DP0106 vom Hersteller Drotek (Drotek 2020) zusammengebaut worden, welches als externe Komponente über eine UART[35]-Schnittstelle (serielle Schnittstelle) an den Flugregler angeschlossen ist (s. Abbildung 4 (3)).

NEO-M8T empfängt Signale der GNSS GPS (USA), GLONASS (Russland), Galileo (Europa) und Beidou (China), wobei aber aufgrund der teilweisen Überschneidung von Frequenzbändern der genannten GNSS (Bauer 2018) nur bestimmte Empfangs-Kombinationen möglich sind (ublox 2021). Voreingestellt, und für die Navigationsaufgaben des Flugreglers und die in dieser Arbeit fokussierten Georeferenzierungszwecke sinnvoll, ist die Kombination GPS mit GLONASS[36]. NEO-M8T empfängt lediglich Signale aus den relativ tieffrequenten Level-1-Bändern (untere L-Bänder (Bauer 2018)). Die hochfrequenten Signale der Level-2-Bänder (obere L-Bänder (Bauer 2018)), die im Empfänger oder im PPK-Verfahren in Kombination mit den L1-Signalen die rechnerische Eliminierung von Störeinflüssen unmittelbar erheblich erleichtern und dadurch zu erheblich genaueren Positionslösungen führen, werden vom NEO-M8T nicht empfangen. Die vom Empfänger berechnete Positionslösung erreicht eine horizontale Genauigkeit von mindestens 2,5 bis 4 m (ublox 2021). Die Bewegungsgeschwindigkeit des Empfängers (Rover) und damit des

[35] engl.: Universal Asynchronous Receiver Transmitter

[36] Aufgrund der deutlich unterschiedlichen Frequenzbandaufteilung der beiden GNSS ergibt sich eine deutliche Signaltrennung, was die Signalweiterverarbeitung im Empfänger und im PPK-Verfahren begünstigt.

UAS wird in einer Genauigkeit von 0,05 m/s bestimmt (ublox 2021). Die Messepochenabfolge erreicht eine maximale Frequenz von 4 Hz (250 ms) (ublox 2021). NEO-M8T kann bis in eine Höhe über Grund von 50000 m, bis zu einer Geschwindigkeit von 500 m/s und bis zu einem Erdbeschleunigungseinfluss von 4g eingesetzt werden (ublox 2021). Diese Betriebsgrenzen werden im Rahmen der hier durchgeführten Flugkampagne bei weitem nicht erreicht und es ist daher von einem zuverlässigen Betrieb des GNSS-Empfängers auszugehen.

Die von NEO-M8T berechnete Positionslösung (WGS84-Geokoordinaten [x,y,z]) wird mittels des proprietären UBX-NAV-Protokolls[37] über eine UART-Schnittstelle an den Flugregler ausgegeben (Dronecode Project, Inc. 2020), der diese als Navigationsgröße im Rahmen der Flugregelung nutzt. Im UBX-NAV-Protokoll sind pro Messepoche neben der Positionslösung unter anderem die horizontale und vertikale Empfängergeschwindigkeit und die GPS-Zeit enthalten (ublox 2021). In der Flugregler-Logdatei werden diese in der Nachricht GPS wiedergegeben (ArduPilot Dev Team 2021) (s. Anhang III 13). Darüber hinaus enthält das UBX-NAV-Protokoll Mitteilungen zur Genauigkeit der GNSS-Messwerte, welche in der Flugregler-Logdatei gesondert in der Nachricht GPA[38] (GPS accuracy information) wiedergegeben werden (ArduPilot Dev Team 2021) (s. Anhang III 13).

Zur groben Georeferenzierung (Positionsungenauigkeit von max. 2,5 bis 4 m) von im Bildflug erfassten Einzelbildern würden die Geokoordinaten in der GPS-Nachricht der Log-Datei bereits ausreichen.

Die mögliche Ungenauigkeit der Positionslösung von bis zu 4 Metern in der Horizontalen erfordert jedoch eine Korrektur, wie sie unter 3.4.2 vorgenommen wird. Bei der Korrektur der Positionslösung im PPK-Verfahren wird die im Empfänger bereits durchgeführte Berechnung der Positionslösung unter Einbezug von Messgrößen eines zweiten Empfängers (Base) wiederholt (Zhang, et al. 2019) (Dinkov und Kitev 2020). Zu den dafür erforderlichen Messgrößen gehören die Pseudorange, die Trägerphasenverschiebung und die Doppler-Frequenzverschiebung (Bauer 2018).

[37] nicht NMEA

[38] https://ardupilot.org/copter/docs/logmessages.html#gpa

Diese Größen werden im GNSS-Empfänger zwar in jeder Messepoche ermittelt und verarbeitet sowie über das UBX-RXM-RAWX-Protokoll nach Außen übertragen. In der Werkseinstellung ignoriert der Flugregler das UBX-RXM-RAWX-Protokoll und übernimmt die darin enthaltenen Mitteilungen nicht in die Log-Datei. Dies muss geändert werden indem der Parameter GPS_RAW_DATE vom default-Wert 0 (=Ignore) auf 2 (=Stop logging when disarmed) gesetzt wird[39] (ArduPilot Dev Team 2021). Danach werden im operativen Betrieb des UAS[40] und damit des eingesetzten Flugreglers im Abstand von 200 ms (5 Hz) auf Basis des UBX-RXM-RAWX-Protokolls vom Flugregler die oben genannten Messgrößen als GRXH- bzw. GRXS-Nachrichten in der Log-Datei aufgezeichnet.

3.1.1.4 Kamera

An der Drohne ist eine Action-Kamera (Renkforce RF-AC-4K 120) angebracht. Sie ist sehr preiswert (s. 3.1.3), wiegt (inklusive Lithium-Ionen-Akku) lediglich 58 g und hat die Abmessungen 59 x 40 x 30 mm (Conrad Electronic SE 2021). Sie ist aufgrund ihrer geringen Masse und der kompakten Abmessungen für den Einsatz an einem UAS geeignet. Die Kamera ist mit einem Weitwinkelobjektiv (Öffnungswinkel 120°) ausgestattet. Die aufgenommenen Bilder weisen deswegen eine tonnenförmige Verzeichnung[41] auf, die aufgrund von noch zu ermittelnden Kameraparametern vor dem SfM-Prozess (s. 3.6) zu kompensieren ist. Als Bildebene fungiert der CMOS-Sensor GC2035 des chinesischen Herstellers GalaxyCore Inc. (GalaxyCore Inc. 2012), der im Videomodus bis zu 10 Einzelbilder pro Sekunde (10 fps) in einer Auflösung von 3840 x 2160 Pixeln[42] erfasst. Der CMOS-Sensor hat die Abmessungen 6,16 x 4,62 mm. Die aufgenommenen Videos werden persistent und dateibasiert im mp4-Format auf einer micro-SD-Karte (max. 128 GB) als externem Speichermedium gespeichert.

[39] Z. B. unter Mission Planner im config-Bereich.

[40] Sobald sich das UAV im Status ARMED befindet.

[41] Sog. Fischaugeneffekt (engl.: fisheye).

[42] Die Kamera fällt damit in die Kategorie der HD(=high definition)-Kameras

In der Kamera ist ein Akkumulator mit einer Nennbetriebsdauer von 60 min. (im Videomodus) integriert. Die Kamera wird ausschließlich über diesen Akkumulator mit Strom versorgt, sie ist nicht an den Stromkreis der Drohne angeschlossen.

Grundsätzlich besteht das Problem, dass die Kamera aufgrund fehlender Signalisierungswege während des Flugs nicht von außen angesteuert werden kann. Es bestünde die Möglichkeit die Kamera über die entsprechenden Schaltknöpfe durch den Operator manuell vor dem Start einzuschalten und die Videoaufnahme zu starten. Dies hat, neben dem geringen Komfort für den Operator, den entscheidenden Nachteil, dass der Startzeitpunkt des Videos nicht mit (geloggten) Ereignissen des UAS-Flugs in Bezug gesetzt werden kann. Der Startzeitpunkt des Videos kann hingegen mit (geloggten) Ereignissen des UAS-Flugs in Bezug gesetzt werden, wenn die Kamera durch den Flugregler beschaltet wird. Denkbar wäre z. B. die Zwischenschaltung eines Companion-Computers (z. B. Raspberry Pi) der vom Flugregler ereignisbasierte Befehle entgegennimmt und in Quasi-Echtzeit entsprechende Schaltsignale per WiFi-Funkverbindung an die Kamera überträgt. Da jedoch die WiFi-Funksignale im selben Frequenzband (2,4 GHz) wie die Duplex-Signale zwischen Flugregler und Fernsteuerung übertragen werden, sollte man von dieser Möglichkeit aus sicherheitstechnischen Gründen[43] absehen.

Es wurde daher folgende kabelgebundene Beschaltung realisiert: An der Kameravorderseite und der Kameraoberseite ist jeweils ein Drucktaster angebracht. Der Drucktaster auf der Vorderseite schließt Stromkreis 1, der auf der Oberseite Stromkreis 2. Bei 3-sekündigem Schluss von Stromkreis 1 wird die Kamera eingeschaltet. Bei eingeschalteter Kamera und 0,5-sekündigem Schluss von Stromkreis 1 wird der Aufnahmemodus gewechselt (z. B. Foto- zu Videomodus) bzw. in den Konfigurationsmodus gewechselt. Bei 1-sekündigem Schluss von *Stromkreis 2* wird die Videoaufnahme gestartet. Ein erneuter 1-sekündiger Schluss von *Stromkreis 2* stoppt die Videoaufnahme.

Um die Schaltzustände der beiden Stromkreise über den Flugregler zu steuern, wurden die beiden Drucktaster entfernt und jeweils 2 Kabelenden mit den dort

[43] Es besteht die Gefahr des Auftretens von Interferenzen und damit unbeabsichtigten Steuerungsanweisungen.

befindlichen Kontaktstellen verlötet (s. Abbildung 4 (6)). Die jeweils 2 noch unverbundenen Kabelenden wurden an Relais 1 bzw. Relais 2 angeschlossen (s. Abbildung 4 (5)). Werden nun die Relais geschaltet, ändert sich auch der Schaltzustand in Stromkreis 1 bzw. Stromkreis 2. Die Relais und die Kabelverbindungen an die Kontaktstellen ersetzen somit die mechanischen Drucktaster.

Relais 1 und Relais 2 werden jeweils über ein 3-adriges Servokabel mit einem freien PWM[44]-Ausgang am Flugregler verbunden, wodurch nun Schaltbefehle vom Flugregler an die Relais übertragen werden. Am Flugregler müssen dazu noch folgende Konfigurationen vorgenommen werden: Unter anderem ist der Parameter BRD_PWM_COUNT auf 4 zu setzten um sicherzustellen, dass die PWM-Ausgänge Nr.5 und Nr.6 der insgesamt 6 PWM-Ausgänge nicht als PWM-Ausgänge (z. B. zur Steuerung von Motoren, Servos oder anderen Aktoren), sondern als GPIO-Ausgänge[45] (ArduPilot Dev Team 2021) zur Relaisschaltung fungieren (ArduPilot Dev Team 2021). Danach ist der Parameter RELAY_PIN auf den Wert 54 und der Parameter RELAY_PIN2 auf den Wert 55 zu setzten. Die Werte 54 und 55 sind dabei Bezeichner für die PWM-Ausgänge AUXOUT5 bzw. AUXOUT6, an die die Signalkabel zu den Relais angeschlossen werden (ArduPilot Dev Team 2021). Relais 1 kann nun über AUXOUT5 und Relais 2 über AUXOUT6 durch den Flugregler geschaltet werden.

Die Kamera ist an der Drohne in einem Case untergebracht, das wiederum an einer kardanischen, um die Nick- und Rollachse beweglichen Aufhängung (sog. Gimbal) montiert ist (s. Abbildung 4 (6)). 2 Elektromotoren regulieren Drehbewegungen um die Nick- und Rollachse und halten die Kamera praktisch kontinuierlich in einer Soll-Stellung (Büchi 2018). An der Unterseite vom Kamera-Case ist ein Beschleunigungssensor angebracht, der Winkelabweichungen und –beschleunigungen in allen 3 Raumrichtungen erfasst. Ein Mikrocontroller, als Teil des Gimbal-Systems, verarbeitet diese Messgrößen als Abweichungen von der Soll-Stellung und berechnet daraus Steuersignale für die beiden Elektromotoren an der Nick- und Rollachse (Büchi 2018). Beim Bildflug wird auf diese Weise sichergestellt,

[44] PWM: Pulsweitenmodulation. Erläuterungen dazu bei (Schnabel 2022).

[45] General Purpose Input/Output

dass das Kameraobjektiv stets entlang der Lotrichtung zum Geoid, also zum Massenschwerpunkt der Erde, ausgerichtet ist (Orthoaufnahme).

Der Mikrocontroller des Gimbals ist per Servokabel an einem freien PWM-Ausgang des Flugreglers angeschlossen. Per PWM-Signal kann die Kamera vom Flugregler um die Nick-Achse verdreht werden.

3.1.2 Offboard

3.1.2.1 Hardware

3.1.2.1.1 Fernsteuerung

Der Bildflug wird als automatisch-autonomer Flug über eine programmierte Kommandoabfolge abgewickelt (s. 3.2.2). Jedoch muss der UAS-Operator nach luftfahrtrechtlichen Vorgaben jederzeit aktiv in den Flugbetrieb eingreifen können. Insbesondere aus diesem Grund gehört zur Ausrüstung eine Fernsteuerung. Außerdem erfolgen die Motorscharfstellung durch den Flugregler sowie die Flugmodi-Wechsel[46] zu Beginn der Flugdurchführung über Steuersignale aus der Fernsteuerung (ArduPilot Dev Team 2021).

Zum Einsatz kommt das Modell Futaba T10J (T-FHSS Air-2.4 GHz 10J), das über 10 Funkkanäle im 2,4 GHz-Frequenzband verfügt (Futaba Corporation of America 2018). Neben den 4 Hauptkanälen (Rollen, Nicken, Gas, Gieren) wird zwingend ein weiterer Kanal für die Setzung des Flugmodus benötigt. Es wäre möglich, vor dem Hintergrund einer Investitionskostenminimierung, die Bildflugoperation mit einer deutlich preisgünstigeren Fernsteuerung mit lediglich 5 Funkkanälen umzusetzen.

3.1.2.1.2 Geländecomputer

Bei der Flugdurchführung ist ein Geländecomputer im unmittelbaren Einsatzbereich als Bodenstation (GCS[47]) zu postieren[48], auf dem eine Python-Laufzeitumgebung einschließlich der Python-Bibliotheken dronekit und pymavlink eingerichtet ist. Von

[46] Z. B. Loiter zu Waypoint

[47] engl: ground control station

[48] Die Positionierung der GCS wird durch die Reichweite der 433-MHz-Datenfunkverbindung begrenzt.

diesem Computer aus werden die Kommandosequenzen der Flugmission auf den Flugregler übertragen. Zum Einsatz kommt das Modell Panasonic Toughbook cf-19 (Panasonic Corporation 2008). Hierbei handelt es sich um einen robusten und für den Einsatzzweck ausreichend performanten Geländecomputer, der z. B. als Gebrauchtgerät sehr preisgünstig erhältlich ist (s. 3.1.3).

3.1.2.1.3 Datalink

Zwischen Flugregler und Geländecomputer wird eine bidirektionale Funkverbindung zur Übertragung der Kommandosequenzen und von Zustandsdaten der Drohne im 433-MHz Frequenzband hergestellt. Die Daten werden mittels MAVLink-Protokoll übertragen (Koubaa, et al. 2019). Dazu wird auf jeder Teilnehmerseite (UAV bzw. Geländecomputer) jeweils ein externer Transceiver (Datalink) über eine serielle Daten-Schnittstelle an das Quell- bzw. Zielsystem angeschlossen. Zwischen Transceiver und Flugregler erfolgt die Datenübertragung via UART-Schnittstelle (s. Abbildung 4 (4)), zwischen Transceiver und Geländecomputer via USB[49]-Schnittstelle[50] (ArduPilot Dev Team 2021).

3.1.2.1.4 GNSS-Kontrollstation (Base)

Bei der Positionsbestimmung mittels differenziellem GNSS (D-GNSS) werden in jeder Messepoche die GNSS-Signale von mindestens 2 Empfängern gemessen (Bauer 2018). Auch das Postprocessed-Kinematic-Verfahren (PPK), bei dem die Positionslösungen eines Empfängers nachträglich und nicht in Echtzeit korrigiert werden sollen, ist der Konstellation der Empfänger nach ein D-GNSS (Bauer 2018). Neben dem Empfänger am UAV, dem Rover, der sich meistens im Raum bewegt, wird dazu ein zweiter, ortsfester Empfänger, die Base, am Einsatzort positioniert. Sowohl der Rover (s. 3.1.1.3) als auch die Base protokollieren die Messgrößen jeder Epoche (Pseudorange, Trägerphasenverschiebung, Doppler-Frequenzverschiebung).

Die Landesvermessungsämter unterhalten landesweit im Rahmen der Dienstleitung SAPOS ein Netz von ortsfesten GNSS-Empfängern, sog. Referenzstationen (Riecken

[49] engl.: Universal Serial Bus

[50] Zugriff auf Transceiver über COM-Port.

und Kurtenbach 2017). Die Referenzstationen sind in Abständen von ca. 30 km über das Land verteilt. Die Messwerte jeder Referenzstation werden an ein zentrales Rechenzentrum übertragen und dort stationsbezogen im RINEX-Format langfristig gespeichert (Riecken und Kurtenbach 2017). Die Interpolation der Messwerte der einzelnen Referenzstationen ermöglicht die Berechnung der Messwerte, die an einem beliebigen Ort im Feld zwischen 2 oder mehreren Referenzstationen gemessen würden, falls dort ein GNSS-Empfänger positioniert wäre. Der Standort auf den solch eine Interpolation ausgerichtet ist, wird als virtuelle Referenzstation (VRS) bezeichnet (Bauer 2018, 255). Das Landesvermessungsamt stellt als Teil der Dienstleistung SAPOS die Messwerte einer solchen VRS im RINEX-Format bereit (Riecken und Kurtenbach 2017). Die Dienstleistung erfolgt entgeltfrei. Der Vorgang wird unter 3.4.2.3 näher beschrieben.

3.1.2.2 Software

3.1.2.2.1 Python-Bibliotheken

Von der Flugvorbereitung über die Flugdurchführung, die GNSS-Datenaufbereitung bis zur Einzelbildextraktion und –georeferenzierung werden die jeweiligen Prozesse über Python-Skripte abgewickelt. Weitestgehend werden die Programme ohne Einbezug externer Module und mit einfachen Mitteln wie Listen, Dictionaries und Schleifen realisiert. Wo dies nicht möglich ist, werden geeignete externe Module in die Programme eingebunden. Tabelle 2 listet alle eingesetzten externen Module auf. Die Nutzung der Python-Ressourcen erfolgt entgeltfrei.

Tabelle 2: Verwendete Python-Module

Modul-Name	**Beschreibung**
calendar	Bereitstellung allgemeiner kalenderbezogener Funktionen (Python Software Foundation 2022)
opencv-python (cv2)	Umfangreiche Open-Source-Bibliothek für Computer-Vision, maschinelles Lernen und Bildverarbeitung. (OpenCV 2022)
Datetime	Bereitstellung von Funktionen zur Bearbeitung von Datums- und Zeitangaben. (Python Software Foundation 2022)

dronekit	Bereitstellung von Funktionen zur Kommunikation mit Flugreglern via MAV-Link von externen Computern aus. (3D Robotics 2016)
math	Bereitstellung mathematischer Funktionen. (Python Software Foundation 2021)
numpy	Bereitstellung von mathematischen Funktionen, Zufallszahlengeneratoren, Routinen der linearen Algebra, Fourier-Transformationen usw. (NumPy Developers 2022)
piexif	Bereitstellung von Funktionen zum Extrahieren, Erstellen, Manipulieren, Konvertieren und Schreiben von EXIF-Daten in JPEG-, und TIFF-Dateien. (hMatoba 2018)
PIL	Python Imaging Library. Bereitstellung von Bildverarbeitungs-Funktionen. (Lundh und Clark 2011)
pymavlink	Verarbeitung von MAV-Link-Protokollströmen und Protokolldateien. (Dronecode Project, Inc. 2022)
qgis.core	Bereitstellung aller grundlegenden GIS-Funktionen. (QGIS 2022)

3.1.2.2.2 Groundstation

Zur Konfiguration des Flugreglers und zur Echtzeit-Wiedergabe der Telemetriedaten während der Flugdurchführung (Flugmonitoring auf der Ground Station) kommt die Anwendung Mission Planner[51] zum Einsatz. Bezug und Nutzung der Anwendung sind entgeltfrei.

3.1.2.2.3 Zugang SAPOS

Voraussetzung für die Beauftragung des SAPOS-NRW[52] mit der Erstellung von VRS-Daten im RINEX-Format (s. 3.1.2.1.4 und 3.4.2.3) ist ein Nutzerkonto. Die Erstellung und Nutzung des Nutzerkontos sowie die Erstellung und Bereitstellung der VRS-Daten sind entgeltfrei.

3.1.2.2.4 Emlid Studio

Emlid-Studio[53] ist eine Anwendung mit grafischer Benutzeroberfläche zur Umsetzung der Nachberechnung der GNSS-Daten mit dem PPK-Verfahren (s. 2.3.2). Im Kern basiert die Anwendung auf die weit verbreitete Software mit offenem Quell-Code RTK-LIB, im Speziellen der Unteranwendung RTK-Post. Emlid-Studio ist entgeltfrei erhältlich und nutzbar.

3.1.2.2.5 OpendroneMap

OpendroneMap[54] (ODM) ist eine kommandozeilenbasierte, unter Windows ausführbare, Anwendung zur Umsetzung der unter 2.5 beschriebenen SfM-Prozesskette. ODM ist entgeltfrei erhältlich und nutzbar. Der Quell-Code ist offen.

(Burdziakowski 2017) zeigt in seiner Untersuchung, dass sich die durch ODM (Version 0.3) erzeugten photogrammetrischen Produkte für geodätische Zwecke eignen und sich von kommerzieller Software (z. B. Pix4D oder Agisoft Photoscan) nur durch den geringeren Bedienkomfort und die unzureichende Auflösung von Kamera-Verzeichnungsfehlern bei Weitwinkelobjektiv-Kameras unterscheiden. Letzteres

[51] https://ardupilot.org/planner/

[52] https://gppspro.saposnrw.de/

[53] https://docs.emlid.com/emlid-studio/

[54] https://www.opendronemap.org/odm/

wurde jedoch in den neueren Versionen, u. a. der im Rahmen dieser Arbeit verwendeten Version 2.5.7, behoben.

3.1.3 Kosten

Die Investitionskosten für die unter 3.1 aufgeführten UAS-Komponenten belaufen sich in der Summe auf rund 900 €. Die größte Kostenposition bildet dabei die Fernsteuerung in Höhe von 384 €. Eine geeignete Fernsteuerung ist aber bereits ab 50 €erhältlich, sodass sich die Kosten für ein ähnliches Hardware-Set auf ca. 580 € belaufen würden. Tabelle 3 listet alle Investitionskostenpositionen auf. Die Herstellungskosten belaufen sich auf 1120 € (zugrunde gelegter Stundensatz: 35 €) werden in Tabelle 4 aufgeführt.

Tabelle 3: Investitionskostenaufstellung (HW=Hardware, SW=Software)

Pos	Kat.	Bezeichnung	Anzahl	Einzel-preis [€]	Preis [€]
1	HW	Actioncam	1	29,99	29,99
2	HW	Fernsteuerung(mit Empfänger)	1	384,00	384,00
3	HW	Gimbal	1	58,99	58,99
4	HW	GNSS-Empfänger (Base)	1	0,00	0,00
5	HW	GNSS-Empfänger (Rover) + Kompass	1	49,90	49,90
6	HW	Landegestell	1	33,59	33,59
7	HW	Motor + Motorregler	4	10,50	42,00
8	HW	Panasonic Toughpad CF-19	1	150,00	150,00
9	HW	Pixhawk (Datalink, Spannungsteiler)	1	141,99	141,99
10	HW	UAV-Rahmen (Verteilerboard, Ausleger, Schrauben)	1	21,59	21,59
11	SW	Groundstation	1	0,00	0,00
12	SW	Open-Drone-Map	1	0,00	0,00
13	SW	Python-Bibliotheken	1	0,00	0,00
14	SW	Zugang SAPOS NRW	1	0,00	0,00
15	SW	Emlid Studio	1	0,00	0,00
					912,05

Tabelle 4: Herstellungskostenaufstellung

Pos	Bezeichnung	Anzahl [h]	Einzel-preis [€/h]	Preis [€]
1	Rahmenzusammenbau	2	35	70
2	Anschluss Motoren/Motorregler	3	35	105
3	Konstruktion Gimbal-Case	4	35	140
4	Fertigung und Montage Gimbal-Case/Relais-Schaltung	4	35	140
5	Konstruktion Pixhawk-/GNSS-Träger	4	35	140
6	Fertigung und Montage Pixhawk-/GNSS-Träger	4	35	140
7	Konstruktion Flugakku-Halterung	3	35	105
8	Fertigung und Montage Flugakku-Halterung	3	35	105
9	Konfiguration Pixhawk	5	35	175
		32		**1120**

3.2 Flug

3.2.1 Flugvorbereitung

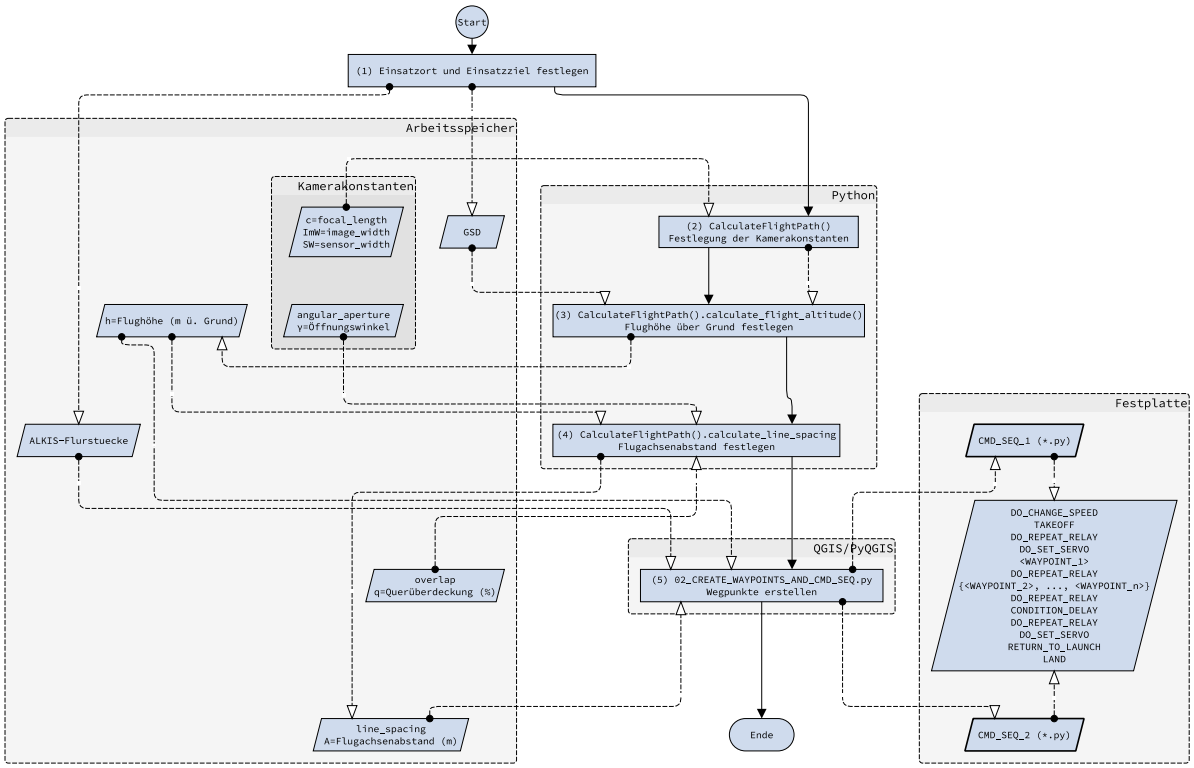


Abbildung 5: Ablaufschema der Flugvorbereitung. Durchgezogene Graphen: Prozessflüsse, gestrichelte Graphen: Datenflüsse. Quelle: Eigene Darstellung.

3.2.1.1 Einsatzort und Einsatzziel festlegen

Zuerst wird der Einsatzort festgelegt. Im Rahmen dieser Arbeite handelt es sich dabei um ein Einzelhausgrundstück. Daraus ergibt sich die Lage und Ausdehnung des AOI[55], das z. B. durch ein ALKIS[56]-Flurstück repräsentiert werden kann. Das Einsatzziel ist die Durchführung eines UAS-Bildflugs zur Anfertigung einer Ortho-Videoaufnahme aus der 3D-Punktwolen (LAS-Format) und Orthofotos unter Nutzung von OpenDroneMap produziert werden können. Ziel ist es, das auf dem Grundstück befindliche Hauptgebäude, die Nebengebäude (Gartenschuppen) und die Vegetation in ausreichender Genauigkeit abzubilden.

[55] engl: area of interest.

[56] Amtliches Liegenschaftskatasterinformationssystem.

In Übereinstimmung mit der Bodenabtastrate (GSD) der als Vergleichsgrundlage unter 3.7.1 heranzuziehenden DOP10-Orthofotos wird eine GSD von 10 cm/Pixel festgelegt.

3.2.1.2 Flughöhe über Grund festlegen

Tabelle 5 listet die zur Berechnung der Flughöhe über Grund h erforderlichen Größen auf. Abbildung 6 veranschaulicht diese Größen. Die Bildbreite ImW[57], die Sensorbreite SW[58] und die Kamerakonstante[59] c sind über die Kennwerte der Kamera (vgl. 3.1.1.4) bekannt. Aufgrund der Genauigkeitsanforderungen (vgl. 2.6.2) wird eine Bodenabtastrate GSD von 5 cm pro Pixel festgelegt.

Tabelle 5: Formelelemente zur Berechnung der Flughöhe über Grund.

Kürzel	Bezeichnung	Wert	Einheit	Status	
c	Kamerakonstante	8,00	mm	konstant	gegeben
GSD	Bodenabtastabstand (ground sampling distance)	10,00	cm/Pixel	variabel	gegeben
h	**Flughöhe über Grund**		**m**	**variabel**	**gesucht**
ImW	Bildbreite (image width)	3840	Pixel	konstant	gegeben
SW	Sensorbreite (sensor width)	6,16	mm	konstant	gegeben

Zur Berechnung der Flughöhe über Grund h wird innerhalb eines Python-Scripts[60] der Funktion calculate_flight_altitude der Klasse CalculateFlightPath[61] (s. Anhang I) die in Tabelle 5 als gegeben gekennzeichneten Größen als Parameter übergeben. Formel 1 gibt den in der vorgenannten Funktion programmierten Rechenweg in mathematischer Notation wieder. In Formel 2 sind die gegebenen Werte eingesetzt und h berechnet. Es ergibt sich eine Flughöhe über Grund h von rd. 25 m.

[57] engl.: image width

[58] engl.: sensor width

[59] Oftmals auch als Brennweite oder mit dem englischen Begriff focal length bezeichnet.

[60] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/01_CALCULATE_FLIGHTALT_AND_FLIGHTLINESPACING.py

[61] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/modules/calculate_flight_path.py

$$h = \frac{ImW[px] * GSD\left[\frac{cm}{px}\right] * c[mm]}{SW[mm] * 1000}$$ *Formel 1*

$$h = \frac{3840[px] * 5\left[\frac{cm}{px}\right] * 8[mm]}{6{,}16[mm] * 1000} \approx 50[m]$$ *Formel 2*

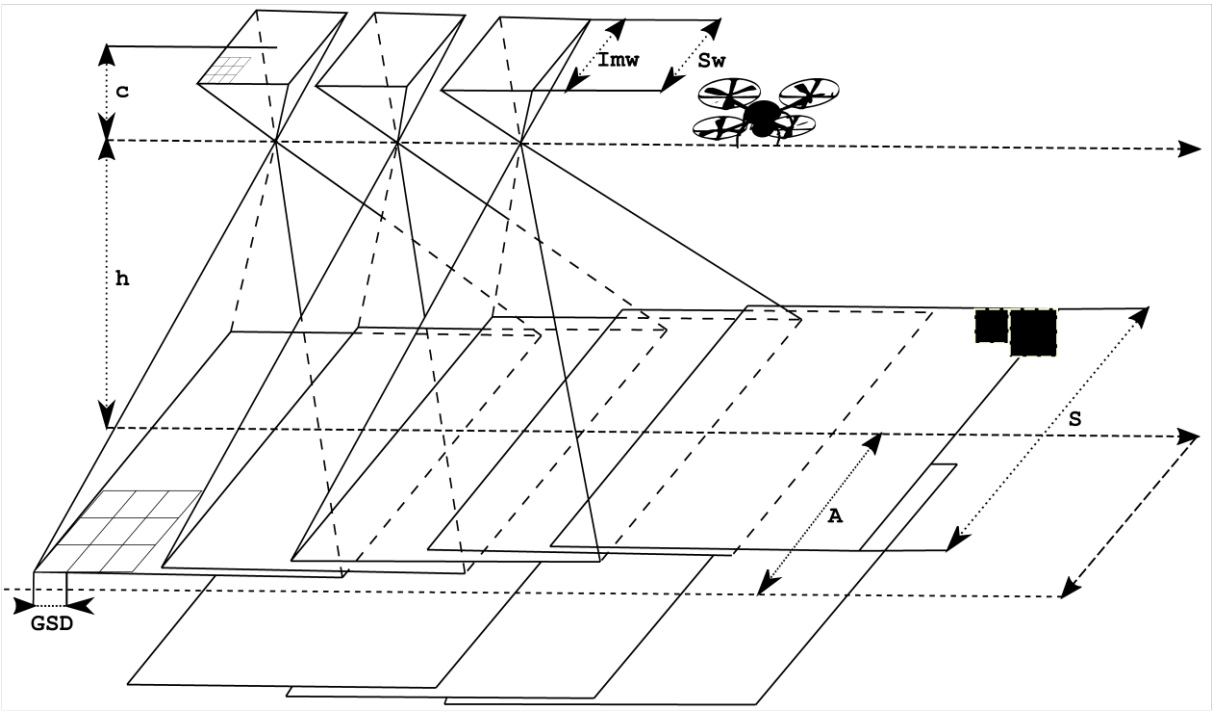


Abbildung 6: Schematische Darstellung der wichtigsten Parameter der Flughöhenberechnung. Quelle: In Anlehnung an (Kraus 2004) und (Pix4D 2021).

3.2.1.3 Flugachsenabstand festlegen

Maßgebend für die Festlegung des Flugachsenabstands A (s. Abbildung 6) ist die beabsichtigte Querüberdeckung q der Bilder. Empfohlen wird eine Querüberdeckung von 75% und eine Längsüberdeckung l von 80%, um im SfM-Prozess gute Ergebnisse zu erzielen (Toffanin 2019) (Die Längsüberdeckung ist erst im Rahmen der Einzelbildextraktion (s. 3.5) zu berücksichtigen.). Zur Berechnung von A wird die unter 3.2.1.2 berechnete Flughöhe über Grund h, der Kameraöffnungswinkel γ (vgl. 3.1.1.4) und die beabsichtigte Querüberdeckung q, als bekannt vorausgesetzt. Tabelle 6 listet die zur Berechnung erforderlichen Größen auf. Abbildung 6 veranschaulicht diese Größen.

Tabelle 6: Formelelemente zur Berechnung des Flugachsenabstands.

Kürzel	Bezeichnung	Wert	Einheit	Status	
A	**Flugachsenabstand**		**m**	**variabel**	**gesucht**
h	Flughöhe über Grund	50,00	m	variabel	gegeben
q	Querüberdeckung	0,75	-	variabel	gegeben
S	**Aufnahmebreite**		**m**	**variabel**	**gesucht**
α	Dreieckswinkel		°	variabel	gesucht
γ	Öffnungswinkel	120,00	°	konstant	gegeben

Zur Berechnung des Flugachsenabstands A wird der Python-Funktion calculate_line_spacing der Klasse CalculateFlightPath[62] (s. Anhang I) die in Tabelle 6 als gegeben gekennzeichneten Größen als Parameter übergeben.

Wie in Formel 3 und Formel 4 mathematisch notiert wiedergegeben, werden innerhalb von calculate_line_spacing auf Basis der gegebenen Größen γ und h zuerst die beiden unbekannten Schenkelseiten a und b des vom Kameraobjektiv aufgespannten Winkels in der XZ-Ebene berechnet. a und b sind lediglich Zwischenergebnisse im Gültigkeitsbereich der Funktion und werden von ihr nicht zurückgegeben.

$$\alpha = \frac{180° - \gamma}{2} = \frac{180° - 120°}{2} = 30°$$ *Formel 3*

$$a = b = \frac{h}{\sin \alpha} = \frac{20\text{m}}{\sin 30°} = 100\ \text{m}$$ *Formel 4*

Mit dem Kosinussatz (Formel 5) und den nun vollständig bekannten Größen a, b und γ wird innerhalb der Funktion anschließend die Aufnahmebreite S berechnet.

$$S = \sqrt{a^2 + b^2 - 2 * a * b * \cos \gamma} \approx 173\text{m}$$ *Formel 5*

[62] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/modules/calculate_flight_path.py

Aus der Aufnahmebreite S und der beabsichtigten Querüberlappung q wird danach der Flugachsenabstand A berechnet.

$$A = S * (1 - q) = 173m * (1 - 0{,}75) \approx 43m$$ *Formel 6*

3.2.1.4 Wegpunkte erstellen

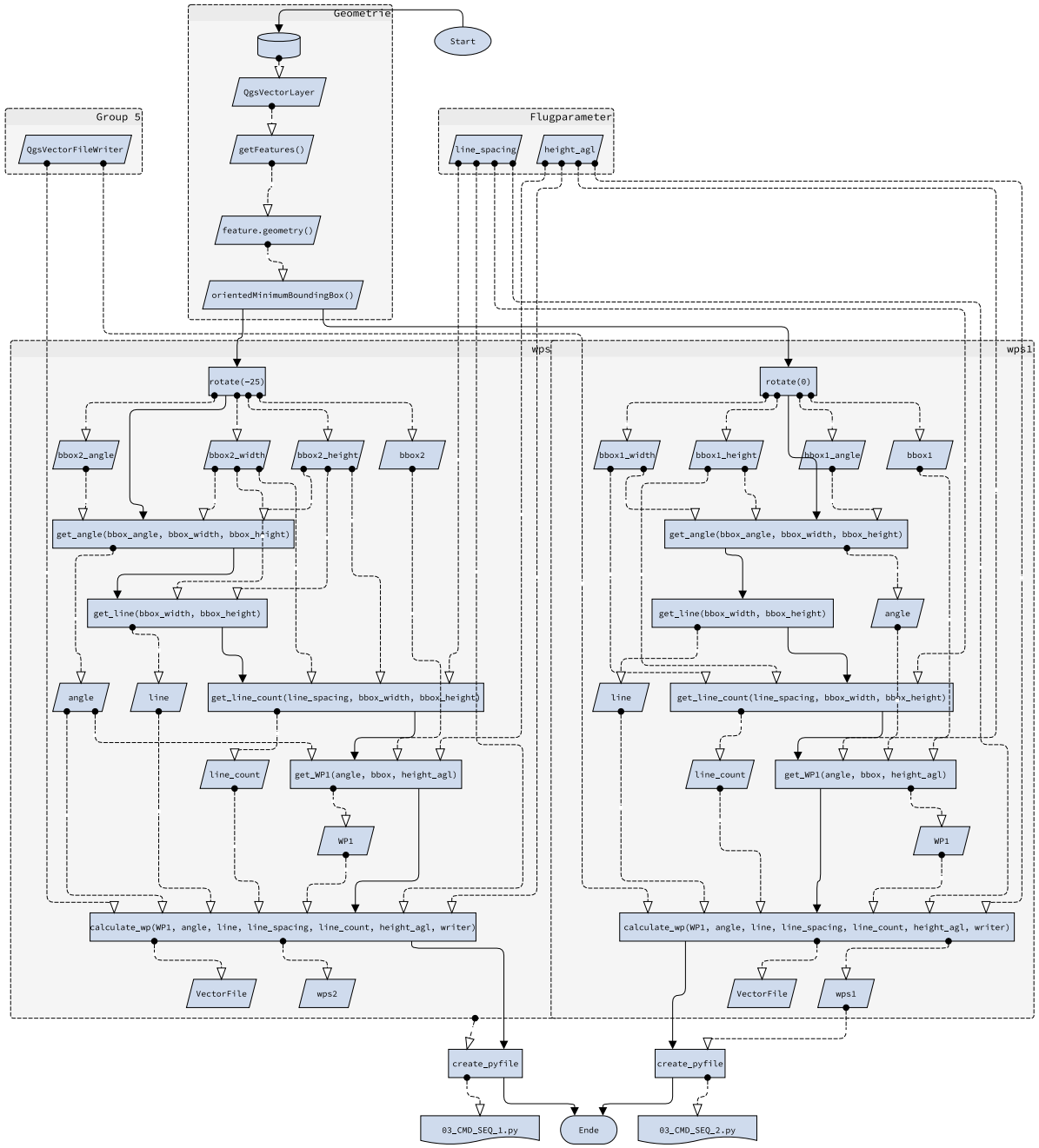


Abbildung 7: Ablaufschema der Wegpunkte-Erstellung. Durchgezogene Graphen: Prozessflüsse, gestrichelte Graphen: Datenflüsse. Quelle: Eigene Darstellung.

Nach der Festlegung des zu überfliegenden Bereichs (AOI), des Flugachsenabstands A und der Höhe über Grund h sind alle Voraussetzungen zur Erstellung der Flugregler-Kommando-Sequenzen gegeben.

Das AOI ist dafür als Geoobjekt (feature) in einem Geodatenlayer vorzuhalten oder in einem solchen zu erstellen. Innerhalb eines Python-Scripts[63] wird dieser Layer als QgsVectorLayer geladen, das betreffende Geoobjekt über die Methode getFeatures() als Objekt der Klasse QgsFeature instanziiert und von diesem Objekt über die Methode geometry() die zugehörige Geometrie als Objekt der Klasse QgsGeometry abgefragt. Die Klasse QgsGeometry bietet die Methode orientedMinimumBoundingBox(), die das nach Flächeninhalt kleinste Rechteck zurückgibt, das die Geometrie vollständig umschließt. Die weiter unten zu definierenden Flugachsen verlaufen parallel zu den Längsseiten des Rechtecks. Im Gegensatz zum Rechteck, das von der Methode boundingBox() zurückgegeben wird, ist dieses Rechteck in der Regel verdreht[64] und weist einen geringeren Flächeninhalt auf. Damit wird sichergestellt, dass die beim Überflug zurückgelegte Strecke minimal ist. Neben der eigentlichen Rechteckgeometrie liefert die Methode auch dessen Verdrehwinkel (bbox_angle) sowie die Höhe (bbox_width) und Breite (bbox_height). Von dieser Rechteckgeometrie (bbox_1) wird eine Kopie angelegt, die um 25° verdreht wird (bbox_2).

bbox1 ist die Fläche, die in Flugmission 1 und bbox2 ist die Fläche, die in Flugmission 2 überflogen wird. Die Verdrehung der 2. Flugroute um 25° zielt darauf ab, die abzulichtenden Geoobjekte aus einer 2. Perspektive zu erfassen, was im SfM-Prozess das Aufkommen von Nodata-Bereichen reduziert und die Kamera-Kalibrierung unterstützt.

Die Methode orientedMinimumBoundingBox() liefert zwar den Verdrehwinkel des Rechtecks, es ist aber unklar, ob sich dieser auf die Höhen- oder Breitenseite des Rechtecks bezieht. Daher wird mittels der Funktion get_angle() vom Winkelgrad 90 der Verdrehwinkel subtrahiert und das Ergebnis zurückgegeben, wenn die Rechteck-Höhe länger ist als die Rechteck-Weite. Andernfalls wird der ursprüngliche Winkelgrad zurückgegeben.

Als nächstes wird ermittelt, welche Rechteckseite die längere ist, die Breite oder die Höhe. (Zur längeren Rechteckseite werden die Flugachsen verlaufen.) Die Funktion

[63] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/02_CREATE_WAYPOINTS_AND_CMD_SEQ.py

[64] Die Rechteckseiten verlaufen nicht parallel zu den Achsen des Koordinatensystems.

get_line() gibt die längere Rechteckseite zurück. Danach wird mittels der Funktion get_line_count(), aufgrund des vorgegebenen Flugachsenabstands sowie der Rechteckgeometrie, ermittelt, wie viele Flugachsen für das AOI anzulegen sind.

Danach wird über die Funktion get_WP1() der erste Wegpunkt der Flugmission ermittelt. Dazu wird, abhängig davon ob die Rechteck-Längsseite näher an der x- oder y-Koordinatenachse ausgerichtet ist, der Geometriestützpunkt mit dem minimalen x- bzw. y-Koordinatenwert ermittelt und der betreffende Punkt als Objekt der Klasse QgsPointXY zurückgegeben.

Mit dem 1. Wegpunkt, dem Verdrehwinkel der langen Flugachsen, der Länge der langen Flugachsen, dem Flugachsenabstand, der Anzahl der Flugachsen und der Flughöhe über Grund sind nun alle erforderlichen Größen bekannt um die weiteren Wegpunkte zu berechnen. Dies erfolgt mittels der Funktion calculate_wp().

Dabei wird zur Berechnung von Wegpunkt 2 die Längsseitenlänge mit dem Kosinus bzw. Sinus des Verdrehwinkels multipliziert und das Ergebnis mit den Koordinaten des bereits ermittelten Wegpunkts 1 addiert. Zur Berechnung von Wegpunkt 3 wird der Flugachsenabstand mit dem Kosinus bzw. Sinus des Verdrehwinkels multipliziert und das Ergebnis mit den Koordinaten des bereits ermittelten Wegpunkts 2 verrechnet. Dieser Vorgang setzt sich solange fort bis die Anzahl berechneter Wegpunkte den line_count-Wert erreicht hat. Jeder berechnete Wegpunkt wird als Objekt der Klasse QgsPointXY in einer Dictionary (wps1) gespeichtert, die der Rückgabewert der Funktion calculate_wp() ist. Die Punktgeometrie eines Wegpunkts wird innerhalb der Funktion in das RBZ WGS84 transformiert, da der Pixhawk solche Koordinaten benötigt.

Beiläufig werden aus den Punktgeometrien noch QgsFeature-Objekte erstellt, die einem QgsVectorFileWriter angehängt werden und durch diesen als ESRI-shapefile dateibasiert gespeichert werden.

Der obige Vorgang wird ein 2. mal mit dem um 25° verdrehten Rechteck durchgeführt. Im Ergebnis erhält man eine 2. Dictionary mit Wegpunkten (wps2).

Auf Basis der Dictionary mit den Wegpunkten (wps1 bzw. wps2), in der die Wegpunktnummern die Schlüssel und die zugehörigen Koordinatenpaare die Werte bilden, wird nun ein Python-Script erstellt, das Kommandos enthält, die unmittelbar vor

Flugdurchführung über die 433-MHz-Datenfunkverbindung, kodiert im MAV-Link-Protokoll, an den Flugregler gesendet und dort im EEPROM gespeichert wird. Dazu wird eine Zeichenkette im String Format erstellt und als Datei im py-Format gespeichert. Die Zeichenkette enthält den Python-Programm-Code. Im Programm-Code werden vorab die Python-Module dronekit und pymavlink importiert. Dann wird mittels der Funktion connect des Moduls dronekit über eine serielle Schnittstelle (hier: COM3), dem 433-MHz-Transceiver (ground-end), der Luftschnittstelle, dem 433-MHz-Transceiver (air-end), der UART-Schnittstelle eine MAV-Link-Verbindung aufgebaut. Nach Verbindungsherstellung werden vorab alle etwaigen noch im EEPROM gespeicherten Kommandos gelöscht. Es wird nun eine Liste mit den hochzuladenden neuen Kommandos angelegt. Dieser wird zuerst das Kommando MAV_CMD_DO_CHANGE_SPEED angehängt, das vor dem Start die horizontale Geschwindigkeit auf 2 m/s setzt. Als nächstes wird das Kommando MAV_CMD_NAV_TAKEOFF angehängt, das die Drohne veranlasst auf die oben festgelegte Flughöhe über Grund aufzusteigen. Es folgt das Kommando MAV_CMD_DO_REPEAT_RELAY, welches Relais 1 für 3s einschaltet, damit Stromkreis 1 der Kamera schließt und damit die Kamera einschaltet. Das folgende Kommando MAV_CMD_DO_SET_SERVO verdreht die nach vorne ausgerichtete Kamera um 90° in Richtung Erdmassenschwerpunkt. Es werden nun in einer for-Schleife und der richtigen Abfolge die in der Dictionary wps1 bzw wps2 vorgehaltenen Wegpunkte über das Kommando MAV_CMD_NAV_WAYPOINT in die Kommandosequenz integriert. In dieser Schleife gibt es eine Bedingung: Nach dem 1. und vor dem 2. Wegpunkt wird das Kommando MAV_CMD_DO_REPEAT_RELAY integriert, das den Flugregler veranlasst Relais 2 für 1s einzuschalten, damit Stromkreis 2 an der Kamera zu schließen und somit die Videoaufnahme zu starten.

Nachdem das letzte MAV_CMD_NAV_WAYPOINT-Kommando, also der letzte Wegpunkt in die Sequenz aufgenommen wurde, wird erneut das Kommando MAV_CMD_DO_REPEAT_RELAY gegeben, das Relais 2 für 1s einschaltet, damit Stromkreis 2 an der Kamera schließt und somit die Videoaufnahme stoppt. Es folgt das Kommando MAV_CMD_CONDITION_DELAY, das dafür sorgt, dass alle folgenden DO-Kommandos erst in 4s ausgeführt werden. Nach diesen 4s folgt ein weiteres Mal das Kommando MAV_CMD_DO_REPEAT_RELAY, das Relais 1 für 3s einschaltet, damit Stromkreis 1 an der Kamera schließt und somit die Kamera

ausschaltet. Es folgen die Kommandos MAV_CMD_DO_SET_SERVO, das die Kamera wieder nach vorne ausrichtet, MAV_CMD_NAV_RETURN_TO_LAUNCH, das die Drohne dazu veranlasst an die beim Booten gespeicherte Startposition zurückzufliegen und MAV_CMD_NAV_LAND, wodurch die Drohne an der Startposition den Landevorgang durchführt.

Die Kommandosequenz[65] wird über die am Anfang des Scripts eingerichtete MAV-Link-Verbindung in den EEPROM des Flugreglers geladen.

Es wird eine weitere Kommandosequenz mit den 2. Wegpunkten (wps2) erstellt.

[65] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/03_EXEC_CMD_SEQ_1.py

3.2.2 Flugdurchführung

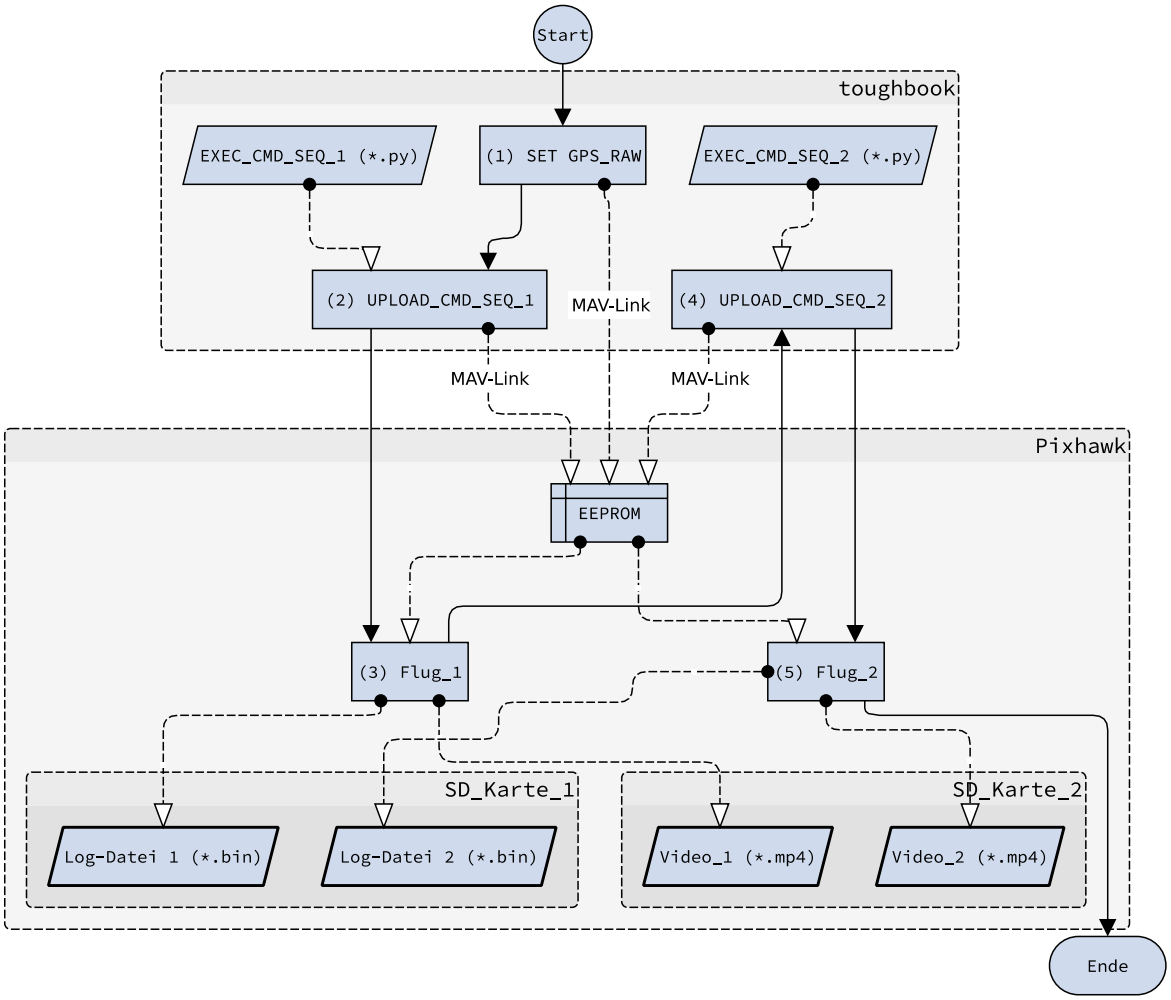


Abbildung 8: Ablaufschema der Flugdurchführung. Quelle: Eigene Darstellung.

Nach abgeschlossener Flugvorbereitung kann der Flug durchgeführt werden sobald günstige Wetterbedingungen gegeben sind. Unmittelbar vor der Flugdurchführung und am Einsatzort sind sämtliche nach den anerkannten Regeln der Technik gebotenen Sicherheitschecks am UAS durchzuführen.

Der UAV-Stromkreis wird an die Stromquelle (LIPO-Akku) angeschlossen und damit alle Komponenten unter Strom gesetzt sowie der Flight-Controller gebootet. Falls noch nicht erfolgt ist spätestens jetzt der Flugregler-Parameter GPS_RAW_DATE auf den Wert 2 zu setzen (s. 3.1.1.3) und der Flugregler zu rebooten.

Auf dem Panasonic-Toughbook wird innerhalb der Python-Laufzeitumgebung als erstes das Script 03_EXEC_CMD_SEQ_1.py[66] ausgeführt und damit die Kommandosequenz für Flugmission 1 mittels MAV-Link in den EEPROM geladen.

Es werden nun die Motoren des UAV über die Fernsteuerung gestartet (Status: ARMED). Dabei ist darauf zu achten, dass sich der Flugregler zunächst noch nicht im Flugmodus AUTO sondern in den Flugmodi STABLE oder LOITER befindet. Sobald die Motoren laufen, wird das UAV per Fernsteuerung in den Modus AUTO versetzt und damit die im EEPROM gespeicherte Kommando-Sequenz gestartet.

Anmerkung: Bei der Durchführung von Flugmission Nr. 1 kam es bei Wegpunkt 6 (Endpunkt der 3. Flugachse und 2 min. nach Beginn der Videoaufnahme) zu einem Absturz des UAV, sodass nur aus diesem Zeitraum GNSS-Messdaten und Video-Bildmaterial erfasst sind. Flugmission Nr. 2 konnten nicht durchgeführt werden.

3.3 Log-Datei-Konvertierung

Die Flugdaten werden im Flugregler in einem nicht menschenlesbaren Binär-Format auf der SD-Karte dateibasiert abgespeichert. Unter der Anwendung Mission-Planner wird die Binär-Datei in eine Ascii-Datei konvertiert.

3.4 GNSS-Datenaufbereitung

Der GNSS-Empfänger berechnet auf Basis der Pseudoentfernungen und Phasenverschiebungen die Positionslösung und stellt diese dem Flugregler zur Verfügung. Im Flugprotokoll wird die Positionslösung (X-, Y- und Z-Koordinaten [WGS84]) zum Zeitpunkt t_s der Systemlaufzeit über den Parameter GPS wiedergegeben. In dieser Form können sie bereits an das Bildmaterial gebunden werden (s. 3.4.1). Jedoch ist bei den Geokoordinaten von Ungenauigkeiten auszugehen.

Für die Nachkorrektur von GNSS-Daten über das PPK-Verfahren sind die gemessenen Ephemeriden und Navigationsdaten der Satelliten erforderlich deren Signale vom GNSS-Empfänger empfangen und zu Positionslösungen verrechnet

[66] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/03_EXEC_CMD_SEQ_1.py

wurden. Diese Rohdaten interessieren Nutzer bzw. auf Positionslösungen zurückgreifende Schnittstellen, wie z. B. den Flugregler, i. d. R. nicht. Beim PPK-Verfahren werden aber aus den Beobachtungsdaten des Rovers (hier: der GNSS-Empfänger am UAV) und den Beobachtungs- sowie Navigationsdaten der Kontrollstation (hier: die virtuelle SAPOS-Referenzstation) die Positionslösungen von Grund auf neu berechnet (s. 3.4.2).

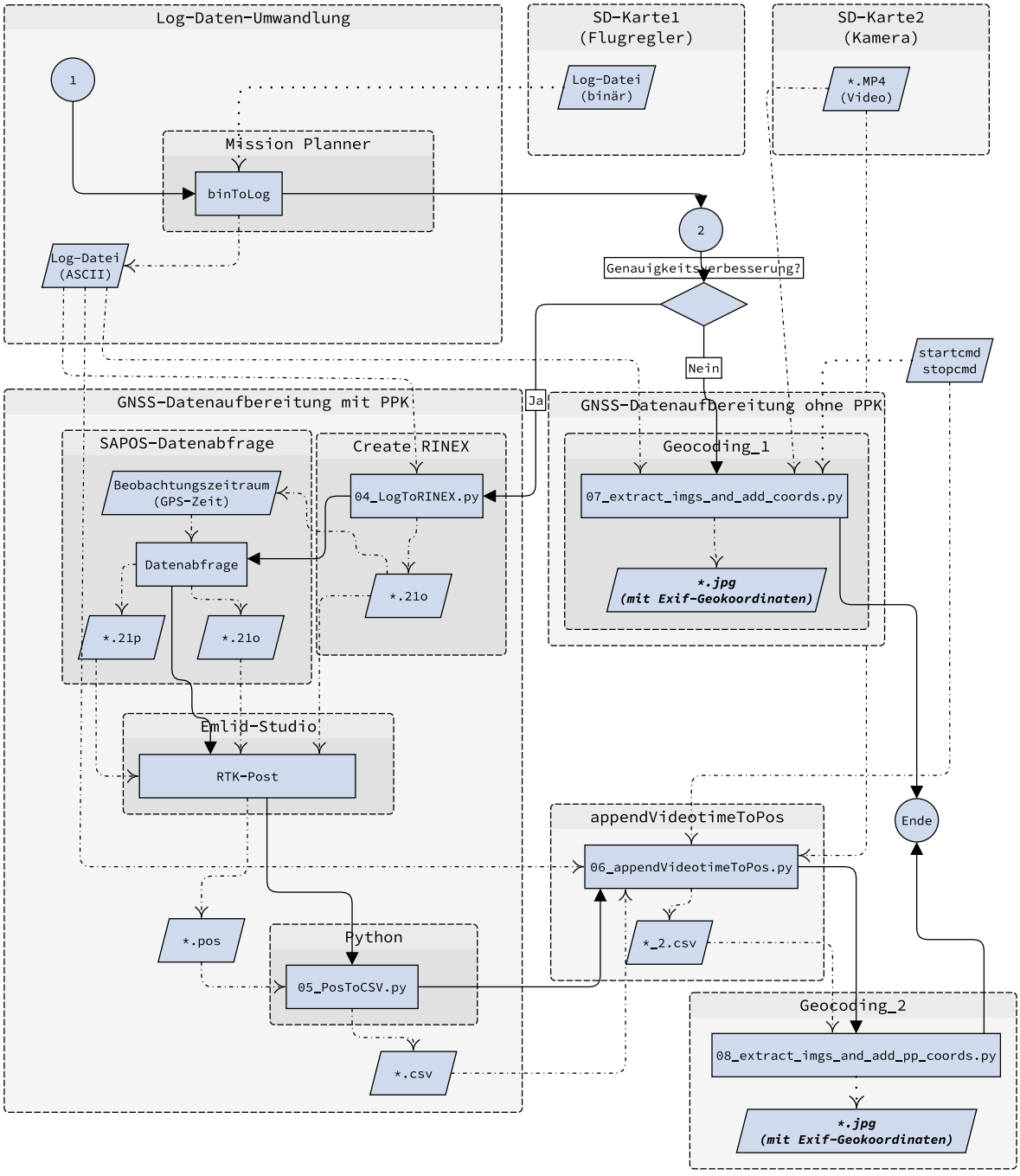


Abbildung 9: Ablaufschema GNSS-Datenaufbereitung und Einzelbildextraktion. Quelle: Eigene Darstellung.

3.4.1 GNSS-Datenextraktion ohne Postprocessing

Mit dem Einschalten der Motoren des UAV (Betriebszustand: ARMED) beginnt der Flugregler mit der Aufzeichnung von Flugparameterwerten[67] in die auf einer SD-Karte persistent gespeicherte Log-Datei. Der Parameter GPS[68] wird im zeitlichen Abstand von ca. 200 ms aufgezeichnet. Bevor die Kamera eingeschaltet und die Videoaufnahme gestartet wird vergeht Zeit, in der für den hiesigen Verwendungszweck uninteressante GNSS-Daten aufgezeichnet werden. Erst die GNSS-Daten ab dem Zeitpunkt des Beginns der Videoaufnahme werden benötigt. Dieser Zeitpunkt fällt mit der Ausführung des Kommandos (bzw. des Ereignisses) DO_REPEAT_RELAY[69] zusammen, welches wiederum in der Log-Datei über den Parameter MSG[70] und den Wert „*Mission: 7 RepeatRelay*“ protokolliert ist.

Mit Ausführung eines Python-Scripts[71] (s. Anhang I 07) geht folgendes vor: Die ASCII-Log-Datei wird geöffnet und Zeile für Zeile gelesen. Wenn die Zeile den Parameter MSG und den Wert „Mission: 7 RepeatRelay“ enthält wird die zugehörige Systemlaufzeit (in µs) in der Variable starttime gespeichert und die boolesche Variable lock auf false gesetzt. Letzteres bewirkt, dass beim Iterieren der ab dieser Stelle folgenden Log-Datei-Zeilen, Zeilen mit dem Parameter GPS ausgewertet werden. Bei der Auswertung der GPS-Parameter-Zeile passiert folgendes: Vom Zeitstempel (in µs) der GPS-Meldung wird der Zeitwert der Variablen starttime subtrahiert. Der Ergebniswert wird von Mikrosekunden in das Zeitformat Sekunden und Millisekunden umgerechnet (*microsecondsToSecondsMilliseconds*) und in der Variablen time gespeichert. Der jeweilige GPS-Parameterwert ist damit mit der Laufzeit t_v der Videoaufzeichnung synchronisiert. Die über den GPS-Parameter aufgezeichneten

[67] https://ardupilot.org/copter/docs/logmessages.html
(ArduPilot Dev Team 2021)

[68] https://ardupilot.org/copter/docs/logmessages.html#gps
(ArduPilot Dev Team 2021)

[69] https://ardupilot.org/copter/docs/mission-command-list.html#do-repeat-relay
(ArduPilot Dev Team 2021)

[70] https://ardupilot.org/copter/docs/logmessages.html#msg
(ArduPilot Dev Team 2021)

[71] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/07_extract_imgs_and_add_coords.py

WGS84 X-, Y- und Z-Koordinaten werden zusammen mit dem videosynchronen Zeitstempel in einer numpy-Matrix zwischengespeichert.

Der Zeitstempel fungiert als Schlüsselwert bei der unter 3.5 folgenden Zusammenführung der Geokoordinaten mit den ebenfalls mit einem Zeitstempel versehenen Einzelbildern.

3.4.2 GNSS-Datenextraktion mit Postprocessing

3.4.2.1 GNSS-Datenextraktion aus Log-Datei

Das UBX-RXM-RAWX-Protokoll teilt sich in einen Header- und Payload-Anteil auf (ublox 2021). Der Header-Anteil wird im Flugschreiber-Protokoll über den Parameter GRXH[72], der Payload-Anteil über den Parameter GRXS[73] wiedergegeben. Anhang IV 13 zeigt hierzu exemplarisch einen Auszug aus dem Flugprotokoll. Die Parameter GRXH und GRXS werden im zeitlichen Abstand von ca. 400 ms aufgezeichnet.

Diese Nachricht enthält die Informationen, die benötigt werden, um eine RINEX 3-Multi-GNSS-Beobachtungsdatei erzeugen zu können. Diese Nachricht enthält Informationen zu Pseudoentfernung, Dopplerfrequenzverschiebung, Trägerphase, Phasenverriegelung und Signalqualität für GNSS-Satelliten, sobald die Signale synchronisiert wurden. Diese Nachricht unterstützt alle aktiven GNSS. (ublox 2021). Die einzelnen Komponenten des Protokolls sind in Anhang IV 20 aufgeführt.

Die zielgerichtete Extraktion der UBX-RXM-RAWX-Nachrichten aus der Log-Datei zur späteren Übertragung in das RINEX-Format erfolgt innerhalb eine Python-Scripts[74] durch Ausführung der Funktion extract_GRXH_and_GRXS_from_logfile der Python-Klasse LogFile[75] (s. 10). Vor Ausführung dieser Funktion wird zuerst dem Konstruktor der Klasse LogFile die zu lesende Log-Datei als einziger Parameter übergeben. Der Konstruktor öffnet die Log-Datei mit Leserecht, liest alle Zeilen der Log-Datei und speichert die Zeilen als Elemente (Datentyp: string) in einer Liste als klasseneigenes

[72] https://ardupilot.org/copter/docs/logmessages.html#grxh

[73] https://ardupilot.org/copter/docs/logmessages.html#grxs

[74] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/04_LogToRINEX.py

[75] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/modules/log_file.py

Attribut namens log_read ab. Die klasseneigene Funktion extract_GRXH_and_GRXS_from_logfile benötigt dieses Attribut, die eingelesenen Zeilen, als obligaten Parameter.

extract_GRXH_and_GRXS_from_logfile erstellt eine Dictionary, wobei die den Log-Messages zugeordneten Systemzeiten (TimeUS) die Schlüssel bilden, denen die GRXH- und GRXS-Nachrichten als Werte zugewiesen werden. Einer Systemzeit, also einem Schlüssel, ist mindestens 1 GRXH- und 1 GRXS-Nachricht zugeordnet (Kardinalität: 1:n). Einer Systemzeit wird daher in der Dictionary eine Liste zugeordnet, der mehrere GRXH- bzw. GRXS-Nachrichten hinzugefügt werden können. Das erste Element der Liste ist immer eine GRXH-Nachricht, gefolgt von keiner, einer oder mehreren GRXS-Nachrichten[76]. Die GRXH- und GRXS-messages bestehen jeweils aus mehreren Elemente, die in der Zeichenkette durch Kommata getrennt sind. Die Zeichenkette wird anhand der Kommata in als string gespeicherte Listenelemente aufgetrennt. Es liegt also eine Verschachtelung zweier Listen vor. extract_GRXH_and_GRXS_from_logfile wird im Kontext des Python--Scripts 04_LogToRINEX[77] (s. Anhang I 04) aufgerufen und gibt die Dictionary an das aufrufende Programm zurück.

3.4.2.2 Konvertierung ins RINEX-Format

Die als Python-Dictionary aus der Log-Datei extrahierten GNSS-Rohdaten mit zugeordneter Systemzeit werden anschließend im Kontext des Python-Scripts 04_LogToRINEX[78] (s. Anhang I 04) zusammen mit dem Pfad der zu erstellenden RINEX-Beobachtungs-Datei als Parameter an den Konstruktor der Klasse RinexFile[79] (s. Anhang II 11) übergeben. Eine RINEX-Datei besteht aus einem Datei-Header und –Body, die innerhalb der Klasse RinexFile mit den Funktionen create_RINEX_header bzw. create_RINEX_body erstellt werden. Der syntaktische Aufbau einer RINEX-Datei

[76] Die Anzahl der GRXS-Nachrichten pro Epoche, die der Anzahl der beobachteten GNSS-Satelliten entspricht, wird in der GRXH-Nachricht über den Parameter *numMeas* angegeben. Für jede Epoche wird eine GRXH-Nachricht ausgegeben.

[77] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/04_LogToRINEX.py

[78] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/04_LogToRINEX.py

[79] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/modules/RINEX_file.py

ist standardisiert und kann versionsabhängig variieren. Die Klasse RinexFile baut die RINEX-Datei gemäß dem aktuellen Standard 3.05.

Der Header enthält Metainformationen zum Dateiinhalt. Im Rahmen dieser Arbeit wird davon ausgegangen, dass das RINEX-Dateiformat, der eingesetzte GNSS-Empfänger einschließlich Antenne und der Dateiurheber unverändert bleiben. Der Großteil des Headerinhalts wird daher als konstante Zeichenkette in der Funktion create_RINEX_header angelegt. Lediglich die Attribute DATE (Erstellungsdatum/ -uhrzeit) APPROX POSITION XYZ (mittlere Position des Empfängers) und TIME OF FIRST OBS/ TIME OF LAST OBS (Zeitpunkte der ersten und letzten Satellitenbeobachtung die in der Datei protokolliert ist) gehen als variable Größen in die Zeichenkette des Headers ein.

Die Funktion create_RINEX_body der Klasse RinexFile gibt den RINEX-Body als Zeichenkette zurück, die zu Beginn des Funktionsablaufs 0 Zeichen umfasst um im Funktionsablauf um Zeichenkettenelemente erweitert wird, die durch Zeilenumbrüche voneinander getrennt sind. Dazu werden die Werte der unter 3.4.2.1 mit extract_GRXH_and_GRXS_from_logfile erstellten Dictionary[80], die chronologisch sequenzierten und in sich als Listenelemente untertrennten GRXH- bzw. GRXS-messages, in einer for-Schleife einzeln abgerufen. Innerhalb der Funktion werden Indexe als Konstanten zum zielgerichteten Aufrufen von UBX-RXM-RAWX-Protokoll-Elementen (z.B. GPS-Woche, Pseudorange, Trägerphase). die als Python-Listenelemente gespeichert sind, definiert.

Die Liste der GRXH-Protokoll-Elemente setzt sich aus der GPS-Wochenzeit (0), der GPS-Woche (1), dem Schaltsekundenwert (2), der Anzahl der in der betreffenden Epoche beobachteten Satelliten (3) und einem Empfängerstatus-Flag-Wert (4) zusammen. Die GPS-Woche, GPS-Wochenzeit und die Schaltsekunde werden aufgerufen, in Ganzzahl- bzw. Fließkommawerte gewandelt und der Funktion weeks_and_seconds_to_utc aus der Klasse TimeFormatConverter[81] (s. Anhang II 12) übergeben, die die Zeitgrößen in UTC-Zeit (Datum und Uhrzeit in

[80] Die Schlüssel, also die den GRXH- bzw. GRXS-messages zugeordneten Systemzeiten, sind in diesem Zusammenhang irrelevant.

[81] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/modules/time_format_converter.py

Mikrosekundengenauigkeit) zurückgibt. Diese UTC-Zeit wird zusammen mit der Anzahl der beobachteten Satelliten zu einer record-identifier-Zeile verkettet.

Dieser record-identifier-Zeile folgen nun die Beobachtungen (GRXS-messages). Die Liste der GRXS-Protokoll-Elemente setzt sich aus dem Pseudorange-Messwert (0), dem Trägerphasen-Messwert (1), dem Doppler-Frequenzverschiebungs-Messwert (2), dem GNSS-Identifikator (3), dem Satellitenidentifikator (4), dem Signal-Rausch-Verhältnis (7) und einigen weiteren, im gegebenen Kontext irrelevanten, Elementen zusammen. Pro Satellitenbeobachtung gibt es eine GRXS-message. Die GRXS-Elemente werden einzeln abgerufen, so umformatiert, dass sie den Formatvorgaben des RINEX-Standards entsprechen und abschließend verkettet. Im Anhang III 15 ist eine Beispiel-RINEX-Datei, wie sie als Ergebnis des hier beschriebenen Verfahrens vorliegt, aufgeführt.

3.4.2.3 SAPOS-Datenabfrage

Vorab wird ein Account eingerichtet. Mit den Zugangsdaten loggt man sich ein und navigiert in den Bereich Startseite > GPPS-Shop > Postprocessing-Daten > VRS. Dort wird zur Berechnung einer virtuellen Referenzstation (VRS) der relevante Beobachtungszeitraum in GPS-Zeit[82], das Messintervall[83] (1 s) und die Position (Koordinaten im EPSG 4936)[84] der VRS angegeben. Es wird die RINEX-Format-Version angegeben, in der die VRS-Beobachtungen abgespeichert werden sollen und ein Name für die VRS vergeben, nach dem die Ausgabedateien benannt werden. Es werden dann eine Beobachtungsdatei (*.21o) und eine Navigationsnachrichtendatei (*.21p) erstellt und zum Download bereitgestellt.

[82] Sommerzeit: MEZ-2 Stunde, Winter: MEZ-1 Stunden.

[83] Folgende Eingaben (s) sind möglich: 1,2,5,10,15,20,30,60

[84] Die VRS-Koordinate kann hier abgegriffen werden:
https://epsg.io/map#srs=4936&x=6378137.00&y=0.00&z=2&layer=streets

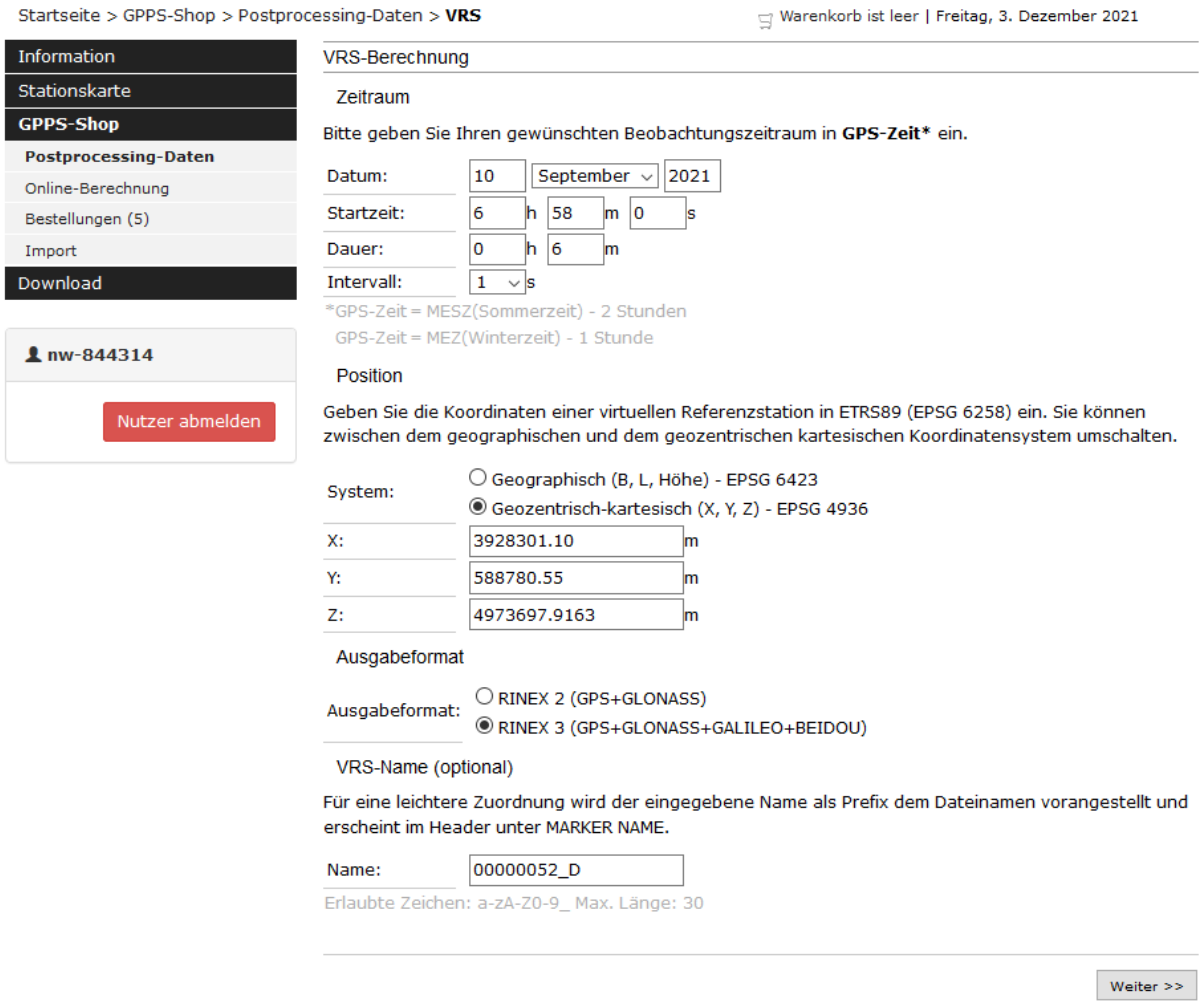


Abbildung 10: Maske zur Eingabe von Daten zur Berechnung von Messdaten einer virtuellen Referenzstation (VRS) in der Web-Anwendung SAPOS-NRW.

3.4.2.4 PPK mit Emlid-Studio

Dem PPK-Verfahren unter Emlid-Studio wird die unter 3.4.2.2 erstellte RINEX-Beobachtungsdatei (*.21o) des Rovers (GNSS-Empfänger am UAS), sowie die Beobachtungsdatei (*.21o) und die RINEX-Navigationsnachricht (*.21p) der unter 3.4.2.3 berechneten virtuellen Referenzstation zugeführt. Als erstes wird aus den drei verfügbaren Positionierungsmodi Single, Kinematic und Static Kinematic ausgewählt. Die VRS-Position wird aus dem Header der RINEX-Beobachtungsdatei übernommen. Als Filtertyp wird Forward ausgewählt. Der Parameter Elevation mask wird auf 30 Grad gesetzt (Signale von Satelliten die niediger als 30° über dem Horizont stehen werden ausgelassen). Die berechneten Positionslösungen werden in einer ASCII-Datei mit der Endung pos gespeichert (s. Anhang III 0).

Bei nur ca. 3 % Prozent der GNSS-Messwerte konnte ein fix, also eine endgültige Auflösung der Phasenmehrdeutigkeit und damit hochgenaue Positionsbestimmung erzielt werden (Bei einem fix-Wert bewegt sich die Genauigkeit im Bereich der Wellenlänge des ausgewerteten Signals, also im cm-Bereich (Bauer 2018)). Bei 90% der GNSS-Messwerte konnten noch keine abschließende Auflösung der Phasenmehrdeutigkeiten erzielt werden (Status: float). Dieser Wert beruht auf statistischen Annahmen, die beim PPK-Verfahren errechnet werden (innerhalb eines um die aktuelle Positionsschätzung gezogenen Kreises, die wahrscheinlichste Position). Die Genauigkeit von Float-Werten liegt im Allgemeinen zwischen 40 und 20 cm (Bauer 2018).

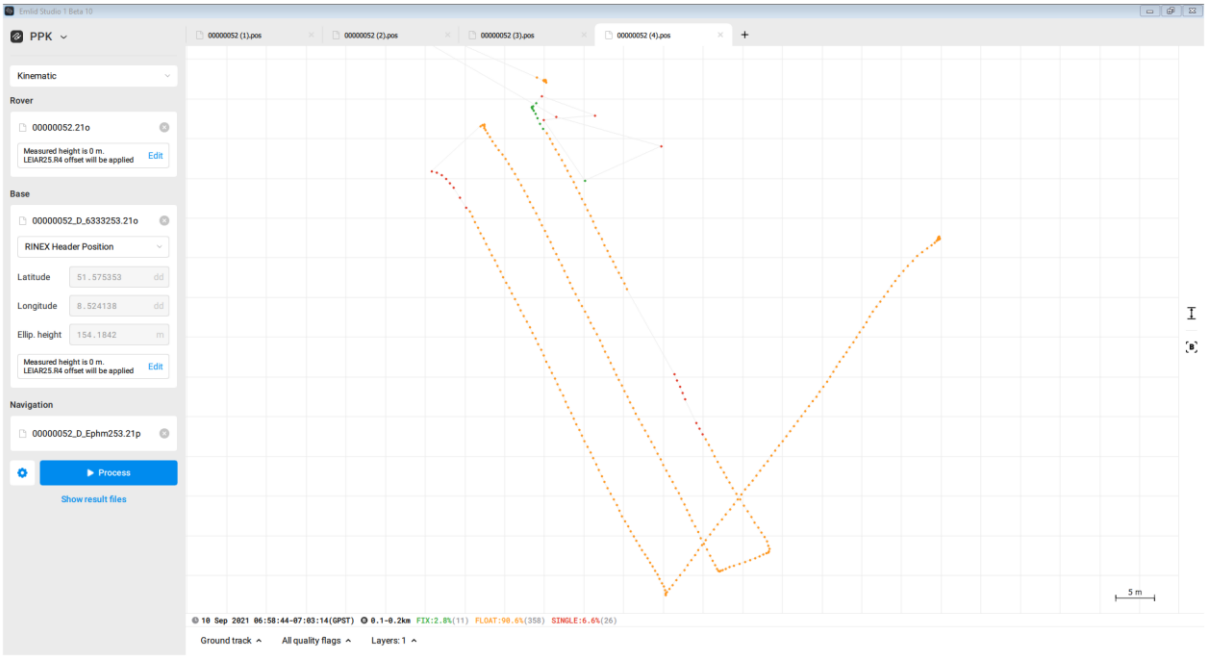


Abbildung 11: Screenshot der Durchführung der PPK-Anwendung auf der grafischen Oberfläche von Emlid Studio mit Darstellung der nachkorrigierten Geokoordinaten. Bei nur ca. 3 % Prozent der GNSS-Messwerte konnte ein fix, also eine endgültige Auflösung der Phasenmehrdeutigkeit und damit hochgenaue Positionsbestimmung erzielt werden. Bei 90% der GNSS-Messwerte konnten noch keine abschließende Auflösung der Phasenmehrdeutigkeiten erzielt werden (Status: float).

3.4.2.5 Umformatierung der Positionslösung

Die Elemente der Positionslösung, insbesondere die Zeitangabe (GPST), die Koordinaten (latitude(deg), longitude(deg) und height(m)) und der Wert der Positionierungsgüte (Q), sind in der vorliegenden Form in den Folgeprozessen schwer

zugreifbar. Daher werden die Datensätze in einem Zwischenschritt durch das Python-Script PosToCSV[85] (s. Anhang I 05) in das CSV-Format (s. Anhang III 0) umgeformt.

3.4.2.6 Positionslösung um Videozeitstempel ergänzen

Die nachberechneten Positionslösungen (Geokoordinaten mit GPS-Zeitangabe) werden daraufhin mit der Videozeit in Beziehung gesetzt. Dazu wird auf die Log-Datei und die CSV-Datei mit den Positionslösungen zugegriffen. Die Datensätze der CSV-Datei werden um die Attribute videotime und videotime(sec) erweitert und in einer neuen CSV-Datei gespeichert. Die nachberechneten Geokoordinaten und die korrespondierenden Videozeiten liegen dann in einem leicht zugreifbaren Format beim folgenden Standbildexport und der Standbildgeoreferenzierung vor. Dieser Vorgang wird über das Python-Script 06_appendVideotimeToPos[86] (s. Anhang I 06) abgewickelt.

Zuerst werden die Start- (t_{vs}) und End- (t_{ve}) Zeitpunkte der Videoaufzeichnung in der Systemlaufzeit ermittelt. t_{v0} (Beginn der Videoaufnahme in der Systemzeit) liegt mit Δt = 1s hinter dem Zeitpunkt t_{sd1}, an dem die Do-Anweisung REPEAT_RELAY im Flugregler ausgeführt wurde (REPEAT_RELAY löste als Ereignis die Einschaltung des 2. Relais für die Dauer von 1 Sekunde aus. Durch die 1-sekündige Relaisschaltung wurde in der Kamera der Schaltkreis zur Aktivierung der Videoaufnahme geschlossen.). Das Ereignis REPEAT_RELAY wird im Flugprotokoll mit Angabe der Systemzeit t_{sd1} bei Ausführung und unter der Bezeichnung „Mission: <Nr.> RepeatRelay“[87] aufgezeichnet.

$$t_{vs} = t_{sd1} + \Delta t \qquad \text{Formel 7}$$

$$t_{ve} = t_{sd2} + \Delta t \qquad \text{Formel 8}$$

[85] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/05_PosToCSV.py

[86] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/06_appendVideotimeToPos.py

[87] <Nr.> ist variabel und ein Platzhalter für die Position in der Reihenfolge der Fluganweisungen. Im Python-Script ist die Bezeichnung der Variablen startcmd zugewiesen.

Mit t_{vs} ist der Beginn der Videoaufnahme mit der Systemlaufzeit des Flugreglers synchronisiert.

t_{ve} (Stopp der Videoaufnahme in der Systemzeit) liegt mit $\Delta t = 1s$ hinter dem Zeitpunkt t_{s2}, an dem die Do-Anweisung "Reached command #12"[88] augeführt wurde.

t_{vs} und t_{ve} sind Voraussetzung für die Inbezugsetzung der Videolaufzeit (0s + x, x: Videolaufzeit) mit den in der Systemlaufzeit gespeicherten GNSS-Informationen. Dazu wird der neuen CSV-Datei zunächst eine neue Kopfzeile hinzugefügt, die der alten Kopfzeile gleicht aber zusätzlich die Attribute videotime und videotime(sec) erhält.

Dann werden nacheinander die Datensätze der Quell-CSV-Datei eingelesen und die Werte der Attribute Datum (z. B. "2021/09/10") und Zeitpunkt (z. B. "06:58:43.800") der Positionslösung extrahiert. Datum und Uhrzeit werden weiter in ihre Einzelkomponenten (Jahr, Monat, Tag, Stunde, Minute, Sekunde, Mikrosekunde) zerlegt und mit der Python-Funktion utc_to_week_and_second (s. 12) in die Zeitgrößen GPS-Woche und GPS-Sekunden umgerechnet.

Vorher wurde einmalig über die Funktion map_gpstime_to_timeUS[89] (s. Anhang II 10) eine Dictionary erstellt, in der die Zeitstempel der GPS-Meldungen in Systemzeit (Schlüssel) und die in der GPS-Meldung angegebenen GPS-Sekunden (Wert) in Bezug gesetzt wurden. Aus dieser Dictionary wird nun versucht mit dem soeben berechneten GPS-Sekunden der Positionslösung als Schlüssel die zugehörige Systemzeit der GPS-Messung abzufragen. Sollte es keinen Treffer geben (z. B. weil im PPK-Verfahren eine Messepoche nicht verarbeitet werden konnte) wird den Attributen videotime und videotime(sec) als Nicht-Wert jeweils NULL zugewiesen. Gibt es einen Treffer wird die erfolgreich abgefragte Systemzeit der GPS-Meldung t_{gps} in die Videozeitlaufzeit (timeVideo) umgerechnet. Dazu wird t_{vs} von t_{gps} subtrahiert. Das in der Einheit Mikrosekunde vorliegende Ergebnis timeVideo wird danach mit der Funktion microseconds_to_clocktime[90] (s. Anhang II 12) in die UTC-Zeitgrößen Minuten, Sekunden und Millisekunden (Attribut: videotime) und zusätzlich in

[88] Im Python-Script ist die Bezeichnung der Variablen stopcmd zugewiesen.

[89] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/modules/log_file.py

[90] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/modules/time_format_converter.py

Dezimalsekunden (Attribut: videotime(sec)) umgerechnet und dem CSV-Datensatz hinzugefügt.

3.5 Einzelbildextraktion und Geokodierung

Es werden innerhalb eines Python-Scripts[91] aus der soeben erstellten CSV-Datei mit den Positionslösungen aus jedem Datensatz die nachberechneten Koordinaten (Breite, Länge, Höhe) als Werte und die zugehörigen Videozeiten als Schlüssel entnommen und in einer Dictionary namens timestamps_pos_dictio zwischengespeichert. Dabei werden nur solche Positionslösungen mit einer Güte von 1 (=fix) oder 2 (=float) übernommen.

Danach wird mit dem externen Python-Modul openCV ein erstes Mal die Videodatei geöffnet, als Eigenschaft des Videos der fps-Wert abgefragt, und in einer Schleife jedes Einzelbild des Videos nacheinander abgerufen. Dabei wird jedes Einzelbild über eine Zählvariable (counter) nummeriert und durch den konstanten fps-Wert dividiert. Damit ist der Videozeitpunkt jedes Einzelbilds berechnet. Er wird einer Liste namens timestamps_vid_list hinzugefügt.

Danach erfolgt ein Schleifendurlauf der Elemente der Dictionay timestamps_pos_dictio. In jedem Iterationsschritt erfolgt ein untergeordneter Schleifendurchlauf durch die Liste timestamps_vid_list mit den Videozeitpunkten der Standbilder. Im untergeordneten Schleifendurchlauf wird der Differenzbetrag aus dem aktuell aus timestamps_pos_dictio aufgerufenen Videozeitpunkt aus der Positionslösung und dem aus timestamps_vid_list aufgerufenem Videozeitpunkt des Standbilds berechnet. Falls der Differenzbetrag einen vorher festgelegten Schwellenwert (z. B. 0,041) unterschreitet weisen Positionslösung und Standbild denselben Videozeitpunkt auf und passen daher zueinander. Einer zuvor angelegten neuen Dictionary wird in dem Fall als Schlüssel der Videozeitpunkt des aktuell iterierten Einzelbilds und als Wert die Liste mit den korrigierten Koordinaten hinzugefügt.

[91] https://github.com/BS-Code-01/UAS-Code-Rep/blob/main/08_extract_imgs_and_add_pp_coords.py

Danach wird mit dem externen Python-Modul openCV ein zweites Mal die Videodatei geöffnet und in einer Schleife jedes Einzelbild des Videos nacheinander abgerufen. Es wird erneut in oben beschriebener Weise für jedes Standbild der Zeitstempel berechnet. Die Standbilder werden nun in 2 Fälle unterteilt: Standbilder für die Geokoordinaten aus der Positionslösung vorliegen und solche bei den diese fehlen. Dazu wird überprüft, ob der Standbildvideozeitpunkt in timestamps_vid_dictio als Schlüssel vorhanden ist, also eine Positionslösung für diesen Videozeitpunkt vorliegt. Falls nicht, bleibt das zu exportierende Bild ohne Geoinformation bzw. es wird nicht exportiert. Andernfalls werden die vorliegenden Positionsdaten in die Exif-Daten des zu exportierenden Standbildes geschrieben. Die Exif-Daten werden mithilfe des Python-Moduls piexif formatiert und als Byte-Sequenz im Arbeitsspeicher zwischengespeichert. Das betreffende Standbild wird zunächst ohne Exif-Daten über die Funktion imwrite des Python-Moduls openCV auf der Festplatte gespeichert. Da eine Zuweisung von Exif-Daten mithilfe der Funktion imwrite nicht möglich ist, wird die soeben gespeicherte jpg-Datei über das Untermodul Image des Python-Moduls Pillow (PIL) geöffnet und mit der zugehörigen Funktion save erneut unter gleichem Namen gespeichert. Der Funktion save werden dabei als optionalem Parameter die im Arbeitsspeicher hinterlegten exif-Daten zugewiesen. Nach Abschluss der Schleifendurchläufe wird die Videodatei geschlossen und die geokodierten Standbilder liegen im Verzeichnis images zur Verarbeitung unter ODM bereit.

3.6 ODM-Prozessierung

Die über die Exif-Metadaten mit Geokoordinaten ausgestatteten Einzelbilder sind nun soweit vorbereitet, der Anwendung OpenDroneMap als Daten-Input zugeführt werden zu können. Dazu wird vorab ein Wurzelverzeichnis im Dateisystem angelegt (z. B. D:\UAV-Befliegungen\04_ODM) und darin ein Unterverzeichnis namens images, in das die vorbereiteten Bilddateien zu kopieren sind. Anschließend wird über die Kommandozeile das in Abbildung 12 wiedergegebene Kommando abgesetzt.

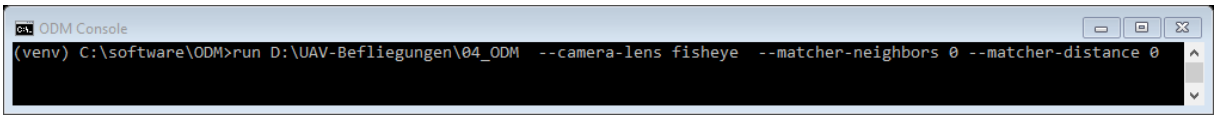


Abbildung 12: Screenshot des Kommandos zum Starten des ODM-Prozesses.

Der Ablauf der dann folgenden Prozesskette, in dessen Kern die unter 2.5 beschriebenen SfM-Algorithmen ablaufen, kann durch Beigabe zusätzlicher Argumente[92] beeinflusst werden. Im Rahmen dieser Arbeit wurde beim Parameter camera lens der Wert fisheye übergeben, um dem Algorithmus bei der Berechnung der Kameraparameter vorab engere Grenzen vorzugeben. Außerdem wurden bei den Parametern matcher-neighbors und matcher-distance der Wert auf 0 gesetzt. Dies hat zur Folge, dass der feature-matching-Algorithmus (Suche nach homologen Bildpunkten), sich über den gesamten Bilddatensatz erstreckt und nicht lediglich auf einen bestimmten Nachbarschaftsbereich[93] (z. B. innerhalb von 5 m). Somit werden Inkosistenzen, wie Höhensprünge, im Endergebnis vermieden. Am Ende des Prozesses erhält man als Ausgabe eine 3D-Punktwolke im LAS-Format (s. Abbildung 13) und ein digitales Orthofoto im TIF-Format (s. Abbildung 14).

[92] https://docs.opendronemap.org/arguments/

[93] Solche Suchradien setzten voraus, dass in den Exif-Metadaten der Bilddateien Geokoordinaten hinterlegt sind.

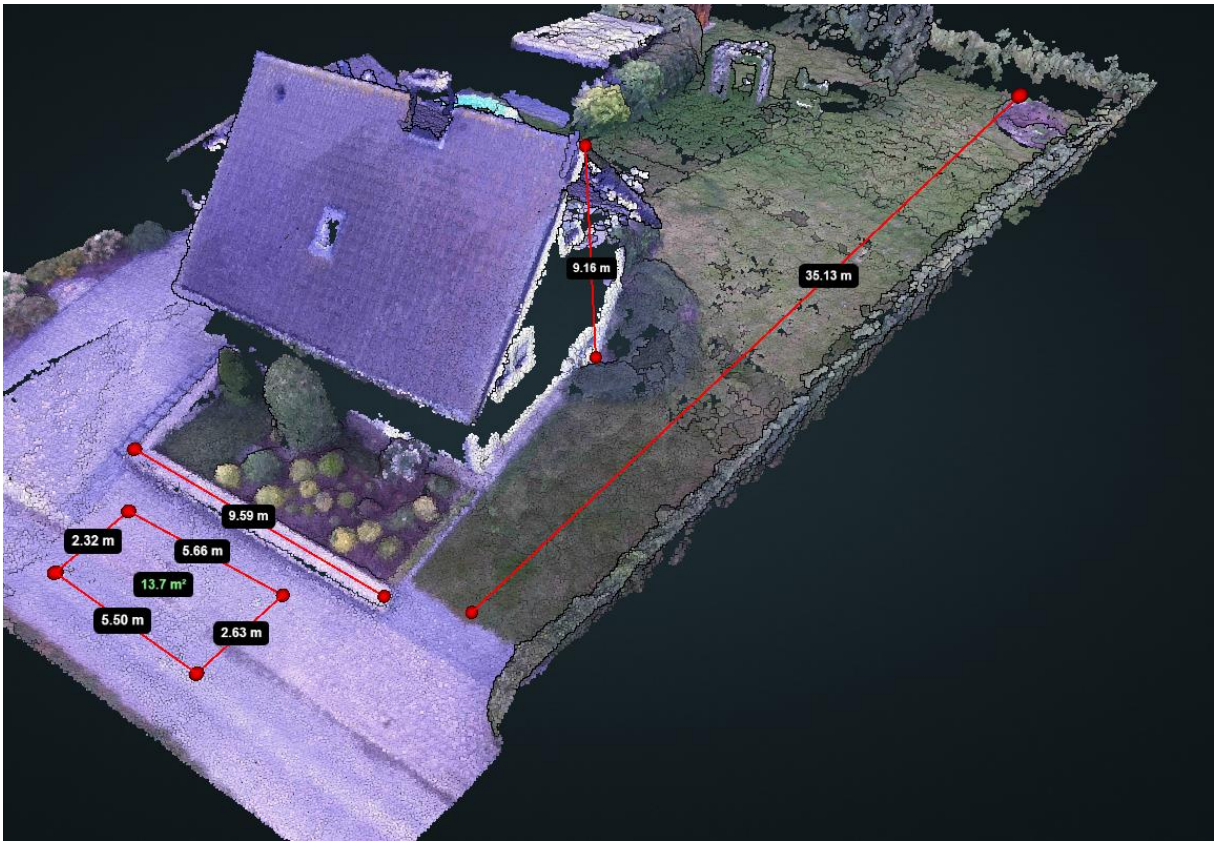


Abbildung 13: Die durch ODM erzeugte 3D-Punktwolke. Visualisiert mit zusätzlichen metrischen Angaben (Strecke, Höhe, Flächengröße) über die Anwendung Potree.

Abbildung 14: Das durch ODM erzeugte digitale Orthofoto.

3.7 Positionsgenauigkeitsmessung

Die Positionsgenauigkeit der im LAS-Format vorliegenden 3D-Punktwolke sowie des im TIF-Formats vorliegenden digitalen Orthofotos werden nun durch Vergleich mit Referenzdaten gemessen. Von Interesse ist dabei die Beantwortung der Frage, ob und inwieweit sich eine direkte Georeferenzierung auf Basis von postprozessierten GNSS-Daten im Vergleich zu einer direkten Georeferenzierung auf Basis von nicht postprozessierten GNSS-Daten auf die Positionsgenauigkeit auswirkt.

3.7.1 Abgleich mit Referenzpunkten

Als Referenzdaten für den Genauigkeitsabgleich der Orthofotos dienen digitale Orthofotos des Anbieters Geobasis NRW[94] mit einer GSD von 10 cm/Pixel (DOP10), die als WMS bereitgestellt werden. Sie weisen eine Lagegenauigkeit von +/- 20 – 30 cm auf.

Zu vergleichen ist die Lagegenauigkeit (X- und Y-Koordinaten) einerseits des Orthofotos, das auf Basis unkorrigierter GNSS-Messdaten direkt georeferenziert wurde (ohne PPK) und andererseits des Orthofotos, das auf Basis korrigierter GNSS-Messdaten direkt georeferenziert wurde (mit PPK). Unter QGIS wird dazu der DOP10-WMS geladen und jeweils Punkt-Vektorlayer für die Referenzpunkte und die Vergleichspunkte auf den zu vergleichenden Orthofotos erstellt.

Es werden von allen drei Orthofotos 10 homologe Punkte, die ortsfeste und unveränderliche Punkte (z. B. Gebäudeecken, Kreuz-Pflasterfugen) repräsentieren, in die Punkt-Vektorlayer übertragen und übereinstimmend durchnummeriert. Anschließend werden die Beträge der Koordinatendifferenzen zwischen einem Referenzpunkt und einem Vergleichspunkt gleicher Nummer berechnet (X- bzw. Y-Koordinatendifferenzen) bzw. der euklidische Abstand mithilfe des Tools Point to Point Distances[95] berechnet.

---

[94] https://www.bezreg-koeln.nrw.de/brk_internet/geobasis/luftbildinformationen/aktuell/digitale_orthophotos/index.html

[95] https://saga-gis.sourceforge.io/saga_tool_doc/7.8.0/shapes_points_3.html

3.7.2 Abgleich durch Punktwolkendifferenz

In Anlehnung an Ahmad Fuad, N.; Yusoff, A. R.; Ismail, Z.; Majid, Z. (2018) sowie Girardeau-Montaut, et al. (2006) wurde zur Feststellung der Positionsgenauigkeit ein Abgleich der erzeugten 3D-Punktwolken mit Referenzpunktwolken vorgenommen.

Als Referenzdaten für den Genauigkeitsabgleich der 3D-Punktwolken dienen Lidar-Punktwolken des Anbieters Geobasis NRW[96]. Diese weisen eine Lagegenauigkeit von +/- 30 cm und eine Höhengenauigkeit von +/- 15 cm (doppelte Standardabweichung) auf. Die Punkteverteilung ist unregelmäßig, bei einer mittleren Dichte von 4 – 10 Pkt/m².

Die zu vergleichenden Punktwolken werden vorab durch das Tool LAS-clip[97] unter QGIS auf ein AOI zugeschnitten, in dem eine vollständige Überschneidung aller 3 Punktwolken gegeben ist und anschließend in der Anwendung CloudCompare geladen.

Mit dem Werkzeug Cloud-to-Cloud-Distance[98] werden nun die Abstände zwischen der Referenzpunktwolke und der zu vergleichenden Punktwolke (und damit die Positionsgenauigkeiten) berechnet[99]. Für jeden Punkt der zu vergleichenden Wolke wird der euklidische Abstand zum nächsten Nachbarn aus der Referenzpunktwolke berechnet. Die Referenzpunktwolke besteht aus 4087 Punkten. Die Vergleichspunktwolken aus 1605563 (ohne PPK-Anwendung) bzw. 1564303 (mit PPK-Anwendung) Punkten. Da die Referenzpunktwolke eine deutlich geringere Dichte aufweist und sich allein dadurch große Abstände zum nächsten Nachbarn auftun, wird als Referenzpunkt kein wahrer Punkt sondern ein virtueller Punkt auf einer Bestanpassungebene[100] herangezogen. Als Parameter r ist der sphärische Radius um den Punkt anzugeben, innerhalb dessen der nächste Nachbar gesucht werden soll (hier: 1 m).

---

[96] https://www.bezreg-koeln.nrw.de/brk_internet/geobasis/hoehenmodelle/3d-messdaten/index.html

[97] https://rapidlasso.com/lastools/lasclip/

[98] https://cloudcompare.org/doc/wiki/index.php?title=Cloud-to-Cloud_Distance

[99] Zur Berechnungsbeschleunigung werden vor der Berechnung die Punktwolken im selben Octree organisiert. Der Zerteilungsgrad des Octrees (Octree level) liegt im vorliegenden Fall bei 6.

[100] interpolierte Fläche nach Methode der kleinsten Quadrate (least square Plane)

4 Ergebnisse

4.1 Positionsgenauigkeit gemäß Abgleich mit Referenzpunkten

Die 10 Vergleichspunkte auf dem Orthofoto (s. Abbildung 15), das auf Basis von nicht nachkorrigierten GNSS-Daten georeferenziert wurde, weisen einen mittleren euklidischen Abstand zu den Referenzpunkten in Höhe von 2,06 m auf (vgl. Tabelle 7). Beim Orthofoto, das auf Basis von nachkorrigierten GNSS-Daten georeferenziert wurde, liegt er bei lediglich 0,43 m. Gemäß der Stichprobe wurde die mittlere Positionsgenauigkeit durch PPK-Anwendung um 1,62 m verbessert. Der maximale euklidische Abstand innerhalb der Stichprobe liegt beim Orthofoto ohne PPK-Anwendung bei 4,37 m und beim Orthofoto mit PPK-Anwendung bei 0,93 m. Der minimale euklidische Abstand liegt beim Orthofoto ohne PPK-Anwendung bei 1,06 m und beim Orthofoto mit PPK-Anwendung bei 0,05 m. Die Standardabweichung liegt beim Orthofoto ohne PPK-Anwendung bei 1,03 m und beim Orthofoto mit PPK-Anwendung bei 0,25 m.

Abbildung 15: Referenz-Orthofoto (DOP10) mit überlagerten Referenz- und Vergleichspunkten. Rote Kreuze: Referenzpunkte, rote Kreise: Vergleichspunkte der Georeferenzierung ohne PPK-Nachkorrektur, blaue Kreise: Vergleichspunkte der Georeferenzierung mit PPK-Nachkorrektur.

Tabelle 7: Mittlere (M), minimale (min) und maximale (max) Positionsabweichungen als Ergebnisse des Abgleichs mit Referenzpunkten. In der letzten Spalte sind die Werte der Positionsgenauigkeitsverbesserung aufgeführt.

	ohne PPK			**mit PPK**			
Ref.-Pkt.-Nr.	ΔX (m)	ΔY (m)	**ΔR (m)**	ΔX (m)	ΔY (m)	**ΔR (m)**	
1	1,01	0,33	**1,06**	0,08	0,66	**0,66**	0,40
2	1,03	1,03	**1,46**	0,11	0,18	**0,21**	1,26
3	1,48	0,12	**1,48**	0,50	0,36	**0,62**	0,86
4	0,33	1,43	**1,47**	0,06	0,93	**0,93**	0,54
6	0,85	1,50	**1,72**	0,15	0,11	**0,19**	1,54
5	0,51	1,26	**1,35**	0,05	0,02	**0,05**	1,30
7	0,88	3,14	**3,26**	0,32	0,13	**0,34**	2,92
8	2,21	3,77	**4,37**	0,22	0,38	**0,43**	3,94
9	1,34	0,57	**1,46**	0,36	0,31	**0,47**	0,98
10	2,08	2,07	**2,93**	0,14	0,42	**0,45**	2,49
M	1,17	1,52	**2,06**	0,20	0,35	**0,43**	**1,62**
Min	0,33	0,12	**1,06**	0,05	0,02	**0,05**	**0,40**
Max	2,21	3,77	**4,37**	0,50	0,93	**0,93**	**3,94**
SD	0,58	1,12	**1,03**	0,14	0,26	**0,25**	**1,08**

4.2 Positionsgenauigkeit gemäß Abgleich durch Punktwolkendifferenz

Die 3D-Punkte, die auf Basis von nicht nachkorrigierten GNSS-Daten georeferenziert wurden, weisen einen mittleren euklidischen Abstand zur Referenzpunktwolke in Höhe von 1,14 m auf (vgl. Tabelle 8 sowie Abbildung 16 und Abbildung 18). Bei den 3D-Punkten, die auf Basis von nachkorrigierten GNSS-Daten georeferenziert wurden, liegt er bei lediglich 0,44 m (vgl. Tabelle 8 sowie Abbildung 17 und Abbildung 19). Gemäß der Stichprobe wurde die mittlere Positionsgenauigkeit durch Anwendung des PPK-Verfahrens um 0,7 m verbessert. Der maximale euklidische Abstand innerhalb der

Stichprobe liegt bei der 3D-Punktwolke ohne PPK-Anwendung bei 4,33 m und bei der 3D-Punktwolke mit PPK-Anwendung bei 3,16 m. Der minimale euklidische Abstand liegt bei der 3D-Punktwolke ohne PPK-Anwendung bei 0,00 m und bei der 3D-Punktwolke mit PPK-Anwendung ebenfalls bei 0,00 m. Die Standardabweichung liegt bei der 3D-Punktwolke ohne PPK-Anwendung bei 0,8 m und bei der 3D-Punktwolke mit PPK-Anwendung bei 0,41 m.

Tabelle 8: Mittlere (M), minimale (min) und maximale (max) Positionsabweichungen der 3D-Punktwolken zu einer Referenzpunktwolke. In der letzten Spalte sind die Werte der Positionsgenauigkeitsverbesserung aufgeführt. ΔR = Radialer (euklidischer Abstand).

	ohne PPK	**mit PPK**	
	ΔR (m)	**ΔR (m)**	**Δ (m)**
M	**1,14**	**0,44**	**0,70**
Min	**0,00**	**0,00**	**0,00**
Max	**4,33**	**3,16**	**1,17**
SD	**0,80**	**0,41**	**0,39**

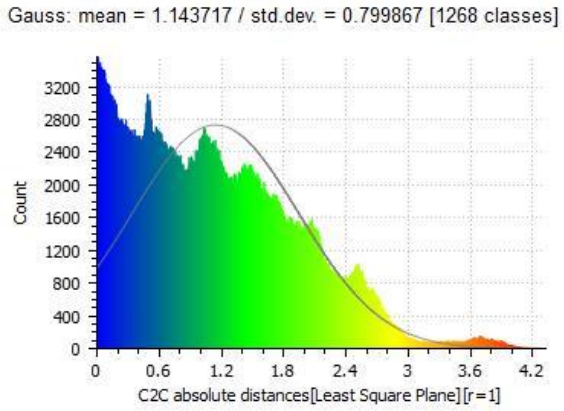


Abbildung 16: Histogramm der Positionsabweichungen der auf Basis von nicht nachkorrigierten GNSS-Daten georeferenzierten 3D-Punkte nach Vergleich mit Referenzdaten. r = Suchradius (hier 1 m) zum nächsten Nachbarn auf der virtuellen Referenzebene.

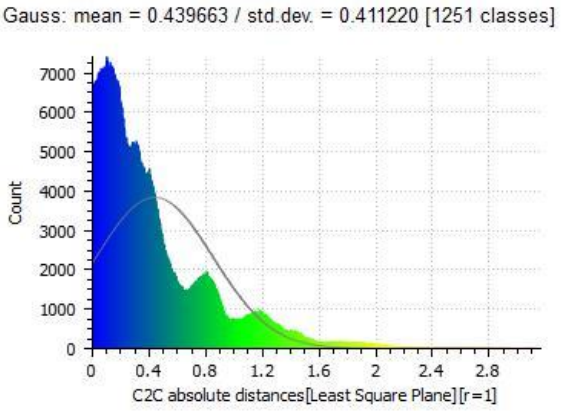


Abbildung 17: Histogramm der Positionsabweichungen der auf Basis von nachkorrigierten GNSS-Daten georeferenzierten 3D-Punkte nach Vergleich mit Referenzdaten. r = Suchradius (hier 1 m) zum nächsten Nachbarn auf der virtuellen Referenzebene.

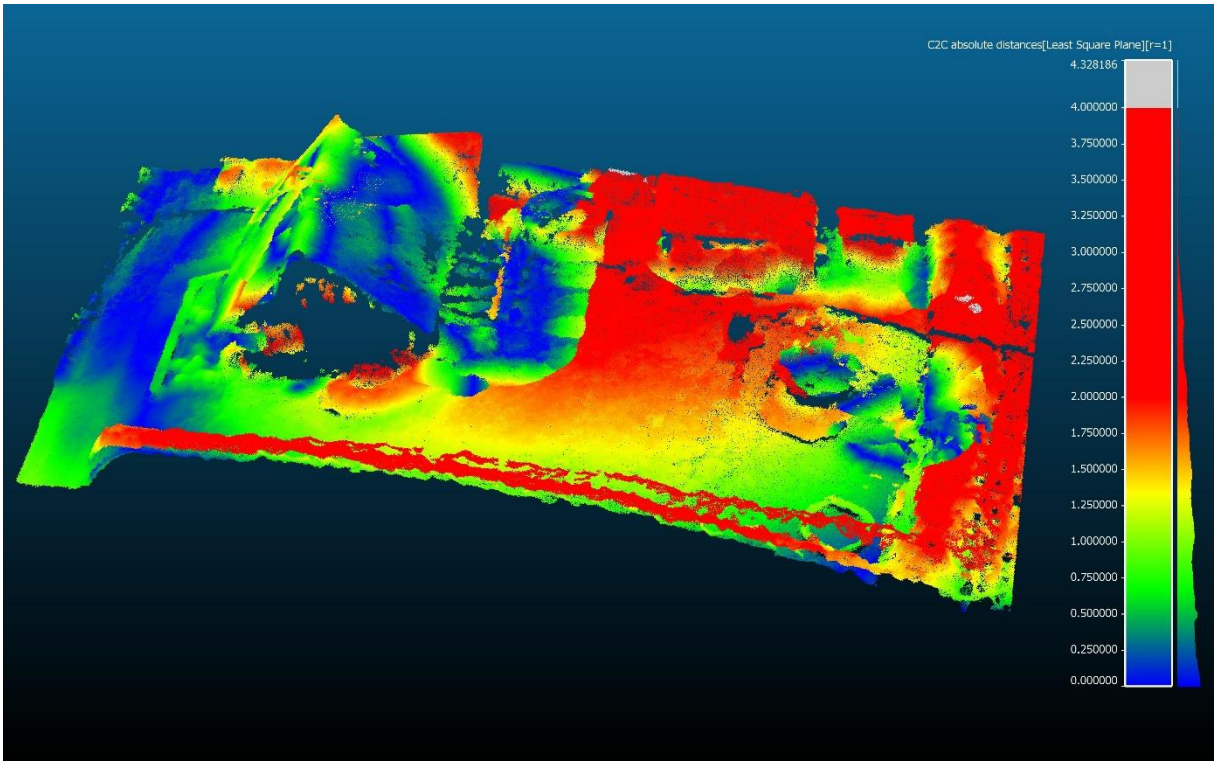


Abbildung 18: Farblich symbolisierte Positionsabweichungen der auf Basis von nicht nachkorrigierten GNSS-Daten georeferenzierten 3D-Punkte nach Vergleich mit Referenzdaten.

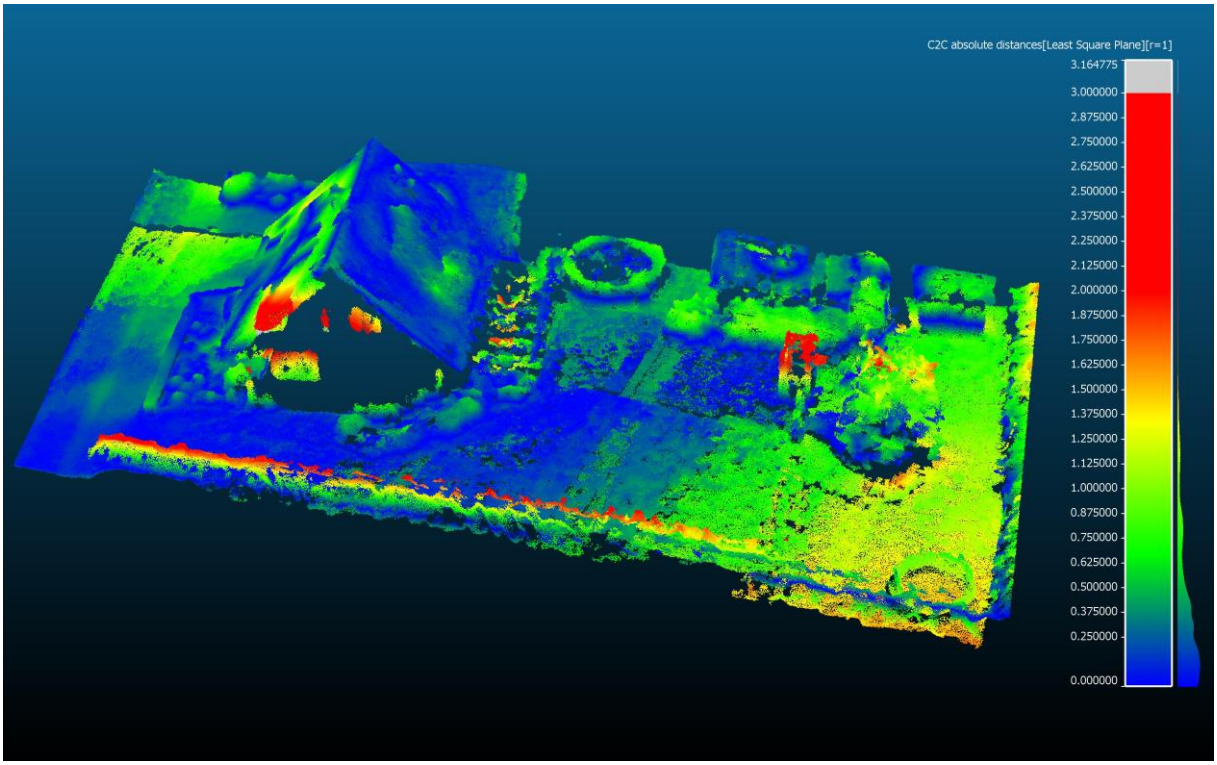


Abbildung 19: Farblich symbolisierte Positionsabweichungen der auf Basis von nachkorrigierten GNSS-Daten georeferenzierten 3D-Punkte nach Vergleich mit Referenzdaten.

5 Fazit und Ausblick

Die Abgleichungen mit den Referenz-Lagepunkten bzw. den Referenzpunktwolken zeigen übereinstimmend, dass bei Durchführung der oben im Detail beschriebenen Teilprozesse mittlere Positionsgenauigkeiten von ca. 0,4 m erreicht werden[101]. Bei Auslassung der PPK-Teilschritte werden mittlere Positionsgenauigkeiten von 2 bis 1,1 m erreicht. Die erzielten Werte bewegen sich damit sehr eng in dem Wertebereich anderer Untersuchungen (s. 1.2) und belegen damit die Praxistauglichkeit des Verfahrens.

Es ist zu berücksichtigen, dass die Referenzdaten lediglich Positionsgenauigkeiten von 0,2 bis 0,3 m aufweisen und sich allein daraus Abstände dieser Größenordnung zu den Neupunkten aus der Befliegung ergeben. Möglicherweise weist daher ein unbekannter Anteil der Neupunkte eine Positionsgenauigkeit besser als 0,4 m auf.

Weitere Verbesserungen bei der direkten Georeferenzierung könnten sich durch eine längere GNSS-Beobachtungszeit (also durch eine Verlängerung der Flugdauer) ergeben, da damit die Auflösung der Trägerphasenmehrdeutigkeiten begünstigt und infolgedessen der Positionswert genauer wird. Im vorliegenden Fall betrug die relativ kurze Zeitdauer der auswertbaren GNSS-Daten lediglich 2 Minuten (vgl. 3.2.2).

Außerdem kann eine stärkere Bildabdeckung des AOI, insbesondere durch einen 2., um 25° verdrehten Überflug (Flugmission 2), das SfM-Ergebnis (keine No-data-Bereiche, mehr features zur Punkt-Rekonstruktion) und damit wiederum die Positionsgenauigkeit der abgebildeten Geoobjekte verbessern.

Auch Latenzen in der Beschaltungstechnik (z. B. Dauer zwischen Event-Zeitpunkt und Einschalten der Videoaufnahme, vgl. 3.4.2.6) beeinflussen die letztliche Positionsgenauigkeit. Genaue Kenntnisse über solche Zeitwerte und deren Verrechnung tragen zu einer Genauigkeitssteigerung bei.

---

[101] Die Anwendung des PPK-Verfahrens führte trotz des hohen Anteils von Messungen mit unaufgelösten Trägerphasenmehrdeutigkeiten (s. 3.1.2.2.4) zu erheblichen Genauigkeitsnachbesserungen.

Bei sehr hohen Genauigkeitsanforderungen (z. B. Vermessungsaufgaben) – das zeigen alle bisherigen Untersuchungen – sollte von einer direkten Georeferenzierung abgesehen werden, bzw. diese nur zusätzlich angewendet werden und stattdessen eine indirekte Georeferenzierung mit hochgenauen Passpunkten vorgenommen werden.

6 Verweise

3D Robotics. *Introducing DroneKit-Python.* 2016. https://dronekit-python.readthedocs.io/en/latest/about/index.html (Zugriff am 21. Januar 2022).

ArduPilot Dev Team. *Arming the motors.* 2021. https://ardupilot.org/copter/docs/arming_the_motors.html (Zugriff am 7. Januar 2022).

—. *Complete Parameter List - GPS_RAW_DATA: Raw data logging.* 2021. https://ardupilot.org/copter/docs/parameters.html#gps-raw-data-raw-data-logging (Zugriff am 7. Januar 2022).

—. *Complete Parameter List - RELAY_PIN: First Relay Pin.* 2021. https://ardupilot.org/copter/docs/parameters.html#gps-raw-data-raw-data-logging (Zugriff am 7. Januar 2022).

—. *GPIOs.* 2021. https://ardupilot.org/copter/docs/common-gpios.html (Zugriff am 7. Januar 2022).

—. *Onboard Message Log Messages - Copter documentation.* 2021. https://ardupilot.org/copter/docs/logmessages.html (Zugriff am 15. Oktober 2021).

—. *Onboard Message Log Messages - GPA: GPS accuracy information.* 2021. https://ardupilot.org/copter/docs/logmessages.html#gpa-gps-accuracy-information (Zugriff am 7. Januar 2022).

—. *Onboard Message Log Messages - GPS: Information received from GNSS systems attached to the autopilot.* 2021. https://ardupilot.org/copter/docs/logmessages.html#gps-information-received-from-gnss-systems-attached-to-the-autopilot (Zugriff am 7. Januar 2022).

—. *Pixhawk Overview.* 7. September 2021. http://ardupilot.org/copter/docs/common-pixhawk-overview.html (Zugriff am 7. Januar 2022).

—. *SiK Telemetry Radio.* 2021. https://ardupilot.org/copter/docs/common-sik-telemetry-radio.html (Zugriff am 21. Januar 2022).

Bauer, Manfred. *Vermessung und Ortung mit Satelliten (7. Auflage).* Berlin: Wichmann, 2018.

Büchi, Roland. *Das große Buch der Drohnen.* Baden-Baden: Verlag für Technik und Handwerk neue Medien GmbH, 2018.

Burdziakowski, Pawel. „EVALUATION OF OPEN DRONE MAP TOOLKIT FOR GEODETIC GRADE AERIAL DRONE MAPPING.“ *17th International Multidisciplinary Scientific GeoConference SGEM2017.* 2017.

Conrad Electronic SE. *RF AC4K 120 4K Actioncam - Bedienungsanleitung.* Hirschau, 2021.

Dinkov, Davis, und Atanas Kitev. „Advantages, disadvantages and applicability of GNSS Post-Processing Kinematic (PPK) method for direct georeferencing of UAV-Images.“ Herausgeber: T. Bandrova, M. Konečný und Marinova S. *Proceedings Vol. 1, 8th International Conference on Cartography and GIS (ISSN: 1314-0604).* Nessebar, Bulgaria, 2020.

Dronecode Project, Inc. *PX4 Autopilot User Guide - 3DR Pixhawk 1 Flight Controller.* 9. September 2021. https://docs.px4.io/master/en/flight_controller/pixhawk.html (Zugriff am 7. Januar 2022).

—. *PX4 Autopilot User Guide - GPS & Compass.* 2020. Oktober 2020. http://docs.px4.io/v1.9.0/en/gps_compass/index.html (Zugriff am 06. Januar 2022).

—. *Using Pymavlink Libraries (mavgen).* 2022. https://mavlink.io/en/mavgen_python/ (Zugriff am 21. Januar 2022).

Drotek. *DP0106 - data sheet.* 2020. https://drotek.gitbook.io/gnss/m8n/dp0106 (Zugriff am 7. Januar 2022).

Eugster, H., S. Nebiker, K. Flückiger, und M. Christen. „PLANNING AND MANAGEMENT OF REAL-TIME GEOSPATIALUAS MISSIONS WITHIN A VIRTUAL GLOBE ENVIRONMENT.“ *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 06. September 2012: 259-263.

Futaba Corporation of America. *T-FHSS Air-2.4 GHz 10J - Manual.* Huntsville, 2018.

GalaxyCore Inc. *1/5'' UXGA CMOS Image Sensor GC2035 - data sheet.* Shanghai, 18. September 2012.

Girardeau-Montaut, D., Michel Roux, Raphaël Marc, und Guillaume Thibault. „CHANGE DETECTION ON POINTS CLOUD DATA ACQUIRED WITH A GROUND LASER SCANNER.“ 2006.

hMatoba. *Piexif documentation.* 2018. https://piexif.readthedocs.io/en/latest/ (Zugriff am 21. Januar 2022).

Iizuka, Kotaro, Takuro Ogura, Yuki Akiyama, Hiroyuki Yamauchi, Takeshi Hashimoto, und Yudai Yamada. „Improving the 3D model accuracy with a post-processing kinematic (PPK) method for UAS surveys.“ *Geocarto International*, 2021: 1-21.

Kahmen, Heribert. *Angewandte Geodäsie: Vermessungskunde.* Berlin: de Gruyter, 2006.

Koller, Dirk. *Vor- und Nachteile integrierter Georeferenzierung bei sUAV-Bildflügen.* 2020.

Koubaa, Anis, Azza Allouch, Maram Alajlan, Yasir Javed, Abdelfettah Belghith, und Mohamed Khalgui. „Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey.“ *IEEE Access.* Bd. 7. 2019.

Kraus, Karl. *Photogrammetrie - Geometrische Informationen aus Photographien und Laserscanneraufnahmen.* Bd. 1. Berlin: de Gruyter, 2004.

Laliberte, Andrea S., Mark A. Goforth, Caitriana M. Steele, und Albert Rango. „Multispectral Remote Sensing from Unmanned Aircraft: Image Processing Workflows and Applications for Rangeland Environments.“ *Remote Sensing*, 22. 11 2011: 2529-2551.

Lundh, Frederik, und Alex Clark. *Pillow.* 2011. https://pillow.readthedocs.io/en/stable/ (Zugriff am 21. Januar 2022).

Nebiker, S., und H. Eugster. „REAL-TIME GEOREGISTRATION OF VIDEO STREAMS FROM MINI OR MICRO UAS USING DIGITAL 3D CITY MODELS.“ São Paulo, Brazil, 21. Juli 2009.

NumPy Developers. *NumPy documentation.* 2022. https://numpy.org/doc/stable/ (Zugriff am 21. Januar 2022).

OpenCV. *OpenCV - Open Source Computer Vision.* 21. Januar 2022. https://docs.opencv.org/4.x/index.html (Zugriff am 21. Januar 2022).

Padró, Joan-Cristian, Francisco-Javier Muñoz, Jordi Planas, and Xavier Pons. "Comparison of four UAV georeferencing methods for environmental monitoring purposes focusing on the combined use with airborne and satellite remote sensing platforms." *International Journal of Applied Earth Observation and Geoinformation* 75 (3 2019): 130–140.

Panasonic Corporation. *Reference Manual Personal Computer Model No. CF-19 Series.* 2008.

Pix4D. *Ground sampling distance (GSD) in photogrammetry.* 2021. https://support.pix4d.com/hc/en-us/articles/202559809-Ground-sampling-distance-GSD-in-photogrammetry (Zugriff am 21. Januar 2022).

Python Software Foundation. *calendar - General calendar-related functions.* 21. Januar 2022. https://docs.python.org/3/library/calendar.html (Zugriff am 21. Januar 2022).

—. *datetime - Basic date and time types.* 21. Januar 2022. https://docs.python.org/3/library/datetime.html (Zugriff am 21. Januar 2022).

—. *math - Mathematical functions.* 21. Januar 2021. https://docs.python.org/3/library/math.html (Zugriff am 21. Januar 2021).

QGIS. *QGIS Python API - core.* 21. Januar 2022. https://qgis.org/pyqgis/master/core/index.html (Zugriff am 21. Januar 2022).

Riecken, Jens, und Enrico Kurtenbach. „Der Satellitenpositionierungsdienst der deutschen Landesvermessung – SAPOS®.“ *zfv – Zeitschrift für Geodäsie, Geoinformation und Landmanagement*, 2017: 293-300.

Schnabel, Patrick. *Elektronik Kompendium - PWM - Pulsweitenmodulation.* 2022. https://www.elektronik-kompendium.de/sites/kom/0401111.htm (Zugriff am 7. Januar 2022).

Schubert, Adrian, David Small, Erich Meier, Nuno Miranda, und Dirk Nuno. „Spaceborne SAR product geolocation accuracy: A Sentinel-1 update.“ *IGARSS*

2014 - 2014 IEEE International Geoscience and Remote Sensing Symposium. Quebec City, QC: Quebec City, QC, 2014. 2675-2678.

STMicroelectronics. „LIS3MDL - data sheet.“ Vers. 6. 2. Mai 2017. https://www.st.com/resource/en/datasheet/lis3mdl.pdf (Zugriff am 2022. Januar 2022).

The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. „Ahmad Fuad, N.; Yusoff, A. R.; Ismail, Z.; Majid, Z.“ 26. Oktober 2018: 11-21.

Toffanin, Piero. *OpenDroneMap: The Missing Guide.* MasseranoLabs LLC, 2019.

u-blox. „NEO/LEA-M8T - data sheet.“ Vers. R05. 2. Juni 2020. https://www.u-blox.com/sites/default/files/NEO-LEA-M8T-FW3_DataSheet_%28UBX-15025193%29.pdf (Zugriff am 7. Januar 2022).

ublox. *u-blox 8 / u-blox M8 Receiver description - Including protocol specification.* 19. August 2021.

Westoby, M. J., J. Brasington, N. F. Glasser, M. J. Hambrey, and J. M. Reynolds. "'Structure-from-Motion' photogrammetry: A low-cost, effective tool for geoscience applications." *Geomorphology* 179 (12 2012): 300–314.

Zhang, He, Emilien Aldana-Jague, François Clapuyt, Florian Wilken, Veerle Vanacker, and Kristof Van Oost. "Evaluating the potential of post-processing kinematic (PPK) georeferencing for UAV-based structure- from-motion (SfM) photogrammetry and surface change detection." *Earth Surface Dynamics* 7 (9 2019): 807–827.

Anhang

Anhang I Python-Code -Scripte

01 CALCULATE_FLIGHTALT_AND_FLIGHTLINESPACING

```
import CalculateFlightPath

c   = 8     #focal_length width (mm)
ImW = 3840  #image width (Pixel)
SW  = 6.16  #sensor_width (mm)
GSD = 5     #ground sampling distance (cm/Pixel)

# Klassenkonstruktor mit SW, ImW, c als Pflichtparameter
cfg = CalculateFlightPath(SW, ImW, c)

# Flughöhe über Grund h
h = cfg.calculate_flight_altitude(GSD)      #Flughöhe über Grund (m)
print(h)

# Querüberdeckung q
q = 75

# Kameraöffnungswinkel γ
gamma = 120

# Flugachsenabstand A
A = cfg.calculate_line_spacing(h, q, gamma) #Flugachsenabstand (m)
print(A)
```

02 CREATE_WAYPOINTS_AND_CMD_SEQ

```
from qgis.core import QgsApplication
from qgis.core import QgsCoordinateReferenceSystem
from qgis.core import QgsCoordinateTransform
from qgis.core import QgsCoordinateTransformContext
from qgis.core import QgsFeature
from qgis.core import QgsFields
from qgis.core import QgsGeometry
from qgis.core import QgsPointXY
from qgis.core import QgsProject
from qgis.core import QgsVectorFileWriter
from qgis.core import QgsVectorLayer
from qgis.core import QgsWkbTypes
import math

# Verdrehwinkel der Flugachsen, wird zurückgegeben.
def get_angle(bbox_angle, bbox_width, bbox_height):
    if bbox_height > bbox_width:
        return 90 - bbox_angle
    else:
        return bbox_angle

# Längere Rechteckseite (=1. Flugachse), wird zurückgegeben.
def get_line(bbox_width, bbox_height):
    if bbox_height > bbox_width:
        return bbox_height
    else:
        return bbox_width

# Anzahl anzulegender Flugachsen, wird zurückgegeben.
def get_line_count(line_spacing, bbox_width, bbox_height):
    if bbox_height > bbox_width:
        return (int(bbox_width/line_spacing)+2) * 2
    else:
        return (int(bbox_height/line_spacing)+2) * 2
```

```
# Koordinaten von WP1, werden zurückgegeben.
def get_WP1(angle, bbox, height_agl):
    WP = QgsPointXY()
    # Festsetzung, ob Rechteck-Längsseite näher an der x-Koordinatenachse (=True)
    # oder y-Koordinatenachse (=False) ausgerichtet ist.
    x_oriented = True
    if x_oriented:
        # Liste zur Speicherung der Geometriestützpunkte.
        pt_list = []
        #Schleife durch die Geometriestützpunkte.
        for vertex in bbox.vertices():
            x = vertex.x() # X-Koordinate von Stützpunkt.
            y = vertex.y() # Y-Koordinate von Stützpunkt.
            pt_list.append([y, x])
        # Punkt mit kleinster Y-Koordinate wird abgerufen.
        wp_coords = min(pt_list)
        WP.setX(wp_coords[1])
        WP.setY(wp_coords[0])
    else:
        # Liste zur Speicherung der Geometriestützpunkte.
        pt_list = []
        for vertex in bbox1.vertices():
            x = vertex.x() # X-Koordinate von Stützpunkt.
            y = vertex.y() # Y-Koordinate von Stützpunkt.
            pt_list.append([x, y])
        # Punkt mit kleinster X-Koordinate wird abgerufen.
        wp_coords = min(pt_list)
        WP.setX(wp_coords[0]) # X-Koordinate von WP1 wird gesetzt.
        WP.setY(wp_coords[1]) # Y-Koordinate von WP1 wird gesetzt.
    return WP

def calculate_wp(WP1, angle, line, line_spacing, line_count, height_agl, writer):
    # Dicitionary zur Speicherung der Wegpunkte
    # (Schlüssel: Bezeichner (z. B. "WP1"), Wert: WGS84-Koordinaten)
    wp_dictio = {}
```

```
x =  WP1.x() # X-Koordinate von Wegpunkt 1.
y =  WP1.y() # Y-Koordinate von Wegpunkt 1.

crsSrc  = QgsCoordinateReferenceSystem(25832) # Quell-Koordinatensystem.
crsDest = QgsCoordinateReferenceSystem(4326)  # Ziel-Koordinatensystem.
xform   = QgsCoordinateTransform(crsSrc, crsDest, QgsProject.instance())

# Koordinatentransformation von EPSG 25832 nach EPSG 4326 (WGS84)
WP1 = xform.transform(WP1, QgsCoordinateTransform.ForwardTransform)
# Wegpunkt 1 wird in Dicitionary gespeichert.
wp_dictio["WP1"] = [WP1.x(), WP1.y()]

wp_counter = 2

while wp_counter <= line_count:
    # Falls WP Ende einer langen Flugachse bildet.
    if wp_counter % 2 == 0:
        # Falls WP Ende einer antiparall verlaufenden Flugachse bildet.
        if wp_counter % 4 == 0:
            x -= math.cos(math.radians(angle)) * line
            y -= math.sin(math.radians(angle)) * line
        else:
            x += math.cos(math.radians(angle)) * line
            y += math.sin(math.radians(angle)) * line
    else:
        x -= math.sin(math.radians(angle)) * line_spacing
        y += math.cos(math.radians(angle)) * line_spacing

    WP = QgsPointXY()
    WP.setX(x) # X-Koordinate von Wegpunkt n wird gesetzt.
    WP.setY(y) # Y-Koordinate von Wegpunkt n wird gesetzt.
    WP_f = QgsFeature()
    WP_f.setGeometry(WP) # Punkt-Vektorgeometrie von WP n wird erstellt.
    writer.addFeature(WP_f)
```

```
        WP_Nr = "WP" + str(wp_counter)
        # Koordinatentransformation von EPSG 25832
        # nach EPSG 4326 (WGS84)
        pt1 = xform.transform(QgsPointXY(x,y))

        # Wegpunkt n wird in Dicitionary gespeichert.
        wp_dictio[WP_Nr] = [pt1.x(), pt1.y()]

        wp_counter += 1

    return wp_dictio

# Pfad zum qgis-Installationsort wird angeben.
QgsApplication.setPrefixPath('C:/software/QGIS_3_16/apps/qgis-ltr', True)

# Ausschalten der QGIS-GUI.
qgs = QgsApplication([], False)

# QGIS-Ressourcen werden geladen.
qgs.initQgis()

# Angabe Flugachsenabstand (A) in m
line_spacing = 10

# Angabe Flughöhe über Grund (h) in m
height_agl   = 25

save_options = QgsVectorFileWriter.SaveVectorOptions()
save_options.driverName   = "ESRI Shapefile"
save_options.fileEncoding = "UTF-8"

writer = QgsVectorFileWriter.create(
    "D:/GIS/Missions/shp/points.shp",
    QgsFields(),
    QgsWkbTypes.Point,
    QgsCoordinateReferenceSystem("EPSG:25832"),
```

```
    QgsCoordinateTransformContext(),
    save_options
)

# AOI (zu überfliegender Bereich) wird als Vektorgeometrie-Layer geladen.
path        = "D:/GIS/Missions/shp/Unrechteck.shp"
baseName    = "Unrechteck"
providerLib = "ogr"
AOI         = QgsVectorLayer(path, baseName, providerLib)

# AOI wird als Objekt der Klasse QgsFeature instanziiert.
features    = AOI.getFeatures()

for feature in features:
    # AOI-Geometrie wird als Objekt der Klasse QgsGeometry abgefragt.
    geom = feature.geometry()

    # Abfrage des nach Flächeninhalt kleinsten minimal umgebenden Rechtecks
    # um die AOI-Geometrie (Fläche, die in Flugmission 1 überflogen wird).
    bbox1        = geom.orientedMinimumBoundingBox()[0] # Rechteckgeometrie
    bbox1_area   = geom.orientedMinimumBoundingBox()[1] # Rechteckfläche
    bbox1_angle  = geom.orientedMinimumBoundingBox()[2] # Verdrehwinkel
    bbox1_width  = geom.orientedMinimumBoundingBox()[3] # Rechteckweite
    bbox1_height = geom.orientedMinimumBoundingBox()[4] # Rechteckhöhe

    # Verdrehwinkel der Flugachsen.
    angle      = get_angle(bbox1_angle, bbox1_width, bbox1_height)
    # Längere Rechteckseite (=1. Flugachse).
    line       = get_line(bbox1_width, bbox1_height)
    # Anzahl anzulegender Flugachsen.
    line_count = get_line_count(line_spacing, bbox1_width, bbox1_height)

    WP1 =  get_WP1(bbox1_angle, bbox1, height_agl)
    WP1_f = QgsFeature()
    # Punkt-Vektorgeometrie von WP 1 wird erstellt.
    WP1_f.setGeometry(QgsGeometry.fromQPointF(WP1.toQPointF()))
```

```
writer.addFeature(WP1_f)

# Alle weiteren Wegpunkte von Flugmission 1 werden berechnet
# und in einer Dictionary (wps1) gespeichert.
wps1 = calculate_wp(WP1, angle, line, line_spacing, line_count, \
height_agl, writer)

# Rotation der BoundingBox um 25°.
geom2    = bbox1
geom2.rotate(-25, geom2.centroid().asPoint())

# Abfrage des nach Flächeninhalt kleinsten minimal umgebenden Rechtecks
# um die verdrehte AOI-Geometrie (Fläche, die in Flugmission 2 überflogen wird)
bbox2        = geom2.orientedMinimumBoundingBox()[0] # Rechteckgeometrie
bbox2_area   = geom2.orientedMinimumBoundingBox()[1] # Rechteckfläche
bbox2_angle  = geom2.orientedMinimumBoundingBox()[2] # Verdrehwinkel
bbox2_width  = geom2.orientedMinimumBoundingBox()[3] # Rechteckweite
bbox2_height = geom2.orientedMinimumBoundingBox()[4] # Rechteckhöhe

# Verdrehwinkel der Flugachsen.
angle      = get_angle(bbox2_angle, bbox2_width, bbox2_height)
# Längere Rechteckseite (=1. Flugachse).
line       = get_line(bbox2_width, bbox2_height)
# Anzahl anzulegender Flugachsen.
line_count = get_line_count(line_spacing, bbox2_width, bbox2_height)

WP2 =  get_WP1(bbox2_angle, bbox2, height_agl)
WP2_f = QgsFeature()
# Punkt-Vektorgeometrie von WP 2 wird erstellt.
WP2_f.setGeometry(QgsGeometry.fromQPointF(WP1.toQPointF()))
writer.addFeature(WP2_f)

# Alle weiteren Wegpunkte von Flugmission 2 werden berechnet
# und in einer Dictionary (wps1) gespeichert.
wps2 = calculate_wp(WP2, angle, line, line_spacing, line_count, \
height_agl, writer)
```

```
del writer
features.close()

print(wps1)
print(wps2)

# Provider- und Layer-Registrierungen werden aus Arbeitsspeicher gelöscht.
qgs.exitQgis()
```

```
#############################################################################
py_file = open("D:/py/03_EXEC_CMD_SEQ_1.py", "w")

s = ""

s += "import dronekit\n"
s += "from dronekit import connect\n"
s += "from pymavlink import mavutil\n\n"

s += "vehicle = dronekit.connect('com3', baud=57600, wait_ready=True)\n"
s += "print('vehicle connected')\n\n"

s += "cmds = vehicle.commands\n"
s += "cmds.clear()\n"
s += "cmds.wait_ready()\n"
s += "print('old commands cleared')\n\n"

s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_CHANGE_SPEED, 0, 0, 0, 2, 0, 0,  0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_NAV_TAKEOFF, \n"
s += "0, 0, 0, 0, 0, 0, 0, {}))\n".format(height_agl)
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0, 0, 1, 3, 0,  0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_SET_SERVO, 0, 0, 7, 1987, 0, 0,  0, 0, 0))\n"

counter = 1
for wp in wps1.values():
    print(wp)
    if counter == 2:
        s += "cmds.add(Command(0, 0, 0, " + \
        "mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, " + \
        "mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0,  1, 1, 1, 0,  0, 0, 0))\n"
        s += "cmds.add(Command(0, 0, 0, " + \
        "mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, " + \
        "mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, "
        "0, 0, 0, 0, 0, 0, {},  {}, {}))\n".format(wp[1], wp[0], height_agl)
    else:
```

```
        s += "cmds.add(Command(0, 0, 0, " + \
        "mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, " + \
        "mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, "
        "0, 0, 0, 0, 0, 0, {},  {}, {}))\n".format(wp[1], wp[0], height_agl))
    counter += 1

s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0, 1, 1, 1, 0,  0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_CONDITION_DELAY, 0, 0, 4, 0, 0, 0,  0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0, 0, 1, 3, 0,  0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_SET_SERVO, 0, 0,  7, 1507, 0, 0,  0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_NAV_RETURN_TO_LAUNCH, 0, 0,  0, 0, 0, 0, 0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_NAV_LAND, 0, 0,  0, 0, 0, 0, 0, 0, 0))\n\n"

s += "cmds.upload()\n"
s += "print('commands uploaded')\n\n"

s += "vehicle.flush()\n"
s += "vehicle.close()\n"
s += "print('vehicle disconnected')"

py_file.write(s)
py_file.close()

################################################################################
py_file = open("D:/py/03_EXEC_CMD_SEQ_2.py", "w")

s = ""

s += "import dronekit\n"
s += "from dronekit import Command\n"
s += "from pymavlink import mavutil\n\n"

s += "vehicle = dronekit.connect('com3', baud=57600, wait_ready=True)\n"
```

```
s += "print('vehicle connected')\n\n"

s += "cmds = vehicle.commands\n"
s += "cmds.clear()\n"
s += "cmds.wait_ready()\n"
s += "print('old commands cleared')\n\n"

s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_CHANGE_SPEED, 0, 0, 0, 2, 0, 0,  0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_NAV_TAKEOFF, \n"
s += "0, 0, 0, 0, 0, 0, 0, {}))\n".format(height_agl)
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0, 0, 1, 3, 0,  0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_SET_SERVO, 0, 0, 7, 1987, 0, 0,  0, 0, 0))\n"

counter = 1
for wp in wps2.values():
    print(wp)
    if counter == 2:
        s += "cmds.add(Command(0, 0, 0, " + \
        "mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, " + \
        "mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0,  1, 1, 1, 0,  0, 0, 0))\n"
        s += "cmds.add(Command(0, 0, 0, " + \
        "mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, " + \
        "mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, "
        "0, 0, 0, 0, 0, 0, {},  {}, {}))\n".format(wp[1], wp[0], height_agl)
    else:
        s += "cmds.add(Command(0, 0, 0, " + \
        "mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, " + \
        "mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, "
        "0, 0, 0, 0, 0, 0, {},  {}, {}))\n".format(wp[1], wp[0], height_agl))
    counter += 1

s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0, 1, 1, 1, 0,  0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_CONDITION_DELAY, 0, 0, 4, 0, 0, 0,  0, 0, 0))\n"
```

```
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0, 0, 1, 3, 0,  0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_DO_SET_SERVO, 0, 0,  7, 1507, 0, 0,  0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_NAV_RETURN_TO_LAUNCH, 0, 0,  0, 0, 0, 0, 0, 0, 0))\n"
s += "cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, "
s += "mavutil.mavlink.MAV_CMD_NAV_LAND, 0, 0,  0, 0, 0, 0, 0, 0, 0))\n\n"

s += "cmds.upload()\n"
s += "print('commands uploaded')\n\n"

s += "vehicle.flush()\n"
s += "vehicle.close()\n"
s += "print('vehicle disconnected')"

py_file.write(s)
py_file.close()
```

03 EXEC_CMD_SEQ

```
import dronekit
from dronekit import Command
from pymavlink import mavutil

# Datenfunkverbindung mittels MAV-Link-Protokoll aufbauen.
vehicle = dronekit.connect('com3', baud=57600, wait_ready=True)
print('vehicle connected')

# Im EEPROM-Speicher des Flugreglers gespeicherte Kommandos  als Liste abrufen.
cmds = vehicle.commands
# Im EEPROM-Speicher des Flugreglers gespeicherte Kommandos löschen.
cmds.clear()
# Warten bis Löschvorgang beendet.
cmds.wait_ready()
print('old commands cleared')

# Festsetzung der horizontalen Geschwindigkeit auf 2 m/s.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_DO_CHANGE_SPEED, 0, 0, 0, 2, 0, 0,  0, 0, 0))

# Takeoff auf 25 m über Grund.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_TAKEOFF, 0, 0, 0, 0, 0, 0, 0, 0, 25))

# Relais 1 für 3s einschalten, damit sich Stromkreis 1 der Kamera schließt
# und damit die Kamera eingeschaltet wird.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0, 0, 1, 3, 0,  0, 0, 0))

# Die nach vorne ausgerichtete Kamera wird um 90°
# in Richtung Erdmassenschwerpunkt verdreht.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_DO_SET_SERVO, 0, 0, 7, 1987, 0, 0,  0, 0, 0))
```

```
# Ersten Wegpunkt anfliegen.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.573462111887764,  8.52425119274193, 25))

# Relais 2 für 1s einschalten, damit sich Stromkreis 2 an der Kamera schliesst
# und somit die Videoaufnahme gestartet wird.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0,  1, 1, 1, 0,  0, 0, 0))

# Restliche Wegpunkte nacheinander anfliegen.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.57361421983368,  8.525672915782398, 25))
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.573702838206344,  8.525648499307819, 25))
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.57355072996008,  8.524226773588104, 25))
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.573639348025765,  8.524202354338211, 25))
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.57379145657235,  8.525624082737185, 25))
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.573880074931694,  8.52559966607049, 25))
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.57372796608475,  8.524177934992244, 25))
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.573816584137106,  8.524153515550204, 25))
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
```

```
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.57396869328438,  8.525575249307735, 25))
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.57405731163038,  8.525550832448921, 25))
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, \
51.57390520218277,  8.524129096012091, 25))

# Relais 2 für 1s einschalten, damit sich Stromkreis 2 an der Kamera schließt
# und somit die Videoaufnahme gestoppt wird.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0, 1, 1, 1, 0,  0, 0, 0))

# alle folgenden DO-Kommandos erst in 4s ausführen.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_CONDITION_DELAY, 0, 0, 4, 0, 0, 0,  0, 0, 0))



# Relais 1 für 3s einschalten, damit sich Stromkreis 1 der Kamera schließt
# und damit die Kamera ausgeschaltet wird.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_DO_REPEAT_RELAY, 0, 0, 0, 1, 3, 0,  0, 0, 0))

# Die mit Objektiv nach unten ausgerichtete Kamera
# wird wieder in Richtung der Flugachse verdreht.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_DO_SET_SERVO, 0, 0,  7, 1507, 0, 0,  0, 0, 0))

# Rückflug zur Startposition.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
mavutil.mavlink.MAV_CMD_NAV_RETURN_TO_LAUNCH, 0, 0,  0, 0, 0, 0, 0, 0, 0))

# Landen.
cmds.add(Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, \
```

```
mavutil.mavlink.MAV_CMD_NAV_LAND, 0, 0,  0, 0, 0, 0, 0, 0, 0))

# Liste mit den Kommandos wird in den EEPROM-Speicher des Flugreglers geladen
# und dort gespeichert.
cmds.upload()
print('commands uploaded')

# Verbindungsdaten werden gelöscht und die Verbindung beendet.
vehicle.flush()
vehicle.close()
print('vehicle disconnected')
```

04 LogToRINEX

```
from moduls.log_file import LogFile
from moduls.RINEX_file import RinexFile

# Verzeichnis in dem die ASCII-Log-Datei gespeichtert ist.
path = "D:/UAV-Befliegungen/temp/"

# Name der Log-Datei mit Dateiformats-Suffix
logfile_name = "00000052.log"

# Ganzer Pfad der Logdatei.
logfile_full_path = path + logfile_name

# Instanziierung des Log-File-Objekts: Einlesen der Log-File-Zeilen
# und Rückgabe der Zeilen in einer Liste
log_file = LogFile(logfile_full_path)

# Ordnet den UBX-RXM-RAWX-Nachrichten in der log_file die System-Zeitstempel
# seit dem Systemstart zu.
GRXH_GRXS_values = log_file.extract_GRXH_and_GRXS_from_logfile()

# Die Ausgabedatei (RINEX-Datei), erhält den gleichen Name wie die Log-Datei
# aber mit dem Dateiformatssuffix ".21o"
RINEX_name = logfile_name.replace(".log", ".21o")

# Ganzer Pfad der Ausgabedatei (RINEX-Datei).
RINEX_full_path = path + RINEX_name

# Es wird eine RINEX-Datei erstellt. Dem Konstruktor wird der Dateiname
# mit vollständigem Pfad und die Dictionary mit den extrahierten
# UBX-RXM-RAWX-Nachrichten übergeben
RINEX = RinexFile(RINEX_full_path, GRXH_GRXS_values)

# Die RINEX-Datei wird in das Dateisystem geschrieben.
RINEX.write_RINEX()
```

```
# Objekte werden gelöscht.
del LogFile, RINEX
```

05 PosToCSV

```
# Dateipfad der pos-Datei
# (Datei mit den nachberechneten Positionen aus dem PPK-Prozess).
pos_file_path = "D:/UAV-Befliegungen/temp/00000052.pos"
# Eingelesener Inhalt der pos-Datei.
pos_file_opened = open(pos_file_path, "r")
# Liste mit den Zeilen der pos-Datei.
pos_file_rows = pos_file_opened.readlines()
# Dateipfad der leeren csv-Datei in die die Positionslösungen übertragen werden.
csv_file_path = "D:/UAV-Befliegungen/temp/00000052.csv"
# Schreibender Zugriff auf die CSV-Datei.
csv_file_opened = open(csv_file_path, "w")
# Sequenzielles Aufrufen der Zeilen der pos-Datei.
for pos_file_row in pos_file_rows:
    # Header Zeilen werden ausgelassen.
    if pos_file_row[0] == "%" and pos_file_row[3:7] != "GPST":
        pass
    else:
        # Die Quell-Elemente der zu erstellenden Kopfzeile sind in
        # der Zeile enthalten, die die folgende Bedingung erfüllt.
        if pos_file_row[3:7] == "GPST":
            # Die Elemente der Quellzeile ab dem 3. Zeichen werden in
            # Einzelelemente zerlegt und dann durch Semikolons getrennt
            # wieder zusammengesetzt.
            new_row = ";".join(pos_file_row[3:].split())
            # Es wird eine Zeichenersetzung in der Kopfzeile vorgenommen.
            new_row = new_row.replace("GPST", "Date;Time")
            print(new_row)
            # Der Kopfzeile wird ein Zeilenumbruch angehängt und danach
            # in die CSV-Datei geschrieben.
            csv_file_opened.write(new_row + "\n")
        else:
            # Die Elemente der Quellzeile werden in Einzelelemente zerlegt
            # und dann durch Semikolons getrennt wieder zusammengesetzt.
            new_row = ";".join(pos_file_row.split())
```

```
            print(new_row)
            # Der Zeile wird ein Zeilenumbruch angehängt und danach
            # in die CSV-Datei geschrieben.
            csv_file_opened.write(new_row + "\n")
# Quell-Datei wird geschlossen.
pos_file_opened.close()
# Ziel-Datei wird gespeichert und geschlossen.
csv_file_opened.close()
```

06 appendVideotimeToPos

```
import datetime
import TimeFormatConverter as TFC
import LogFile

root = "D:/UAV-Befliegungen/02_logs/"

# Instanziierung des LogFile-Objekts
log_file = LogFile(root + "00000052.log")

# Festlegung des Kommandos durch das das Relais geschaltet
# und damit die Videoaufnahme gestartet wird.
startcmd = "Mission: 7 RepeatRelay"
stopcmd  = "Reached command #12"

# Abfrage der Systemzeit seit Boot (TimeUS)
# beim Starten und Stoppen der Videoaufnahme (Rückgabetyp: Tupel).
timeUS_of_video_start_and_stop = \
log_file.get_timeUS_of_video_start_and_stop(startcmd, stopcmd)

# Systemzeit beim Starten der Videoaufnahme (tvs = tsd1 + Δt).
vid_start_time = timeUS_of_video_start_and_stop[0]
# print(vid_start_time)

# Systemzeit beim Stoppen der Videoaufnahme (tve = tsd2 + Δt).
vid_stop_time  = timeUS_of_video_start_and_stop[1]
# print(vid_stop_time)

# Abfrage der Dictionary mit den GPS-Zeitstempeln (GPS-Sekunden)
# als Schlüssel und den TimeUS-Zeitstempeln als Wert.
map_gpstime_to_timeUS = log_file.map_gpstime_to_timeUS()
# print(map_gpstime_to_timeUS)

# Dateiname der Quell-CSV-Datei.
csv_file_read  = root + "00000052.csv"
```

```
# Dateiname der Ziel-CSV-Datei.
csv_file_write = root + "00000052_2.csv"

# new_header = \
# "Date;Time;latitude(deg);longitude(deg);height(m);Q;ns;" + \
# "sdn(m);sde(m);sdu(m);sdne(m);sdeu(m);sdun(m);age(s);ratio;" + \
# "videotime;videotime(sec)\n" # "videotime;videotime(sec)" wird ergänzt

new_header = \
"Date;Time;latitude(deg);longitude(deg);height(m);Q;ns;" + \
"sdn(m);sde(m);sdu(m);sdne(m);sdeu(m);sdun(m);age(s);ratio;" + \
"videotime;videotime(sec);alt_agl(m)\n" # "videotime;videotime(sec)" wird ergänzt

def closest_value(valueset, searchvalue):
    dictio = {}
    for value in valueset:
        diff = abs(searchvalue - int(value))
        dictio[diff] = int(value)
    x = min(list(dictio.keys()))
    # print(dictio[x])
    return dictio[x]

# Öffnen der Ziel-CSV-Datei,
# die eine Erweiterung der Quell-CSV-Datei um die Videolaufzeiten darstellt.
with open(csv_file_write, "w") as csv_file_write_opened:
    # Einfügen einer neuen Kopfzeile in die Ziel-CSV-Datei.
    csv_file_write_opened.write(new_header)
    # Öffnen der Quell-CSV-Datei.
    with open(csv_file_read, "r") as csv_file_read_opened:
        # Schleife durch die Zeilen der Quell-CSV-Datei,
        # außer der Kopfzeile.
        for row in csv_file_read_opened.readlines()[1:]:
            # Zeileninhalt (Datum, Zeit, Koordinaten etc.) werden
            # vereinzelt und als separate Elemente in einer Liste
            # gespeichert.
            row_splitted = row.split(";")
```

```
# Das 1. Listenelement, das Datum (z. B. "2021/09/10")
# wird in seine Teilkomponenten Jahr, Monat und Tag zerlegt
# und als Liste gespeichert.
date  = row_splitted[0].split("/")
year  = int(date[0])
month = int(date[1])
day   = int(date[2])
# Das 2. Listenelement, der Zeitpunkt (z. B. "06:58:43.800")
# wird in seine Teilkomponenten Stunde, Minute und Sekunde
# zerlegt und als Liste gespeichert.
time        = row_splitted[1].split(":")
hour        = int(time[0])
minute      = int(time[1])
sec_float   = time[2].split(".")
sec_int     = int(sec_float[0])
microsec    = int(sec_float[1])

# Umrechnung der obigen Zeitgrößen in die GPS-Zeitgrößen
# GPS-Woche und GPS-Sekunden (Beispiel 2174, 457126.59)
gps_weeks_and_microseconds = TFC.utc_to_week_and_seconds(\
year, month, day, hour, minute, sec_int, microsec)
# Zugriff auf die soeben berechneten GPS-Sekunden.
gpsmicroseconds = gps_weeks_and_microseconds[1]

try:
    # Abfrage des TimeUS-Zeitstempels
    # der dem GPS-Zeitstempel zugeordnet ist.
    timeUS = map_gpstime_to_timeUS[gpsmicroseconds]
    print("success: matching gps-record found")
    print("timeUS: " + str(timeUS))

    # Berechnung des Zeitpunks im Video,
    # an dem die GPS-Koordinate gemessen wurde (tgps - tvs).
    timeVideo = timeUS - vid_start_time
    print("timeVideo: " + str(timeVideo))
    # Umrechnung des Videozeitpunkts (Mikrosekunden)
```

```
        # in ein Tupel aus Minuten, Sekunden und Millisekunden.
        timeVideoUTC =  TFC.microseconds_to_clocktime(timeVideo)
        print("timeVideoUTC: " + str(timeVideoUTC) + "\n")
        timeVid_min  = timeVideoUTC[0]
        timeVid_sec  = timeVideoUTC[1]
        timeVid_msec = timeVideoUTC[2]
        # Umrechnung Mikrosekunden in Dezimalsekunden
        # mit 3 Nachkommastellen
        dsec = round(timeVideo/1000000, 3)
        # Schreiben der Quell-Zeile zzgl. der neuen Zeitangaben
        # in die Ziel-CSV-Datei.
        csv_file_write_opened.write(row.strip("\n") + ";" + \
        str(timeVid_min) + "," + str(timeVid_sec) + "," + \
        # str(timeVid_msec) + ";" + str(dsec) + "\n")
        str(timeVid_msec) + ";" + str(dsec)+ ";" + str(BARO) + "\n")

    except:
        # Falls einer Positionslösung kein GPS-Wert in der
        # Log-Datei zugeordnet werden kann.
        # print("failure: no matching gps-record found")
        csv_file_write_opened.write(row.strip("\n") + \
        ";NULL;NULL;NULL\n")
```

07 extract_imgs_and_add_coords

```
import cv2
import datetime
import math
import numpy as np
import piexif
import TimeFormatConverter as TFC
from PIL import Image

# Ordnet einem Videozeitstempel den nächsten GPS-Wert zu.
def find_nearest(array, a0):
    nearest_val = array[abs(array-a0)==abs(array-a0).min()]
    result = np.where(array == nearest_val[0])
    index = result[0]
    return array[index]

# Wandelt einen dezimalen Gradwert in Grad, Minuten und Sekunden um.
def degToDmsRational(degFloat):
    minFloat = degFloat % 1 * 60
    secFloat = minFloat % 1 * 60
    deg = math.floor(degFloat)
    min = math.floor(minFloat)
    sec = int(secFloat * 100)
    return ((deg, 1), (min, 1), (sec, 100))

# Liefert 'N' (Nordhalbkugel) zurück, wenn Breitengrad >= 0°,
# sonst 'S' (Südhalbkugel).
def ChooseLatRef(degree):
    if degree >= 0:
        return 'N'
    else:
        return 'S'

# Liefert 'E' (Osten) zurück, wenn Längengrad >= 0°, sonst 'W' (Westen).
def ChooseLonRef(degree):
    if degree >= 0:
        return 'E'
    else:
        return 'W'

# Log-Datei wird geöffnet.
log_file = "D:/UAV-Befliegungen/210904/02_logs/00000052.log"
# Numpy-Array zur Speicherung der GNSS-Koordinaten
# und der synchronen Systemlaufzeit
arr = np.empty([0,4])

with open(log_file, "r") as log_file_open:
    lock = True
    # Schleife über die Zeilen der Log-Datei.
    for line in log_file_open.readlines():
       # Falls Zeile mit MSG-Parameter
```

```
        if line[:3] == 'MSG':
            items = line.split(", ")
            # Wenn Kommando mit dem das Video gestartet wird enthalten.
            if "Mission: 7 RepeatRelay" in items[2]:
                # Systemlaufzeit abfragen.
                starttime = items[1]
                lock = False
            # Wenn Kommando mit dem das Video gestoppt wird enthalten.
            if "Reached command #12" in items[2]:
                # Systemlaufzeit abfragen.
                starttime = items[1]
                lock = True

        if not lock:
            # Falls Zeile mit GPS-Parameter
            if line[:3] == 'GPS':
                items = line.split(", ")
                # Umrechnung Mikrosekunden Dezimalsekunden
                time = round(((int(items[1]) - int(starttime))/1000000),9)
                print(time)
                lat  = items[7] #Abfrage X-Koordinate
                lon  = items[8] #Abfrage Y-Koordinate
                alt  = items[9] #Abfrage Z-Koordinate
                secs, lat, lon, alt = \
                float(time), float(lat), float(lon), float(alt)
                # Synchrone Systemlaufzeit und GNSS-Koordinaten
                # werden in Array gespeichert.

                arr = np.append(arr, [[secs, lat, lon, alt]], axis=0)

# Video-Datei wird geöffnet.
vc = cv2.VideoCapture('D:/UAV-Befliegungen/03_videos/MOVI0026.mov')

fps     = vc.get(cv2.CAP_PROP_FPS)
frames  = int(vc.get(cv2.CAP_PROP_FRAME_COUNT))
seconds = frames/fps
width   = vc.get(cv2.CAP_PROP_FRAME_WIDTH)
height  = vc.get(cv2.CAP_PROP_FRAME_HEIGHT)

# Zählvariable: Zur Festlegung der Bildexport-Frequenz
counter = 0

if vc.isOpened():
    # Wenn Video-Datei geöffnet ist, wird auf das nächste Bild des Videos
    # zugegriffen. Falls ein Bild zugreifbar ist, wird ein Tupel mit
    # rval(return value)=True und das beteffende Bild (frame)
    # zurückgegeben, andernfalls ein Tupel mit rval=False und frame=False.
    rval, frame = vc.read()
else:
    # Wenn Video-Datei nicht geöffnet ist.
    rval = False # rval(return value)=False

while rval:
```

```
# s. oben
rval, frame = vc.read()
# Zählvariable wird um 1 erhöht.
counter += 1
# Kommandos nach Verzweigung werden nur bei jedem 10. Bild ausgeführt.
# if counter % 10 == 0:
if counter % 10 == 0:
    # Der Zeitstempel des Bilds wird aus der Division
    # von Bildnummer und Bildrate errechnet.
    timestamp = round(counter/fps, 3)
    print("time stamp current frame: ", timestamp)

    # Anhand des Zeitstempels werden aus dem Numpy-Array
    # die nächstliegenden GNSS-Positionen abgefragt.
    tuple = find_nearest(arr, timestamp)
    lat  = tuple[0][1]
    lon  = tuple[0][2]
    alt  = tuple[0][3]

    print(lat)
    print(lon)
    print(alt)

    # Festlegung der Exif-Kameraparameter.
    zeroth_ifd = {
    piexif.ImageIFD.Make: u"Renkforce",
    piexif.ImageIFD.Software: u"piexif"
    }

    # Festlegung des Exif-Erstell-Datums.
    exif_ifd = {
    piexif.ExifIFD.DateTimeOriginal: \
    str(datetime.datetime.now().strftime("%Y:%m:%d %H:%M:%S")),
    piexif.ExifIFD.LensSpecification: \
    ((1, 1), (1, 1), (1, 1), (1, 1)),
    }

    # Festlegung der Exif-Geokoordinaten.
    gps_ifd = {
    piexif.GPSIFD.GPSLatitudeRef: ChooseLatRef(lat),  # N oder S
    piexif.GPSIFD.GPSLatitude: degToDmsRational(lat), # Breitengrad
    piexif.GPSIFD.GPSLongitudeRef: ChooseLonRef(lon), # E oder W
    piexif.GPSIFD.GPSLongitude: degToDmsRational(lon),# Längengrad
    piexif.GPSIFD.GPSAltitudeRef: 0,
    piexif.GPSIFD.GPSAltitude: (int(abs(alt)*100),100)# Höhe ü. NN
    }

    exif_dict = \
    {"0th":zeroth_ifd, "Exif":exif_ifd, \
    "1st":first_ifd, "GPS":gps_ifd}
    # Wandelt die Exif-Daten in bytes um.
    exif_bytes = piexif.dump(exif_dict)
```

```
            filename = 'D:/UAV-Befliegungen/04_ODM/images/' \
            + str(timestamp).replace(".", "_") + '.jpg'
            # Speichern des Einzelbilds als jpg-Datei.
            cv2.imwrite(filename, frame)

            # Öffnen der soeben gespeicherten jpg-Datei mit Pillow.
            im = Image.open(filename)
            # Erneutes Speichern jpg-Datei unter Anfügung der Exif-Daten.
            im.save(filename, exif=exif_bytes)

        else:
            pass

# Schließt die Video-Datei.
vc.release()
```

08 extract_imgs_and_add_pp_coords

```
import cv2
import datetime
import math
import piexif
import time
from PIL import Image

start = time.time()

timestamps_pos_dictio = {}
timestamps_vid_list = []
timestamps_vid_dictio = {}

# Angabe vom Projektverzeichnis
root = "D:/UAV-Befliegungen/210904/"

# CSV-Datei mit den korrigierten GNSS-Koordinaten
# und den zugehörigen Videozeitpunkten.
csv_file_read = root + "02_logs/00000052_2.csv"

with open(csv_file_read, "r") as csv_file_read_opened:
    # Kopfzeile überspringen
    for row in csv_file_read_opened.readlines()[1:]:
        # Zeileninhalt (Datum, Zeit, Koordinaten etc.) werden vereinzelt
        # und als separate Elemente in einer Liste gespeichert.
        row_content = row.strip("\n").split(";")
        # Der Wert für die Postionslösungsgüte wird
        # zur Ganzzahl umgewandelt.
        Q = int(row_content[5])
        # Nur Positionslösungen mit Q=2 (float) oder Q=1 (fix)
        # werden weiterverarbeitet.
        # TODO: Q=1 (fix) ist anzustreben.
        if Q < 3:
            # Extraktion des der Positionslösung
            # zugeordneten Videozeitpunkts.
            videotime = row_content[-2]
            # Nur Positionslösungen ab Videostartzeitpunkt
            # werden übernommen.
            if videotime != "NULL" and float(videotime) >= 0:
                secs = float(videotime)
                lat  = float(row_content[2])
                lon  = float(row_content[3])
                # alt  = float(row_content[4])
                alt  = float(row_content[-1])

                # Einer Dictionary wird der Videozeitpunkt als Schlüssel
                # und die Koordinaten als Werte hinzugefügt.
                timestamps_pos_dictio[secs] = [lat, lon, alt]

# Konstruktor der OpenCV-Klasse VideoCapture. Öffnet und liest die Video-Datei.
```

```
vc = cv2.VideoCapture(root + "03_videos/MOVI0026.mov")

# Abfrage von Videoeigenschaften.
fps     = vc.get(cv2.CAP_PROP_FPS)
frames  = int(vc.get(cv2.CAP_PROP_FRAME_COUNT))
seconds = frames/fps
width   = vc.get(cv2.CAP_PROP_FRAME_WIDTH)
height  = vc.get(cv2.CAP_PROP_FRAME_HEIGHT)

# Zählvariable: Zur Festlegung der Bildexport-Frequenz
counter = 0

# Wenn Video-Datei geöffnet.
if vc.isOpened():
    # Wenn Video-Datei geöffnet ist, wird auf das nächste Bild des Videos
    # zugegriffen. Falls ein Bild zugreifbar ist, wird ein Tupel mit
    # rval(return value)=True und das beteffende Bild (frame)
    # zurückgegeben, andernfalls ein Tupel mit rval=False und frame=False.
       rval, frame = vc.read()
else:
    # Wenn Video-Datei nicht geöffnet ist.
    rval = False # rval(return value)=False
    print("cannot open videofile")

timestamplist_vid = []
# Schleifendurchlauf nur, wenn Bild im Zugriff.
print("1st frame iteration")
while rval:
    # s. oben
    rval, frame = vc.read()
    # Zählvariable wird um 1 erhöht.
    counter += 1
    # Kommandos nach Verzweigung werden nur bei jedem 10. Bild ausgeführt.
    if counter % 10 == 0:
        # Der Zeitstempel des Bilds wird aus der Division
        # von Bildnummer und Bildrate errechnet.
        timestamp = round(counter/fps, 3)
        print("time stamp current frame: ", timestamp)
        # Zeitstempel wird der Liste hinzugefügt.
        timestamps_vid_list.append(timestamp)

# Schließt die Video-Datei.
vc.release()
print("timestamps_vid_list finished")

# print(timestamps_pos_dictio)
# print(timestamps_vid_list)

# Schleife durch die Dictionary mit den Videozeitpunkten als Schlüssel
# und den korrigierten Koordinaten als Werte.
for timestamp_pos in timestamps_pos_dictio:
    # Schleife durch die Liste mit den Zeitstempeln der Standbilder.
```

```
    for timestamp_vid in timestamps_vid_list:
        # Berechnung der Differenz (Betrag) des Videozeitpunkts aus der
        # Dictionary (mit zugeordneten GNSS-Koordinaten) und des aktuell
        # iterierten Videozeitpunkts eines Einzelbilds.
        result = abs(timestamp_vid - timestamp_pos)
        # Kommandos nach Verzweigung werden nur ausgeführt, wenn der
        # Differenzbetrag kleiner als ein vorgegebener Schwellenwert ist.
        if result < 0.041:
            print("timestamp_pos: " + str(timestamp_pos))
            print("timestamp_vid: " + str(timestamp_vid))
            print("abs(timestamp_vid - timestamp_pos): " + str(result))
            # Der Dictionary wird als Schlüssel der Videozeitpunkt des
            # aktuell iterierten Einzelbilds und als Wert die Liste
            # mit den korrigierten Koordinaten hinzugefügt.
            timestamps_vid_dictio[timestamp_vid] = \
            timestamps_pos_dictio[timestamp_pos]

# Gibt 'N' (Nordhalbkugel) zurück, wenn Breitengrad >= 0°,
# sonst 'S' (Südhalbkugel).
def ChooseLatRef(degree):
    if degree >= 0:
        return 'N'
    else:
        return 'S'

# Gibt 'E' (Osten) zurück, wenn Längengrad >= 0°, sonst 'W' (Westen).
def ChooseLonRef(degree):
    if degree >= 0:
        return 'E'
    else:
        return 'W'

# Wandelt einen dezimalen Gradwert in Grad, Minuten und Sekunden um.
def degToDmsRational(degFloat):
    minFloat = degFloat % 1 * 60
    secFloat = minFloat % 1 * 60
    deg = math.floor(degFloat)
    min = math.floor(minFloat)
    sec = int(secFloat * 100)
    return ((deg, 1), (min, 1), (sec, 100))

# Konstruktor der OpenCV-Klasse VideoCapture.
# Öffnet und liest die Video-Datei ein 2. Mal.
vc = cv2.VideoCapture(root + "03_videos/MOVI0026.mov")

# Zählvariable: Zur Festlegung der Bildexport-Frequenz

counter = 0

if vc.isOpened():
    # Wenn Video-Datei geöffnet ist, wird auf das nächste Bild des Videos
    # zugegriffen. Falls ein Bild zugreifbar ist, wird ein Tupel mit
```

```
    # rval(return value)=True und das beteffende Bild (frame)
    # zurückgegeben, andernfalls ein Tupel mit rval=False und frame=False.
       rval, frame = vc.read()
else:
    # Wenn Video-Datei nicht geöffnet ist.
    rval = False # rval(return value)=False

print("2nd frame iteration")
while rval:
    # s. oben
    rval, frame = vc.read()
    # Zählvariable wird um 1 erhöht.
    counter += 1
    # Kommandos nach Verzweigung werden nur bei jedem 10. Bild ausgeführt.
    if counter % 10 == 0:

        # Der Zeitstempel des Bilds wird aus der Division
        # von Bildnummer und Bildrate errechnet.
        # timestamp = round(counter1/fps, 3)
        timestamp = round(counter/fps, 3)
        print(timestamp)
        # Wenn der Zeitstempel des Bildes in der oben angelegten Dictionay
        # als Schlüssel enthalten ist, werden die Kommandos nach der
        # Verzweigung (die Geokodierung) ausgeführt.
        if timestamp in list(timestamps_vid_dictio.keys()):
            print(timestamp)

            # Abfrage der Koordinaten
            lat  = timestamps_vid_dictio[timestamp][0]
            lon  = timestamps_vid_dictio[timestamp][1]
            alt  = timestamps_vid_dictio[timestamp][2]

            print(lat)
            print(lon)
            print(alt)

            # Festlegung der Exif-Kameraparameter.
            zeroth_ifd = {
            piexif.ImageIFD.Make: u"Renkforce",
            piexif.ImageIFD.XResolution: (3840, 1),
            piexif.ImageIFD.YResolution: (2160, 1),
            piexif.ImageIFD.Software: u"piexif"
            }

            # Festlegung des Exif-Erstell-Datums.
            exif_ifd = {
            piexif.ExifIFD.DateTimeOriginal: \
            str(datetime.datetime.now().strftime("%Y:%m:%d %H:%M:%S")),
            piexif.ExifIFD.LensSpecification: \
            ((1, 1), (1, 1), (1, 1), (1, 1)),
            }
```

```
            # Festlegung der Exif-Geokoordinaten.
            gps_ifd = {
            piexif.GPSIFD.GPSLatitudeRef: ChooseLatRef(lat),  # N oder S
            piexif.GPSIFD.GPSLatitude: degToDmsRational(lat), # Breitengrad
            piexif.GPSIFD.GPSLongitudeRef: ChooseLonRef(lon), # E oder W
            piexif.GPSIFD.GPSLongitude: degToDmsRational(lon),# Längengrad
            piexif.GPSIFD.GPSAltitudeRef: 0,
            piexif.GPSIFD.GPSAltitude: (int(abs(alt)*100),100)# Höhe ü. NN
            }

            exif_dict = \
            {"0th":zeroth_ifd, "Exif":exif_ifd, \
            "1st":first_ifd, "GPS":gps_ifd}
            # Wandelt die Exif-Daten in bytes um.
            exif_bytes = piexif.dump(exif_dict)

            # Festlegung des Namens der Ausgabe-jpg-Datei.
            # Der Name entspricht dem Videozeitstempel des Einzelbilds.
            filename = \
            root + '04_ODM/images/' + \
            str(timestamp).replace(".", "_") + '.jpg'
            # Speichern des Einzelbilds als jpg-Datei.
            cv2.imwrite(filename, frame)

            # Öffnen der soeben gespeicherten jpg-Datei mit Pillow.
            im = Image.open(filename)
            # Erneutes Speichern jpg-Datei unter Anfügung der Exif-Daten.
            im.save(filename, exif=exif_bytes)

        else:
            # nur bei jedem 20. Bild
            if counter % 20 == 0:
                # Wenn der Zeitstempel des Bildes in der oben angelegten
                # Dictionay nicht als Schlüssel enthalten ist, wird keine
                # Geokodierung vorgenommen aber trotzdem ohne Exif-Infomation
                # als Standbild exportiert.

                # Festlegung des Namens der Ausgabe-jpg-Datei.
                # Der Name entspricht dem Videozeitstempel des Einzelbilds.
                filename = \
                root + '04_ODM_5/images/' + \
                str(timestamp).replace(".", "_") + '.jpg'
                # Speichern des Einzelbilds als jpg-Datei.
                cv2.imwrite(filename, frame)


# Schließt die Video-Datei.
vc.release()

end = time.time()
print("finished after :", end-start, " seconds")
```

Anhang II Python-Code - Klassen

09 CalculateFlightPath

```
class CalculateFlightPath():
    # Festlegung der Kamerakonstanten
    def __init__ (self, sensor_width, image_width, focal_length):
        # Sensorweite (mm)
        self.sensor_width = sensor_width
        # Bildweite (px)
        self.image_width  = image_width
        # Brennweite  (mm)
        self.focal_length = focal_length

    # Gibt die Flughöhe zurück
    def calculate_flight_altitude(self, GSD):
        # Berechnung der Flughöhe, abhängig von der gewählten GSD
        self.flight_altitude = \
        (self.image_width * GSD * self.focal_length)\
         / (self.sensor_width * 1000)
        # Rückgabe der Flughöhe
        return int(self.flight_altitude)

    # Gibt den Fluglinienabstand zurück
    def calculate_line_spacing(\
    self, flight_altitude, overlap, angular_aperture):
        import math
        # Flughöhe über Grund (m)
        self.flight_altitude = flight_altitude
        # geplante Querüberlappung (%)
        self.overlap = overlap/100
        # Kamera-Öffnungswinkel (Grad)
        self.angular_aperture = angular_aperture
        # Berechnung der beiden anderen Dreieckswinkel
        self.angle_alpha = (180-self.angular_aperture)/2
        self.angle_beta  = self.angle_alpha
```

```
# Sinus von alpha und beta
self.sin_alpha = math.sin(math.radians(self.angle_alpha))
self.sin_beta  = self.sin_alpha
# Berechnung der langen Dreieckseiten
self.side_a = self.flight_altitude / self.sin_alpha
self.side_b = self.side_a
# Berechnung der Dreiecksbasis und damit der Aufnahmenbreite
# mit dem Kosinussatz
self.baseline = math.sqrt((self.side_a*self.side_a)+\
(self.side_b*self.side_b)\
-(2*self.side_a*self.side_a*\
math.cos((math.radians(self.angular_aperture)))))
# Berechnung des Flugachsenabstands
self.line_spacing = self.baseline * (1 - self.overlap)
# Rückgabe des Flugachsenabstands
return int(self.line_spacing)
```

10 LogFile()

```
class LogFile():

    def __init__ (self, logfile):
        self.log = open(logfile, "r")
        self.log_read = self.log.readlines()
        return

    def __del__(self):
        self.log.close()
        del self.log, self.log_read
        return

    # Ordnet den GPS-Zeitstempeln die System-Zeitstempel
    # seit dem Systemstart zu.
    def map_gpstime_to_timeUS(self):
        gpstime_timeus_map = {}
        for row in self.log_read:
            if row[:3] == "GPS":
                row_content = row.split(", ")
                TimeUS = int(row_content[1])
                gpsmicroseconds = int(row_content[3])
                gpstime_timeus_map[gpsmicroseconds] = TimeUS
        return gpstime_timeus_map

    def extract_GPS_from_log(self):
        dictio_GPS = {}
        # message type
        mess_type_ix = 0
        # time since system startup
        TimeUS_ix = 1
        for row in self.log_read:
            if row[:3] == "GPS":
                row_content = row.strip("\n").split(", ")
                TimeUS = row_content[TimeUS_ix]
```

```
                dictio_GPS[TimeUS] = row_content[TimeUS_ix + 1:]
        return dictio_GPS

# Ordnet den UBX-RXM-RAWX-Nachrichten die System-Zeitstempel
# seit dem Systemstart zu.
def extract_GRXH_and_GRXS_from_logfile(self):
    # Dictionary zur Speicherung von Systemzeiten als Schlüssel
    # und UBX-RXM-RAWX-Nachrichten als zuzuordnende Werte.
    dictio_GRXH_GRXS = {}

    # Index des Attributs "message type" (z. B. "GRXH", "GRXS")
    # in der weiter unten aus der zur Log-File-Zeile gewandelten Liste.
    mess_type_ix = 0
    # Index des Attributs "time since system startup" (Systemlaufzeit
    # seit dem Booten) in der weiter unten aus der zur Log-File-Zeile
    # gewandelten Liste.
    TimeUS_ix = 1

    # Schleife durch die Liste mit den eingelesenen Zeilen der
    # Log-Datei.
    for row in self.log_read:
        # Auftrennung der als Zeichenkette gespeicherten und
        # durch Kommata getrennten Teilparameter der Zeile.
        row_content = row.strip("\n").split(", ")

        # Ist die aktuell aufgerufene Zeile eine Nachricht vom  Typ
        # "GRXH", also eine UBX-RXM-RAWX-Header-Message,
        # ist folgende Bedingung erfüllt.
        if row_content[mess_type_ix] == "GRXH":
            # Abfrage der Systemzeit zu der die
            # UBX-RXM-RAWX-Header-Nachricht
            # abgesetzt wurde und Zuweisung zur Variablen TimeUS.
            TimeUS = row_content[TimeUS_ix]
            # Der Dictionary wird als weiteres Element die Systemzeit
            # als Schlüssel und eine leere Liste als Wert hinzugefügt.
            dictio_GRXH_GRXS[TimeUS] = []
```

```
                # Die soeben hinzugefügte leere Liste wird mit den
                # UBX-RXM-RAWX-Header-Komponenten (rcvTow, week, leapS,
                # numMeas, recStat) befüllt.
                dictio_GRXH_GRXS[TimeUS].append(row_content[TimeUS_ix+1:])

            # Ist die aktuell aufgerufene Zeile eine Nachricht vom  Typ
            # "GRXS", also eine UBX-RXM-RAWX-Payload-Message,
            # ist folgende Bedingung erfüllt.
            elif row_content[mess_type_ix] == "GRXS":
                # Abfrage der Systemzeit zu der die
                # UBX-RXM-RAWX-Payload-Nachricht
                # abgesetzt wurde und Zuweisung zur Variablen TimeUS.
                TimeUS = row_content[TimeUS_ix]
                # Dieser GRXS-Nachricht ist eine GRXH-Nachricht
                # vorgeschaltet, für die in der vorigen Verzweigung
                # bereits ein Eintrag (Systemzeit als Schlüssel und Liste
                # als Wert) in der Dictionary angelegt wurde. Der Schlüssel
                # TimeUS ist hier identisch und wird zum Aufruf der
                # zugehörigen Liste verwendet. Die Liste wird dann mit den
                # UBX-RXM-RAWX-Header-Komponenten
                # (rcvTow, week, leapS, numMeas, recStat) befüllt.
                dictio_GRXH_GRXS[TimeUS].append(row_content[TimeUS_ix+1:])

        # Rückgabe der Dictionary mit den Systemzeiten als Schlüssel
        # und Listen mit GRXH und GRXS-Messages als Werte.
        return dictio_GRXH_GRXS

    def get_timeUS_of_video_start_and_stop(self, startcmd, stopcmd):
        # z. B. "Mission: 7 RepeatRelay"
        startcmd = startcmd
        # z. B. "Reached command #12"
        stopcmd  = stopcmd
        for row in self.log_read:
            if row[:3] == 'MSG':
                items = row.split(", ")
                if startcmd in items[2]:
```

```
                starttime = int(items[1])
            if stopcmd in items[2]:
                stoptime = int(items[1])

    return starttime, stoptime
```

11 RINEX_file

```
class RinexFile():

    def __init__ (self, rinexfile, GRXH_GRXS_values):
        self.rinexfile = rinexfile
        self.GRXH_GRXS_values = GRXH_GRXS_values
        return

    def __del__(self):
        del self.rinexfile, self.GRXH_GRXS_values
        return

    def create_RINEX_header(self):
        # header-variables
        RINEX_VERSION =    "3.03"
        FILETYPE =         "OBSERVATION DATA"
        SATELLITE_SYSTEM = "M (MIXED)"
        PGM =              "RINEX_CREATOR_BY_STRATMANN"
        RUN_BY =           "STRATMANN"
        DATE =             "STRATMANN"
        MARKER_NAME =      "UBLOX_M8T_ON_QUADROCOPTER"

        rinex_headerlinestring = \
        "     3.05           OBSERVATION DATA    M (MIXED)           " + \
        "RINEX VERSION / TYPE" + "\n" + \
        "RINEX_CREATOR_BY_STRATMANN  STRATMANN       20210206 134313 " + \
        "LCL PGM / RUN BY / DATE" + "\n" + \
        "UBLOX_M8T_ON_QUADROCOPTER                                   " + \
        "MARKER NAME" + "\n" + \
        "                                                            " + \
        "OBSERVER / AGENCY" + "\n" + \
        "3022589             SEPT POLARX5        5.3.2               " + \
        "REC # / TYPE / VERS" + "\n" + \
        "727178              LEIAR25.R4      LEIT                    " + \
        "ANT # / TYPE" + "\n" + \
```

```
        "  3928425.8707   588854.8559  4973725.6959                  " + \
        "APPROX POSITION XYZ" + "\n" + \
        "        0.0000        0.0000        0.0000                  " + \
        "ANTENNA: DELTA H/E/N" + "\n" + \
        "G    4 C1C L1C D1C S1C                                      " + \
        "SYS / # / OBS TYPES" + "\n" + \
        "R    4 C1C L1C D1C S1C                                      " + \
        "SYS / # / OBS TYPES" + "\n" + \
        "DBHZ                                                        " + \
        "SIGNAL STRENGTH UNIT" + "\n" + \
        "     1                                                      " + \
        "INTERVAL" + "\n" + \
        "2021    04    30    12    03   55.0000000     GPS          " + \
        "TIME OF FIRST OBS" + "\n" + \
        "2021    04    30    12    05   36.0000000     GPS          " + \
        "TIME OF LAST OBS" + "\n" + \
        "                                                            " + \
        "GLONASS COD/PHS/BIS" + "\n" + \
        "    18    18  1929     7                                   " + \
        "LEAP SECONDS" + "\n" + \
        "                                                            " + \
        "END OF HEADER" + "\n"

        return rinex_headerlinestring

    def create_RINEX_body(self):
        from moduls.time_format_converter import TimeFormatConverter as TFC

        # Der gesamt Body wird als Zeichenkette gespeichert
        # und besteht am Anfang aus 0 Zeichen.
        rinex_body_linestring = ""

        # Indizes der Elemente der UBX-RXM-RAWX-Header-Nachrichten (GRXH)
        # receiver TimeOfWeek measurement
        rcvTime_ix = 0
        # GPS week
```

```
week_ix = 1
# GPS leap seconds
leapS_ix = 2
# number of space-vehicle measurements to follow
numMeas_ix = 3
# receiver tracking status bitfield
recStat_ix = 4

# Indizes der Elemente der UBX-RXM-RAWX-Payload-Nachrichten (GRXS)
# Pseudorange measurement
prMes_ix = 0
# Carrier phase measurement
cpMes_ix = 1
# Doppler measurement
doMes_ix = 2
# GNSS identifier
gnss_ix = 3
# Satellite identifier
sv_ix = 4
# GLONASS frequency slot
freq_ix = 5
# carrier phase locktime counter
lock_ix = 6
# carrier-to-noise density ratio
cno_ix = 7
# estimated pseudorange measurement standard deviation
prD_ix = 8
# estimated carrier phase measurement standard deviation
cpD_ix = 9
# estimated Doppler measurement standard deviation
doD_ix = 10
# tracking status bitfield
trk_ix = 11

gnssId_dictio = {
0 : "G", # GPS
```

```
1 : "S", # SBAS
2 : "E", # Galileo
3 : "C", # BeiDou
4 : "I", # IRNSS
5 : "J", # QZSS
6 : "R"  # GLONASS
}

# Schleife durch die Dictionary in der die UBX-RXM-RAWX-Nachrichten
# den System-Zeitstempeln zugeordnet sind. Es werden die Werte
# (values), also die Listen mit den GRXH- bzw. GRXS-Nachrichten
# aufgerufen.
for value in self.GRXH_GRXS_values.values():

    # Das erste Element der Liste, die einer Systemzeit zugeordnet
    # ist, ist immer die GRXH-message.
    GRXH_message = value[0]

    # Abfrage der GPS-Woche, die in der GRXH-message enthalten ist,
    # und Umwandlung in eine Ganzzahl.
    gpsweek = int(GRXH_message[week_ix])

    # Abfrage der Empfänger-Zeit (Sekunde der GPS-Woche),
    # die in der GRXH-message enthalten ist,
    # und Umwandlung in eine Fließkommazahl.
    gpsseconds = float(GRXH_message[rcvTime_ix])

    # Abfrage des GPS-Schaltsekunden-Werts, der in der GRXH-message
    # enthalten ist, und Umwandlung in eine Ganzzahl.
    # TODO: Klären warum Leapseconds nicht nötig sind
    # leapseconds = int(GRXH_message[leapS_ix])
    leapseconds = 0

    # Abfrage der in der GRXH-message enthaltenen Anzahl der als
    # GRXS-message folgenden Satellitenbeobachtungen
    # und Umwandlung in eine Ganzzahl.
```

```
numMeas = int(GRXH_message[numMeas_ix])

# Rechnet die GPS-Zeitgrößen GPS-Woche, GPS-Sekunden und
# Schaltsekunden in die UTC-Zeit (Datum und Uhrzeit in
# Mikrosekundengenauigkeit) um und wandelt diese
# in eine Zeichenkette um.
observation_date_str = \
TFC.weeks_and_seconds_to_utc(gpsweek, gpsseconds, leapseconds)

# Verkettung des Record identifiers (">"),
# der UTC-Zeit (Jahr, Monat, Tag, Stunde, Minute, Sekunde),
# der Epoch-Flag (0:OK, 1: power failure between previous and
# current epoch, >1:Special event) und der Anzahl der in der
# aktuellen Epoche beobachteten Satelliten.
rinex_body_linestring += "> " + str(observation_date_str) \
+ " 0 " + str(numMeas) + "\n"

# Das zweite und alle folgenden Elemente der Liste, die einer
# Systemzeit zugeordnet sind, sind immer GRXS-messages.
GRXS_messages = value[1:]

# Sequenzielles Aufrufen aller GRXS-messages
# in der der Systemzeit zugeordneten Liste.
for entry in GRXS_messages:

    # Abfrage der gnssid (z. B. 0=G=GPS; 6=R=GLONASS)
    # und Umwandlung in eine Ganzzahl
    gnssId = int(entry[gnss_ix])

    # Abfrage der ID des Satelliten
    # und Umwandlung in eine Ganzzahl um Bedingung zu prüfen.
    if int(entry[sv_ix]) <= 9:
        # Voranstellen einer führenden 0
        svId = "0" + entry[sv_ix]
    else:
        svId = entry[sv_ix]
```

```
# Konstruktion der Satellitennummer durch Verkettung
# der GNSS-Kennung (z. B. G=GPS; R=GLONASS)
# und der Satellitenummer (z. B. 27). Beispiel: G27.
sat_number = gnssId_dictio[gnssId] + svId

# Abfrage der Pseudorange-Messung aus der GRXS-Nachricht,
# Umwandlung in eine Fließkommazahl und anschließende
# Umwandlung in eine Zeichenkette, wobei der
# Pseudorange-Wert immer genau 14 Zeichenstellen mit
# 5 Nachkommastellen aufweist. Fehlende Stellen vor dem
# Komma werden mit Nullen aufgefüllt.
prMes = "{:.5f}".format(float(entry[prMes_ix])).zfill(14)

# Abfrage der Trägerphasen-Messung aus der GRXS-Nachricht,
# Umwandlung in eine Fließkommazahl und anschließende
# Umwandlung in eine Zeichenkette, wobei der
# Trägerphasen-Messungs-Wert immer genau 13 Zeichenstellen
# mit 3 Nachkommastellen aufweist. Fehlende Stellen vor dem
# Komma werden mit Nullen aufgefüllt.
cpMes = "{:.3f}".format(float(entry[cpMes_ix])).zfill(13)

# Abfrage der Doppler-Frequenzverschiebung-Messung aus der
# GRXS-Nachricht.
# Umwandlung in eine Fließkommazahl und anschließende
# Umwandlung in eine Zeichenkette, wobei der Wert
# immer genau 3 Nachkommastellen aufweist.
doMes = "{:.3f}".format(float(entry[doMes_ix]))
# Zur rechtsbündigen Ausrichtung der Werte in der
# RINEX-Datei werden der Zeichenketten, antiproportional
# zur Länge, Leerzeichen vorangestellt.
doMes = ((16-len(doMes)) * " ") + doMes

# Abfrage des Signal-Rausch-Verhältnisses aus der
# GRXS-Nachricht,
# Umwandlung in eine Fließkommazahl und anschließende
```

```
                # Umwandlung in eine Zeichenkette, wobei der Wert
                # immer genau 3 Nachkommastellen aufweist.
                cno    = "{:.3f}".format(float(entry[cno_ix]))

                # Verkettung der Satellitennummer, der Pseudorange-Messung,
                # der Trägerphasen-Messung, der Doppler-
                # Frequenzverschiebung und des Signal-Rausch-Verhältnisses
                # zur Beobachtungs-Aufzeichnung.
                rinex_body_linestring += sat_number + "  " + prMes \
                + " " + cpMes + doMes + "          " + cno + "\n"

        return rinex_body_linestring


    def write_RINEX(self):
        rinex = open(self.rinexfile, "w")
        rinex.write(self.create_RINEX_header())
        rinex.write(self.create_RINEX_body())
        rinex.close()
        return
```

12 TimeFormatConverter()

```
class TimeFormatConverter():

    def __init__ (self):
        return

    # Rechnet ein Tupel aus Minuten, Sekunden und Millisekunden
    # in Dezimalsekunden um.
    def clocktime_to_decimalseconds(minutes, seconds, microseconds):
        x1 = float(minutes*60)
        x2 = float(seconds)
        x3 = float(microseconds/1000)
        result = round(x1+x2+x3, 3)
        return result

    # Rechnet Mikrosekunden in Tupel
    # aus Minuten, Sekunden und Millisekunden um.
    def microseconds_to_clocktime(microseconds):

        microseconds_1 = microseconds
        microseconds_2 = microseconds_1 % (60 * 1000000)
        microseconds_3 = microseconds_2 % 1000000
        microseconds_4 = microseconds_3 % 1000

        minutes = int((microseconds_1 - microseconds_2) / (60*1000000))
        seconds = int((microseconds_2 - microseconds_3) / 1000000)
        millis  = int((microseconds_3 - microseconds_4) / 1000)

        return [minutes, seconds, millis]

    # Rechnet die GPS-Zeitgrößen GPS-Woche, GPS-Sekunden und Sprungsekunden
    # in die UTC-Zeit um.
    def weeks_and_seconds_to_utc(gpsweek, gpsseconds, leapseconds):
        #TO_DO: Minuten müssen lt. RINEX-Format 2-stellig sein.
        import datetime, calendar
```

```
    # Festlegung des Datumsformats
    datetimeformat = "%Y  %m  %d  %H %M %S.%f"
    epoch = datetime.datetime.strptime\
    ("1980  1  6  00 00 00.0",datetimeformat)
    elapsed = datetime.timedelta(\
    days=(gpsweek*7),seconds=(gpsseconds+leapseconds))

    return datetime.datetime.strftime(epoch + elapsed, datetimeformat)\
    .lstrip("0").replace(" 0", " ") + "0"

# Rechnet Kalenderdatum und UTC-Zeit in die GPS-Zeitgrößen
# GPS-Woche und GPS-Sekunden um (Beispiel 2174, 457126.59).
def utc_to_week_and_seconds(\
year, month, day, hour, minute, second, microsecond):
    import datetime

    utc_gps_origin = datetime.datetime(year=1980, month=1, day=6)
    utc_gps_log = datetime.datetime(year=year, month=month, day=day, \
    hour=hour, minute=minute, second=second, microsecond=microsecond)

    time_delta = utc_gps_log - utc_gps_origin
    gpsweek = time_delta.days // 7
    days_in_gpsweek = time_delta.days - gpsweek * 7
    gpsseconds = (days_in_gpsweek * 24 * 60 * 60) + time_delta.seconds
    gpsmicroseconds = gpsseconds * 1000 + time_delta.microseconds

    return gpsweek, gpsmicroseconds
```

Anhang III Beispiel-Dateien

13 Beispiel Log-File (1)

...

VIBE, 171175927, 1.741575, 3.589729, 3.122884, 0, 0, 0

CTRL, 171176017, 0.0123546, 0.0174148, 0.01301722, 0.01289344, 0.03509049

IMU, 171197244, 0.0776502, 0.03466254, -0.07255343, 0.2463928, -0.4919422, -9.98427, 0, 0, 35.16879, 1, 1, 999, 999

IMU2, 171197244, 0.07284175, 0.03378487, -0.04690369, -0.1342319, -0.4404037, -10.28084, 0, 0, 0, 1, 1, 752, 1000

IMU, 171237430, -0.02634742, -0.0107482, -0.07875296, 0.1345149, -0.3209772, -10.14561, 0, 0, 35.16899, 1, 1, 999, 999

IMU2, 171237430, -0.02105402, -0.01805229, -0.06038056, -0.1249647, -0.346148, -10.21061, 0, 0, 0, 1, 1, 752, 1000

GPS, 171276359, 3, 457210000, 2174, 20, 0.54, 51.5744974, 8.5237593, 281.78, 1.118, 60.88879, -0.05, 0, 1

GPA, 171276359, 0.85, 0.77, 0.98, 0.22, 0, 1, 171239, 200

POWR, 171276404, 4.771142, 4.768, 0, 6

MAG, 171276429, 132, -185, 713, 166, 85, -140, 0, 0, 0, 1, 171276423

BARO, 171276695, 19.3411, 49110.05, 30.71, -0.5536349, 171276, 0, 34.53544, 1

CTUN, 171276742, 0.1925884, 0.0005063266, 0.1974435, 0.1821743, 19.9, 19.96631, 19.34, NaN, 0, 21.45737, -6, -8, 80

ATT, 171277009, 0.4, 0.88, 5.3, 4.05, 55.13, 58.15, 0.02, 0.29

...

14 Beispiel Log-File (2) - UBX-RXM-RAWX

GRXH, 51004751, 475418.387, 2155, 18, 20, 1

GRXS, 51004751, 16559424.8703267, 87020343.8014486, -1089.069, 0, 29, 0, 47080, 38, 6, 1, 8, 15

GRXS, 51004751, 17469994.8567031, 91805460.1648638, -1357.898, 0, 31, 0, 43120, 34, 7, 2, 8, 15

GRXS, 51004751, 18100806.564472, 95120363.3936535, -3086.781, 0, 25, 0, 8160, 34, 7, 2, 8, 3

GRXS, 51004751, 18185773.000863, 95566883.1855486, 2484.02, 0, 26, 0, 45400, 34, 6, 1, 8, 15

GRXS, 51004751, 18473771.1152352, 97080329.4279995, 3074.613, 0, 18, 0, 7120, 36, 6, 1, 8, 3

GRXS, 51004751, 20045285.8628236, 105338741.483744, 1727.716, 0, 5, 0, 8900, 29, 6, 2, 8, 7

GRXS, 51004751, 15962791.5250651, 85240376.4638322, -2489.498, 6, 13, 5, 44780, 32, 7, 2, 8, 7

GRXS, 51004751, 20826551.0445153, 109444259.707871, -314.4187, 0, 4, 0, 2900, 31, 8, 4, 8, 3

GRXS, 51004751, 15168408.6753441, 81140728.3940282, -450.5369, 6, 23, 10, 47880, 42, 5, 1, 8, 7

GRXS, 51004751, 15987337.2698661, 85221445.2490099, 1918.612, 6, 14, 0, 45660, 42, 5, 1, 8, 15

GRXS, 51004751, 19058547.8098655, 101735782.260389, -4334.253, 6, 22, 4, 11880, 29, 7, 3, 8, 7

GRXS, 51004751, 16500037.6931328, 88233063.2774069, 2671.636, 6, 24, 9, 44300, 41, 5, 1, 8, 15

GRXS, 51004751, 19544094.6914543, 104290959.307393, 276.8088, 6, 6, 3, 0, 26, 9, 15, 10, 1

GRXS, 51004751, 19808115.3409163, 105811328.702667, -4485.792, 6, 12, 6, 47680, 39, 5, 1, 8, 7

GRXS, 51004751, 20800425.3986756, 109306975.171956, 2928.866, 0, 20, 0, 7700, 28, 7, 4, 8, 3

GRXS, 51004751, 19981572.9027036, 106775414.075411, 4034.998, 6, 15, 7, 45680, 40, 5, 1, 8, 15

GRXS, 51004751, 21639623.5233408, 113717005.546757, -3362.735, 0, 2, 0, 11700, 27, 7, 4, 8, 7

GRXS, 51004751, 20351523.1445299, 108790524.484793, -2957.565, 6, 5, 8, 47780, 41, 5, 1, 8, 7

GRXS, 51004751, 21547981.9947596, 113235399.808022, -3802.838, 0, 12, 0, 7840, 27, 7, 5, 8, 3

GRXS, 51004751, 21721790.9448064, 114148781.984053, 1108.375, 0, 9, 0, 12880, 30, 7, 3, 8, 7

15 Beispiel RINEX-Datei

```
     3.05           OBSERVATION DATA    M (MIXED)           RINEX VERSION / TYPE
RINEX_CREATOR_BY_STRATMANN  STRATMANN       20210206 134313 LCL PGM / RUN BY / DATE
UBLOX_M8T_ON_QUADROCOPTER                                   MARKER NAME
                                                            OBSERVER / AGENCY
3022589             SEPT POLARX5        5.3.2               REC # / TYPE / VERS
727178              LEIAR25.R4      LEIT                    ANT # / TYPE
 3928425.8707   588854.8559  4973725.6959                   APPROX POSITION XYZ
       0.0000        0.0000        0.0000                   ANTENNA: DELTA H/E/N
G    4 C1C L1C D1C S1C                                      SYS / # / OBS TYPES
R    4 C1C L1C D1C S1C                                      SYS / # / OBS TYPES
DBHZ                                                        SIGNAL STRENGTH UNIT
    1                                                       INTERVAL
2021    04    30    12    03   55.0000000     GPS          TIME OF FIRST OBS
2021    04    30    12    05   36.0000000     GPS          TIME OF LAST OBS
                                                            GLONASS COD/PHS/BIS
   18    18  1929     7                                    LEAP SECONDS
                                                            END OF HEADER
> 2021  9  10  6 58 43.7900000 0 18
G27  17206566.33480 090421117.258        -37.323         42.000
G10  18194980.62570 095615278.250        927.525         45.000
G08  18345550.10867 096406497.483       1926.893         41.000
G23  18545829.31146 097458967.010      -1518.158         39.000
G16  19123066.18089 100492388.392      -3172.569         40.000
R13  18021256.27390 096232426.532      -1780.262         29.000
G21  20627763.43399 108399614.936       2594.965         36.000
G26  21502668.99690 112997297.271      -3960.232         30.000
```

R22 16831242.65913 089846240.989 -1595.578 29.000

G18 20538449.92151 107930279.108 -3446.120 36.000

R23 18036658.61354 096483938.358 2431.961 39.000

R07 18209530.67884 097477010.723 2488.709 39.000

R06 18405094.84593 098213107.408 -1426.035 30.000

R14 19527303.53498 104091477.985 2288.608 32.000

R12 19760971.15211 105559457.159 -4442.489 31.000

R21 20035990.29812 107216573.081 -3581.076 32.000

G07 21546579.97447 113228036.817 -1294.446 40.000

G30 22153424.12227 116417081.992 210.595 28.000

> 2021 9 10 6 58 44.1900000 0 18

G27 17206569.02214 090421132.273 -37.300 42.000

G10 18194909.93487 095614907.290 927.551 45.000

G08 18345403.43959 096405726.709 1927.002 41.000

G23 18545944.62250 097459574.330 -1518.058 39.000

G16 19123307.72680 100493657.334 -3172.004 40.000

R13 18021389.85532 096233138.895 -1780.788 29.000

G21 20627565.82772 108398577.018 2594.465 36.000

G26 21502972.33404 112998881.261 -3959.734 30.000

R22 16831359.66284 089846865.391 -1595.956 29.000

G18 20538712.37278 107931657.493 -3445.337 36.000

R23 18036476.78227 096482965.724 2431.690 39.000

R07 18209344.72932 097476015.324 2488.933 39.000

R06 18405202.09497 098213677.952 -1425.342 30.000

R14 19527129.10260 104090562.340 2289.407 32.000

R12 19761306.07414 105561234.243 -4442.402 31.000

R21 20036257.87976 107218005.377 -3580.525 32.000

G07 21546678.72525 113228554.696 -1294.471 40.000

G30 22153408.90568 116416997.839 210.575 28.000

16 Beispiel POS-Datei

% program : Emlid Studio 1.0.0 Emlid

% inp file : D:\UAV-Befliegungen\210904_zuHauseNebenan\02_logs\Versuch_211105\00000052.21o

% inp file : D:\UAV-Befliegungen\210904_zuHauseNebenan\02_logs\Versuch_211105\00000052_C_Ephm253.21p

% inp file : D:\UAV-Befliegungen\210904_zuHauseNebenan\02_logs\Versuch_211105\00000052_C_5905253.21o

% obs start : 2021/09/10 06:58:43.8 GPST (week2174 457123.8s)

% obs end : 2021/09/10 07:03:13.8 GPST (week2174 457393.8s)

% ref pos : 51.573927999 8.524680000 279.9999

%

% (lat/lon/height=WGS84/ellipsoidal,Q=1:fix,2:float,3:sbas,4:dgps,5:single,6:ppp,ns=# of satellites)

% GPST	latitude(deg)	longitude(deg)	height(m)	Q	ns	sdn(m)	sde(m)	sdu(m)	sdne(m)	sdeu(m)	sdun(m)	age(s)	ratio
2021/09/10 06:58:43.800	51.574361972	8.524626842	313.1684	2	5	2.4661	2.0246	7.4893	0.3869	0.5961	-2.0057	0.79	0.0
2021/09/10 06:58:45.000	51.574363826	8.524625884	313.4014	2	5	1.8053	1.4779	5.5094	0.2966	0.4496	-1.4823	-0.01	0.0
2021/09/10 06:58:46.200	51.574365865	8.524626351	313.3724	2	5	1.4919	1.2203	4.5485	0.2482	0.3758	-1.2319	0.19	0.0
2021/09/10 06:58:47.400	51.574367078	8.524628302	313.6573	2	5	1.2999	1.0628	3.9489	0.2172	0.3295	-1.0789	0.39	0.0
2021/09/10 06:58:48.600	51.574367148	8.524629818	313.6391	2	5	1.1669	0.9539	3.5257	0.1951	0.2968	-0.9738	0.59	0.0
2021/09/10 06:58:49.800	51.574367707	8.524631196	313.6049	2	5	1.0677	0.8729	3.2042	0.1783	0.2722	-0.8963	0.79	0.0
2021/09/10 06:58:51.000	51.574368633	8.524634003	314.9641	2	5	0.9901	0.8095	2.9462	0.1650	0.2526	-0.8366	-0.01	0.0
2021/09/10 06:58:52.200	51.574368032	8.524634839	317.4389	2	5	0.9272	0.7582	2.7327	0.1541	0.2366	-0.7889	0.19	0.0
2021/09/10 06:58:53.400	51.574367001	8.524633078	320.1329	2	5	0.8749	0.7156	2.5515	0.1449	0.2232	-0.7497	0.39	0.0
2021/09/10 06:58:54.600	51.574367224	8.524632410	322.8446	2	5	0.8305	0.6794	2.3947	0.1371	0.2117	-0.7169	0.59	0.0
2021/09/10 06:58:55.800	51.574367671	8.524632022	325.7658	2	5	0.7922	0.6483	2.2570	0.1302	0.2017	-0.6889	0.79	0.0
2021/09/10 06:58:57.000	51.574367377	8.524632437	328.6077	2	5	0.7587	0.6211	2.1330	0.1242	0.1928	-0.6650	-0.01	0.0
2021/09/10 06:58:58.200	51.574367255	8.524632906	330.9034	2	5	0.7291	0.5970	2.0216	0.1188	0.1849	-0.6440	0.19	0.0
2021/09/10 06:58:59.400	51.574365368	8.524631757	332.2029	2	5	0.7027	0.5756	1.9208	0.1140	0.1779	-0.6254	0.39	0.0
2021/09/10 06:59:00.600	51.574358421	8.524624387	332.4550	2	5	0.6789	0.5563	1.8291	0.1096	0.1715	-0.6087	0.59	0.0
2021/09/10 06:59:01.800	51.574347553	8.524601981	332.3993	2	5	0.6574	0.5388	1.7454	0.1056	0.1657	-0.5936	0.79	0.0
2021/09/10 06:59:03.000	51.574331910	8.524575734	332.1105	2	5	0.6378	0.5229	1.6671	0.1020	0.1604	-0.5801	-0.01	0.0
2021/09/10 06:59:04.200	51.574314300	8.524552024	331.5306	2	5	0.6198	0.5083	1.5948	0.0986	0.1555	-0.5677	0.19	0.0
2021/09/10 06:59:05.400	51.574296839	8.524531055	330.9854	2	5	0.6032	0.4949	1.5280	0.0955	0.1510	-0.5562	0.39	0.0
2021/09/10 06:59:06.600	51.574279576	8.524511144	330.6034	2	5	0.5879	0.4825	1.4662	0.0926	0.1469	-0.5455	0.59	0.0
2021/09/10 06:59:07.800	51.574262018	8.524493159	330.4863	2	5	0.5737	0.4710	1.4090	0.0899	0.1431	-0.5355	0.79	0.0
2021/09/10 06:59:09.000	51.574245156	8.524474695	330.3748	2	5	0.5605	0.4603	1.3545	0.0873	0.1395	-0.5263	-0.01	0.0
2021/09/10 06:59:10.200	51.574228367	8.524456801	330.2750	2	5	0.5481	0.4502	1.3037	0.0849	0.1361	-0.5176	0.19	0.0
2021/09/10 06:59:11.400	51.574211494	8.524438895	330.0774	2	5	0.5365	0.4408	1.2563	0.0827	0.1330	-0.5094	0.39	0.0
2021/09/10 06:59:12.600	51.574194280	8.524421010	329.9520	2	5	0.5257	0.4320	1.2122	0.0805	0.1301	-0.5015	0.59	0.0
2021/09/10 06:59:13.800	51.574177563	8.524401862	329.6344	2	5	0.5154	0.4237	1.1710	0.0785	0.1274	-0.4940	0.79	0.0
2021/09/10 06:59:15.000	51.574160241	8.524382303	329.5216	2	5	0.5057	0.4159	1.1314	0.0766	0.1248	-0.4870	-0.01	0.0

2021/09/10 06:59:16.200	51.574142929	8.524362081	329.5992	2	5	0.4966	0.4084	1.0943	0.0748	0.1224	-0.4803	0.19	0.0
2021/09/10 06:59:17.400	51.574126256	8.524341852	329.5496	2	5	0.4879	0.4014	1.0594	0.0730	0.1201	-0.4738	0.39	0.0
2021/09/10 06:59:18.600	51.574108840	8.524321850	329.4906	2	5	0.4797	0.3947	1.0268	0.0714	0.1180	-0.4676	0.59	0.0
2021/09/10 06:59:19.800	51.574092131	8.524300940	329.4885	2	5	0.4718	0.3884	0.9962	0.0698	0.1159	-0.4616	0.79	0.0
2021/09/10 06:59:21.000	51.574076036	8.524280412	329.6926	2	5	0.4644	0.3823	0.9666	0.0682	0.1140	-0.4559	-0.01	0.0
2021/09/10 06:59:22.200	51.574059160	8.524259828	329.8446	2	5	0.4573	0.3765	0.9387	0.0668	0.1122	-0.4504	0.19	0.0
2021/09/10 06:59:23.400	51.574042288	8.524238405	329.9695	2	5	0.4505	0.3710	0.9124	0.0654	0.1105	-0.4451	0.39	0.0
2021/09/10 06:59:24.600	51.574024363	8.524216817	329.8403	2	5	0.4439	0.3657	0.8876	0.0640	0.1088	-0.4400	0.59	0.0
2021/09/10 06:59:25.800	51.574006438	8.524196199	329.6417	2	5	0.4377	0.3606	0.8644	0.0627	0.1073	-0.4350	0.79	0.0
2021/09/10 06:59:27.000	51.573989444	8.524176679	329.5577	2	5	0.4317	0.3558	0.8418	0.0614	0.1058	-0.4302	-0.01	0.0
2021/09/10 06:59:28.200	51.573972287	8.524155813	329.2003	2	5	0.4260	0.3511	0.8203	0.0602	0.1044	-0.4256	0.19	0.0
2021/09/10 06:59:29.400	51.573958568	8.524138622	329.0263	2	5	0.4205	0.3466	0.8001	0.0590	0.1030	-0.4211	0.39	0.0
2021/09/10 06:59:30.600	51.573955119	8.524136320	328.9584	2	5	0.4152	0.3423	0.7809	0.0579	0.1017	-0.4167	0.59	0.0
2021/09/10 06:59:31.800	51.573964278	8.524132421	329.2345	2	5	0.4101	0.3382	0.7628	0.0568	0.1005	-0.4124	0.79	0.0
2021/09/10 06:59:33.000	51.573977540	8.524121127	329.6527	2	5	0.4051	0.3341	0.7452	0.0557	0.0993	-0.4083	-0.01	0.0
2021/09/10 06:59:34.200	51.573991374	8.524107623	329.9651	2	5	0.4004	0.3303	0.7284	0.0547	0.0982	-0.4043	0.19	0.0
2021/09/10 06:59:35.400	51.574005202	8.524093372	330.1584	2	5	0.3958	0.3265	0.7125	0.0536	0.0971	-0.4004	0.39	0.0
2021/09/10 06:59:36.600	51.574020563	8.524078059	330.1988	2	5	0.3913	0.3229	0.6974	0.0526	0.0961	-0.3966	0.59	0.0
2021/09/10 06:59:37.800	51.574037391	8.524062419	330.1237	2	5	0.3871	0.3194	0.6831	0.0517	0.0951	-0.3929	0.79	0.0
2021/09/10 06:59:39.000	51.574055554	8.524047155	330.0881	2	5	0.3829	0.3161	0.6691	0.0507	0.0941	-0.3893	-0.01	0.0

17 Beispiel CSV-Datei

Date;Time;latitude(deg);longitude(deg);height(m);Q;ns;sdn(m);sde(m);sdu(m);sdne(m);sdeu(m);sdun(m);age(s);ratio

2021/09/10;06:58:43.800;51.574361972;8.524626842;313.1684;2;5;2.4661;2.0246;7.4893;0.3869;0.5961;-2.0057;0.79;0.0

2021/09/10;06:58:45.000;51.574363826;8.524625884;313.4014;2;5;1.8053;1.4779;5.5094;0.2966;0.4496;-1.4823;-0.01;0.0

2021/09/10;06:58:46.200;51.574365865;8.524626351;313.3724;2;5;1.4919;1.2203;4.5485;0.2482;0.3758;-1.2319;0.19;0.0

2021/09/10;06:58:47.400;51.574367078;8.524628302;313.6573;2;5;1.2999;1.0628;3.9489;0.2172;0.3295;-1.0789;0.39;0.0

2021/09/10;06:58:48.600;51.574367148;8.524629818;313.6391;2;5;1.1669;0.9539;3.5257;0.1951;0.2968;-0.9738;0.59;0.0

2021/09/10;06:58:49.800;51.574367707;8.524631196;313.6049;2;5;1.0677;0.8729;3.2042;0.1783;0.2722;-0.8963;0.79;0.0

2021/09/10;06:58:51.000;51.574368633;8.524634003;314.9641;2;5;0.9901;0.8095;2.9462;0.1650;0.2526;-0.8366;-0.01;0.0

2021/09/10;06:58:52.200;51.574368032;8.524634839;317.4389;2;5;0.9272;0.7582;2.7327;0.1541;0.2366;-0.7889;0.19;0.0

2021/09/10;06:58:53.400;51.574367001;8.524633078;320.1329;2;5;0.8749;0.7156;2.5515;0.1449;0.2232;-0.7497;0.39;0.0

2021/09/10;06:58:54.600;51.574367224;8.524632410;322.8446;2;5;0.8305;0.6794;2.3947;0.1371;0.2117;-0.7169;0.59;0.0

2021/09/10;06:58:55.800;51.574367671;8.524632022;325.7658;2;5;0.7922;0.6483;2.2570;0.1302;0.2017;-0.6889;0.79;0.0

2021/09/10;06:58:57.000;51.574367377;8.524632437;328.6077;2;5;0.7587;0.6211;2.1330;0.1242;0.1928;-0.6650;-0.01;0.0

2021/09/10;06:58:58.200;51.574367255;8.524632906;330.9034;2;5;0.7291;0.5970;2.0216;0.1188;0.1849;-0.6440;0.19;0.0

2021/09/10;06:58:59.400;51.574365368;8.524631757;332.2029;2;5;0.7027;0.5756;1.9208;0.1140;0.1779;-0.6254;0.39;0.0

2021/09/10;06:59:00.600;51.574358421;8.524624387;332.4550;2;5;0.6789;0.5563;1.8291;0.1096;0.1715;-0.6087;0.59;0.0

2021/09/10;06:59:01.800;51.574347553;8.524601981;332.3993;2;5;0.6574;0.5388;1.7454;0.1056;0.1657;-0.5936;0.79;0.0

2021/09/10;06:59:03.000;51.574331910;8.524575734;332.1105;2;5;0.6378;0.5229;1.6671;0.1020;0.1604;-0.5801;-0.01;0.0

2021/09/10;06:59:04.200;51.574314300;8.524552024;331.5306;2;5;0.6198;0.5083;1.5948;0.0986;0.1555;-0.5677;0.19;0.0

2021/09/10;06:59:05.400;51.574296839;8.524531055;330.9854;2;5;0.6032;0.4949;1.5280;0.0955;0.1510;-0.5562;0.39;0.0

2021/09/10;06:59:06.600;51.574279576;8.524511144;330.6034;2;5;0.5879;0.4825;1.4662;0.0926;0.1469;-0.5455;0.59;0.0

2021/09/10;06:59:07.800;51.574262018;8.524493159;330.4863;2;5;0.5737;0.4710;1.4090;0.0899;0.1431;-0.5355;0.79;0.0

2021/09/10;06:59:09.000;51.574245156;8.524474695;330.3748;2;5;0.5605;0.4603;1.3545;0.0873;0.1395;-0.5263;-0.01;0.0

2021/09/10;06:59:10.200;51.574228367;8.524456801;330.2750;2;5;0.5481;0.4502;1.3037;0.0849;0.1361;-0.5176;0.19;0.0

2021/09/10;06:59:11.400;51.574211494;8.524438895;330.0774;2;5;0.5365;0.4408;1.2563;0.0827;0.1330;-0.5094;0.39;0.0

2021/09/10;06:59:12.600;51.574194280;8.524421010;329.9520;2;5;0.5257;0.4320;1.2122;0.0805;0.1301;-0.5015;0.59;0.0

2021/09/10;06:59:13.800;51.574177563;8.524401862;329.6344;2;5;0.5154;0.4237;1.1710;0.0785;0.1274;-0.4940;0.79;0.0

2021/09/10;06:59:15.000;51.574160241;8.524382303;329.5216;2;5;0.5057;0.4159;1.1314;0.0766;0.1248;-0.4870;-0.01;0.0

2021/09/10;06:59:16.200;51.574142929;8.524362081;329.5992;2;5;0.4966;0.4084;1.0943;0.0748;0.1224;-0.4803;0.19;0.0

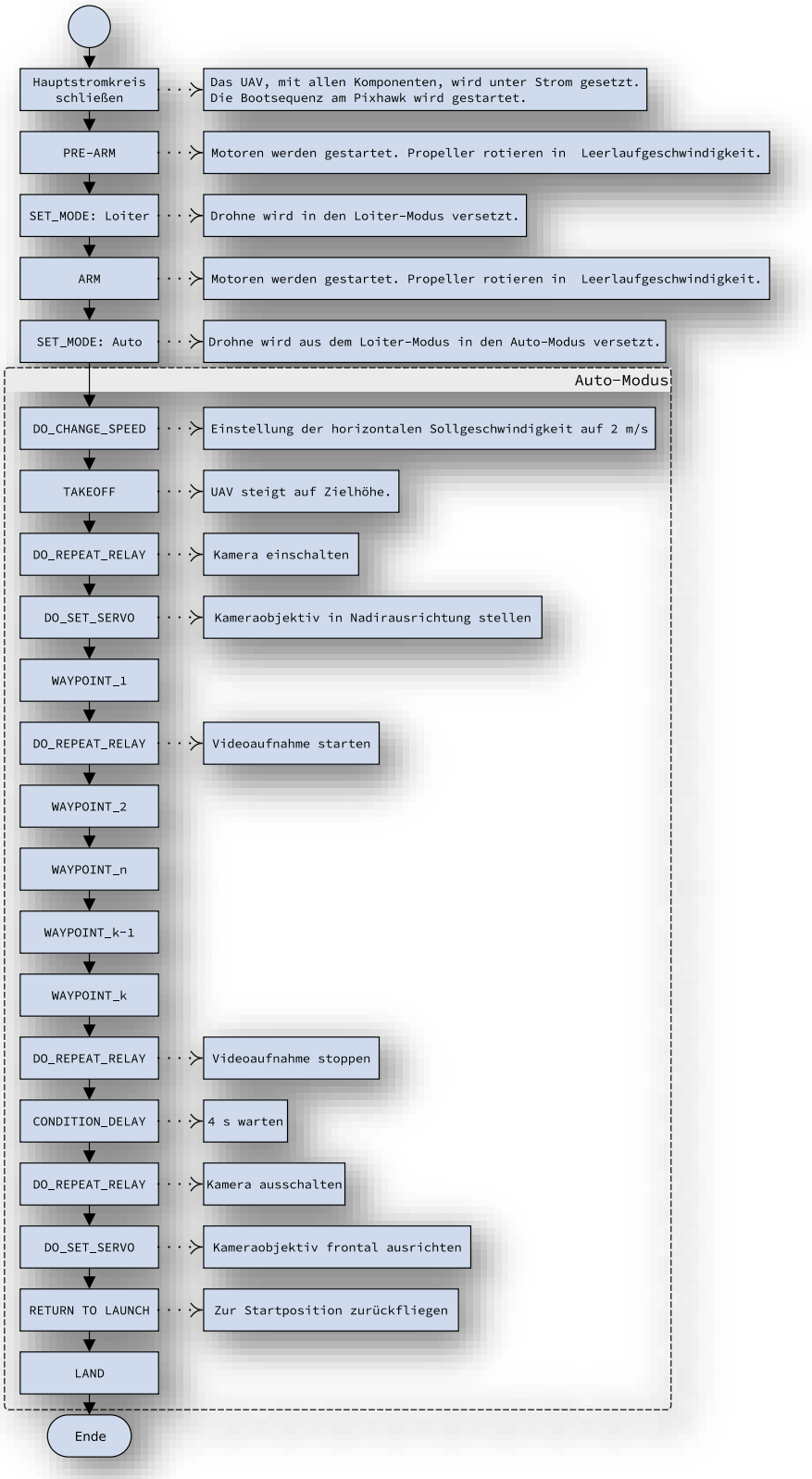
2021/09/10;06:59:17.400;51.574126256;8.524341852;329.5496;2;5;0.4879;0.4014;1.0594;0.0730;0.1201;-0.4738;0.39;0.0

2021/09/10;06:59:18.600;51.574108840;8.524321850;329.4906;2;5;0.4797;0.3947;1.0268;0.0714;0.1180;-0.4676;0.59;0.0

2021/09/10;06:59:19.800;51.574092131;8.524300940;329.4885;2;5;0.4718;0.3884;0.9962;0.0698;0.1159;-0.4616;0.79;0.0

Anhang IV Sonstiges

18 Auto-Modus-Kommoandosequenz



19 Spezifikation u-blox NEO-M8T

Parameter	Specification						
Receiver type	72-channel u-blox M8 engine GPS L1C/A, SBAS L1C/A, QZSS L1C/A, QZSS L1 SAIF, GLONASS L1OF, BeiDou B1, Galileo E1B/C						
	GNSS	GPS & GLONASS	GPS & BeiDou	GPS	GLONASS	BeiDou	Galileo
Time-To-First-Fix	Cold start	25 s	28 s	29 s	30 s	34 s	45 s
	Aided start	2 s	2 s	2 s	2 s	3 s	7 s
	Hot start	1 s	1 s	1 s	1 s	1 s	1 s
Sensitivity	Tracking & Navigation	-167 dBm	-166 dBm	-166 dBm	-166 dBm	-159 dBm	-159 dBm
	Aided acquisition	-157 dBm	-157 dBm	-157 dBm	-151 dBm	-146 dBm	-145 dBm
	Reacquisition	-160 dBm	-160 dBm	-160 dBm	-156 dBm	-156 dBm	-153 dBm
	Cold start	-148 dBm	-148 dBm	-148 dBm	-145 dBm	-143 dBm	-138 dBm
	Hot start	-160 dBm	-160 dBm	-160 dBm	-156 dBm	-155 dBm	-151 dBm
Horizontal position accuracy	Autonomous	2.5 m	2.5 m	2.5 m	4.0 m	3.0 m	N/A
	SBAS	2.0 m	2.0 m	2.0 m	N/A	N/A	N/A
Velocity accuracy		0.05 m/s	0.05 m/s	0.05 m/s	0.05 m/s	0.05 m/s	0.05 m/s
Heading accuracy		0.3°	0.3°	0.3°	0.4°	0.5°	0.5°
Max navigation update rate		4 Hz	4 Hz	10 Hz	10 Hz	10 Hz	10 Hz
Time pulse frequency	0.25 Hz…10 MHz						
Time pulse accuracy	Clear sky	<= 20 ns					
	Indoor	<= 500 ns					
Operational limits	Dynamics	<= 4g					
	Altitude	50000 m					
	Velocity	500 m/s					

20 Komponenten des UBX-RXM-RAWX-Protokolls

Name	Einheit	PX4Log	Beschreibung
rcvTow	s	GRXH	Messung der Wochenzeit in der lokalen Zeit des Empfängers, die annähernd an das GPS-Zeitsystem angepasst ist. Die Informationen über die lokale Wochenzeit des Empfängers, die Wochennummer und die Schaltsekunde können verwendet werden, um die Zeit in andere Zeitsysteme zu übersetzen. Weitere Informationen zu den Unterschieden in den Zeitsystemen finden Sie in der Dokumentation zum RINEX 3-Format. Bei einem Empfänger, der nur im GLONASS-Modus arbeitet, kann die UTC-Zeit durch Subtraktion des Feldes leapS von der GPS-Zeit bestimmt werden, unabhängig davon, ob die GPS-Schaltsekunden gültig sind.
week	Wochen	GRXH	GPS-Wochennummer in der lokalen Zeit des Empfängers.
leapS	s	GRXH	GPS-Schaltsekunden (GPS-UTC). Dieses Feld stellt das beste Wissen des Empfängers über den Schaltsekunden-Offset dar. Im Bitfeld recStat wird ein Flag angegeben, das anzeigt, ob die Schaltsekunden bekannt sind.
numMeas	-	GRXH	Anzahl der folgenden Messungen
recStat	-	GRXH	Empfängerverfolgungsstatus
reserved1	-	-	Reserviert
prMes	M	GRXS	Pseudo-Entfernungsmessung [m]. GLONASS-Zwischenfrequenz-Kanalverzögerungen werden mit einer internen Kalibrierungstabelle kompensiert.
cpMes	Zyklen	GRXS	Trägerphasenmessung [Zyklen]. Die anfängliche Mehrdeutigkeit der Trägerphase wird mit einem Näherungswert initialisiert, damit der Betrag der Phase nahe an der Pseudobereichsmessung liegt. Taktrückstellungen werden sowohl auf die Phasen- als auch auf die Codemessungen gemäß der RINEX-Spezifikation angewendet.
doMes	Hz	GRXS	Doppler-Messung (positives Vorzeichen für sich nähernde Satelliten) [Hz]

gnssId	-	GRXS	GNSS-Bezeichner (siehe Satellitennummerierung für eine Liste von Bezeichnern)
svId	-	GRXS	Satellitenkennung
reserved2	-	-	Reserviert
freqId	-	GRXS	Nur für GLONASS verwendet: Dies ist der Frequenz-Slot + 7 (Bereich von 0 bis 13)
locktime	ms	GRXS	Trägerphasen-Locktime-Zähler (maximal 64500ms)
cno	dBHz	GRXS	Träger-Rausch-Dichte-Verhältnis (Signal Stärke) [dB-Hz]
prStdev	m	GRXS	Geschätzte Pseudo-Entfernungsmessung Standardabweichung
cpStdev	Zyklen	GRXS	Geschätzte Trägerphasenmessung Standardabweichung
doStdev	Hz	GRXS	Geschätzte Dopplermessung Standardabweichung
trkStat	-	GRXS	Tracking-Status-Bitfeld
reserved3	-	-	Reserviert